Realizing Continuity Using Stateful Computations

Liron Cohen
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Vincent Rahli
University of Birmingham, UK

Abstract

The principle of continuity is a seminal property that holds for a number of intuitionistic theories such as System T. Roughly speaking, it states that functions on real numbers only need approximations of these numbers to compute. Generally, continuity principles have been justified using semantical arguments, but it is known that the modulus of continuity of functions can be computed using effectful computations such as exceptions or reference cells. This paper presents a class of intuitionistic theories that features stateful computations, such as reference cells, and shows that these theories can be extended with continuity axioms. The modulus of continuity of the functionals on the Baire space is directly computed using the stateful computations enabled in the theory.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Constructive mathematics

Keywords and phrases Continuity, Stateful computations, Intuitionism, Extensional Type Theory, Constructive Type Theory, Realizability, Theorem proving, Agda

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.15

Supplementary Material *Model (Agda Formalization)*: `https://github.com/vrahli/opentt`

Funding *Liron Cohen*: This research was partially supported by Grant No. 2020145 from the United States-Israel Binational Science Foundation (BSF).

1 Introduction

Continuity is a seminal property in intuitionistic theories which contradicts classical mathematics but is generally accepted by constructivists. Roughly speaking, the principle states that functions on real numbers only need approximations of these numbers to compute. Brouwer, in particular, assumed his so-called *continuity principle for numbers* to derive that all real-valued functions on the unit interval are uniformly continuous [15, 11, 4, 5, 28]. The continuity principle for numbers, sometimes referred to as the weak continuity principle, states that all functions on the Baire space (i.e., $\mathcal{B} \coloneqq \mathsf{Nat}^{\mathsf{Nat}}$) have a modulus of continuity. More concretely, given a function F of type $\mathcal{B} \to \mathsf{Nat}$ and a function α of type $\mathcal{B}$, the principle states that $F(\alpha)$ can only depend on an initial segment of α, and the length of the smallest such segment is the modulus of continuity of F at α. This is standardly formalized as follows, where $\mathcal{B}_n \coloneqq \{x : \mathsf{Nat} \mid x < n\} \to \mathsf{Nat}$ is the set of finite sequences of length n:

$$\mathtt{WCP} = \Pi F{:}\mathcal{B} \to \mathsf{Nat}.\Pi\alpha{:}\mathcal{B}.\downharpoonleft\Sigma n{:}\mathsf{Nat}.\Pi\beta{:}\mathcal{B}.(\alpha{=}\beta{\in}\mathcal{B}_n) \to (F(\alpha){=}F(\beta){\in}\mathsf{Nat})$$

A number of theories have been shown to satisfy Brouwer's continuity principle, or uniform variants, such as N-HA$^\omega$ by Troelstra [29, p.158], MLTT by Coquand and Jaber [8, 9], System T by Escardó [13], CTT by Rahli and Bickford [24], BTT by Baillon, Mahboubi and Pedrot [3], to cite only a few (see Sec. 5 for further details). These proofs often rely on a semantical forcing-based approach [8, 9], where the forcing conditions capture the amount of information needed when applying a function to a sequence in the Baire space, or through suitable models that internalize (C-Spaces in [34]) or exhibit continuous behavior (e.g., dialogue trees in [13, 3]).

Not only can functions on the Baire space be proved to be continuous, but using effectful computations, as for example described in [20], one can *compute* the modulus of continuity of such a function. However, as shown for example by Kreisel [16, p.154], Troelstra [30, Thm.IIA], and Escardó and Xu [12, 33], continuity is not an extensional property in the sense that two equal functions might have different moduli of continuity. Therefore, to realize continuity, the existence of a modulus of continuity has to be truncated as explained, e.g., in [12, 33, 24, 25], which is what the ⇃ operator achieves in `WCP`. Following the effectful approach, continuity was shown to be realizable in [24, 25] using exceptions.

Instead of using exceptions, a more straightforward way to compute the modulus of continuity of a function on the Baire space is to use reference cells. This was explained, e.g., in [20], where the use of references can be seen as the programming counterparts of the more logical forcing conditions. The computation using references is more efficient than when using exceptions as it allows computing the modulus of continuity of a function F at a point α simply by executing F on α, while recording the highest argument that α is applied to, while using exceptions requires repeatedly searching for the modulus of continuity.

Following this line of work, in this paper we show how to use stateful computations to realize a continuity principle. This allows deriving constructive type theories that include continuity axioms where the modulus of continuity is internalized in the sense that it is computed by an expression of the underlying programming language. Concretely, we do so for $\mathrm{TT}_{\mathcal{C}}^{\Box}$ [6], which is a family of extensional type theories parameterized by a type modality $\Box$, and a choice type $\mathcal{C}$, which are presented in more details in Sec. 2. More precisely, we prove in this paper that all $\mathrm{TT}_{\mathcal{C}}^{\Box}$ functions are continuous for some instances of $\Box$ and $\mathcal{C}$: namely for "non-empty" equality modalities, and reference-like stateful choice operators. Our proof is for a variant of the weak continuity principle (see Thm. 13), which we show to be inhabited by a program that relies on a choice operator to keep track of the modulus of continuity of a given function, following Longley's method [20]. This variant is restricted to "pure" functions F, α, and β without side effects, and Sec. 4.1 discusses issues arising with impure functions.

Roadmap. After recalling in Sec. 2 the main aspects of $\mathrm{TT}_{\mathcal{C}}^{\Box}$ that are relevant to the results presented in this paper, Sec. 3 instantiates and extends $\mathrm{TT}_{\mathcal{C}}^{\Box}$ with additional components, which are, in turn, used in Sec. 4 to validate continuity using stateful computations. One key contribution of this paper, discussed in Sec. 3, is the fact that $\mathrm{TT}_{\mathcal{C}}^{\Box}$ now allows computations to modify the current world, which is accounted for in its forcing interpretation. Another key contribution, discussed in Sec. 4, is the internalization of the modulus of continuity of functions, in the sense that it can be computed by a $\mathrm{TT}_{\mathcal{C}}^{\Box}$ expression and used to validate the continuity principle. Finally, Sec. 5 concludes and discusses the related work on continuity.

2 Background

This section recalls $\mathrm{TT}_{\mathcal{C}}^{\Box}$, a family of type theories parameterized by a choice operator $\mathcal{C}$, and a metatheoretical modality $\Box$, which allows typing the choice operators. See [6] for further details. The choice operators are time-progressing elements that we will in particular instantiate with references. Sec. 3 carves out a sub-family for which we can validate computationally relevant continuity rules as shown in Sec. 4.

2.1 Metatheory

Our metatheory is Agda's type theory [1]. The results presented in this paper have been formalized in Agda, and the formalization is available here: `https://github.com/vrahli/opentt/`. We use $\forall, \exists, \wedge, \vee, \rightarrow, \neg$ in place of Agda's logical connectives in this paper. Agda

$$
\begin{array}{llllllll}
v \in \mathsf{Value} ::= vt & \text{(type)} & \mid \lambda x.t & \text{(lambda)} & \mid \star & \text{(constant)} \\
\quad \mid \underline{n} & \text{(number)} & \mid \mathtt{inl}(t) & \text{(left injection)} & \boxed{\mid \ \delta \ \ \text{(choice name)}} & \\
\quad \mid \langle t_1, t_2 \rangle & \text{(pair)} & \mid \mathtt{inr}(t) & \text{(right injection)} & & \\
vt \in \mathsf{Type} ::= \mathbf{\Pi} x{:}t_1.t_2 & \text{(product)} & \mid \{x : t_1 \mid t_2\} & \text{(set)} & \mid t_1{+}t_2 & \text{(disjoint union)} \\
\quad \mid \mathbf{\Sigma} x{:}t_1.t_2 & \text{(sum)} & \mid t_1{=}t_2{\in}t & \text{(equality)} & \boxed{\mid \ \rotatebox[origin=c]{-90}{$\rightsquigarrow$} t \ \ \text{(time truncation)}} & \\
\quad \mid \mathbb{U}_i & \text{(universe)} & \mid \mathsf{Nat} & \text{(numbers)} & & \\
t \in \mathsf{Term} ::= x & \text{(variable)} & \mid v & \text{(value)} & \boxed{\mid \ !t \ \ \text{(read)}} & \\
\quad \mid t_1 \ t_2 & \text{(application)} & \mid \mathtt{fix}(t) & \text{(fixpoint)} & \mid \mathtt{let}\ x, y = t_1\ \mathtt{in}\ t_2 & \text{(pair destructor)} \\
\multicolumn{4}{l}{\quad \mid \mathtt{case}\ t\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 \qquad \text{(injection destructor)}} & & \\
\end{array}
$$

$$
\begin{array}{lll}
(\lambda x.t)\ u \ \mapsto_{\boxed{w}} \ t[x\backslash u] & & \mathtt{let}\ x, y = \langle t_1, t_2 \rangle\ \mathtt{in}\ t \mapsto_{\boxed{w}} t[x\backslash t_1; y\backslash t_2] \\
\mathtt{fix}(v) \ \mapsto_{\boxed{w}} \ v\ \mathtt{fix}(v) & & \mathtt{case}\ \mathtt{inl}(t)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 \mapsto_{\boxed{w}} t_1[x\backslash t] \\
\boxed{!\delta \mapsto_w \mathsf{choice?}(w, \delta)} & & \mathtt{case}\ \mathtt{inr}(t)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 \mapsto_{\boxed{w}} t_2[y\backslash t] \\
\end{array}
$$

Figure 1 Core syntax (above) and small-step operational semantics (below).

provides an hierarchy of types annotated with universe labels which we omit for simplicity. Following Agda's terminology, we refer to an Agda type as a *set*, and reserve the term *type* for $\mathrm{TT}_{\mathcal{C}}^{\square}$'s types. We use $\mathbb{P}$ as the type of sets that denote propositions; $\mathbb{N}$ for the set of natural numbers; and $\mathbb{B}$ for the set of Booleans true and false. We use induction-recursion to define the forcing interpretation in Sec. 3.2, where we use function extensionality to interpret universes. We do not discuss this further here and the interested reader is referred to forcing.lagda in the Agda code for further details.

2.2 Worlds

To capture the time progression notion which underlines choice operators, $\mathrm{TT}_{\mathcal{C}}^{\square}$ is parameterized by a Kripke frame [18, 19] defined as follows:

▶ **Definition 1** (Kripke Frame). *A Kripke frame consists of a set of* worlds $\mathcal{W}$ *equipped with a reflexive and transitive binary relation* $\sqsubseteq$.

Let w range over $\mathcal{W}$. We sometimes write $w' \sqsupseteq w$ for $w \sqsubseteq w'$. Let $\mathcal{P}_w$ be the collection of predicates on world extensions, i.e., functions in $\forall w' \sqsupseteq w.\mathbb{P}$. Note that due to $\sqsubseteq$'s transitivity, if $P \in \mathcal{P}_w$ then for every $w' \sqsupseteq w$ it naturally extends to a predicate in $\mathcal{P}_{w'}$. We further define the following notations for quantifiers. $\forall_w^{\sqsubseteq}(P)$ states that $P \in \mathcal{P}_w$ is true for all extensions of w, i.e., $P\ w'$ holds in all worlds $w' \sqsupseteq w$. $\exists_w^{\sqsubseteq}(P)$ states that $P \in \mathcal{P}_w$ is true at an extension of w, i.e., $P\ w'$ holds for some world $w' \sqsupseteq w$. For readability, we sometime write $\forall_w^{\sqsubseteq}(w'.P)$ (or $\exists_w^{\sqsubseteq}(w'.P)$) instead of $\forall_w^{\sqsubseteq}(\lambda w'.P)$ (or $\exists_w^{\sqsubseteq}(\lambda w'.P)$), respectively.

2.3 $\mathrm{TT}_{\mathcal{C}}^{\square}$'s Syntax and Operational Semantics

Fig. 1 recalls $\mathrm{TT}_{\mathcal{C}}^{\square}$'s syntax and operational semantics, where the blue boxes highlight the time-related components, and where x belongs to a set of variables Var. For simplicity, numbers are considered to be primitive. The constant $\star$ is there for convenience, and is used in place of a term, when the particular term used is irrelevant. Terms are evaluated according to the operational semantics presented in Fig. 1's lower part. In what follows, we use all letters as metavariables for terms. Let $t[x\backslash u]$ stand for the capture-avoiding substitution of all the free occurrences of x in t by u.

Types are syntactic forms that are given semantics in Sec. 3.2 via a forcing interpretation. The type system contains standard types such as dependent products of the form $\mathbf{\Pi}x{:}t_1.t_2$ and dependent sums of the form $\mathbf{\Sigma}x{:}t_1.t_2$. For convenience we write $t_1 \rightarrow t_2$ for the non-dependent $\mathbf{\Pi}$ type; `True` for $\underline{0}$=$\underline{0}$∈Nat; `False` for $\underline{0}$=$\underline{1}$∈Nat; and $\neg T$ for $(T \rightarrow$ `False`$)$.

Fig. 1's lower part presents $\text{TT}_{\mathcal{C}}^{\square}$'s small-step operational semantics, where $t_1 \mapsto_w t_2$ expresses that t_1 reduces to t_2 in one step of computation *w.r.t. the world* w. We omit the congruence rules that allow computing within terms such as: if $t_1 \mapsto_w t_2$ then $t_1(u) \mapsto_w t_2(u)$. We denote by $\Downarrow$ the reflexive transitive closure of $\mapsto$, i.e., $a \Downarrow_w b$ states that a computes to b in $\geq$0 steps. We also write $a \Downarrow\!\!\Downarrow_w b$ if a computes to b in all extensions of w, i.e., if $\forall_w^{\sqsubseteq}(w'.a \Downarrow_{w'} b)$. We write $\sim_w$ for the symmetric and transitive closure of $\Downarrow_w$.

$\text{TT}_{\mathcal{C}}^{\square}$ includes time-progressing notions that rely on worlds to record choices and provides operators to access the choices stored in a world, which we now recall. Choices are referred to through their names. A concrete example of such choices are reference cells in programming languages, where a variable name pointing to a reference cell is the name of the corresponding reference cell. To this end, $\text{TT}_{\mathcal{C}}^{\square}$'s computation system is parameterized by a set $\mathcal{N}$ of *choice names*, that is equipped with a decidable equality, and an operator that given a list of names, returns a name not in the list. This can be given by, e.g., nominal sets [23]. In what follows we let δ range over $\mathcal{N}$, and take $\mathcal{N}$ to be $\mathbb{N}$ for simplicity. $\text{TT}_{\mathcal{C}}^{\square}$ is further parameterized over abstract operators and properties recalled in Defs. 2 and 4–6, which we show how to instantiate in Ex. 7. Definitions such as Def. 2 provide axiomatizations of operators, and in addition informally indicate their intended use. Choices are defined abstractly as follows:

▶ **Definition 2** (Choices). *Let* $\mathcal{C} \subseteq$ Term *be a set of* choices,[1] *and let* κ *range over* $\mathcal{C}$. *We say that a computation system contains* $\langle\mathcal{N},\mathcal{C}\rangle$*-choices if there exists a partial function* choice? $\in \mathcal{W} \rightarrow \mathcal{N} \rightarrow \mathcal{C}$*. Given* $w \in \mathcal{W}$ *and* $\delta \in \mathcal{N}$*, the returned choice, if it exists, is meant to be the last choice made for* δ *according to* w*.* $\mathcal{C}$ *is said to be* non-trivial *if it contains two values* $\boldsymbol{\kappa_0}$ *and* $\boldsymbol{\kappa_1}$*, which are computationally different, i.e., such that* $\neg(\boldsymbol{\kappa_0} \sim_w \boldsymbol{\kappa_1})$ *for all* w*.*

A choice name δ can be used in a computation to access choices from a world as follows: $!\delta \mapsto_w$ choice?(w,δ) (as shown in Fig. 1). This allows getting the last δ-choice from the current world w. The quotienting type operator ↯ allows assigning types to such expressions that compute to different values in different worlds. For example, as defined in Fig. 2, while Nat is the type of expressions that when they compute to $\underline{i}$ in w_1 must also compute to $\underline{i}$ in $w_2 \sqsupseteq w_1$, ↯Nat is the type of expressions that can compute to $\underline{i}$ in w_1 and to another number $\underline{j}$ in $w_2 \sqsupseteq w_1$. This is used to assign types to computations involving choices. For example, $!\delta$ inhabits ↯Nat when its choices are numbers.

Note that the above definition of choice? is a slight simplification of the more general notion of choices presented in [6]. There, the choice? function was of type $\mathcal{W} \rightarrow \mathcal{N} \rightarrow \mathbb{N} \rightarrow \mathcal{C}$. The additional $\mathbb{N}$ component enables a more general notion of choice operators, including ones in which the history is recorded. In references, which is the notion of choices we especially focus on in this paper, one only maintains the latest update and so the $\mathbb{N}$ component becomes moot. Thus, for simplicity of presentation, we elide the $\mathbb{N}$ component in this paper, but full details are available in the Agda implementation.

$\text{TT}_{\mathcal{C}}^{\square}$ also includes the notion of a *restriction*, which allows assuming that the choices made for a given choice name all satisfy a pre-defined constraint. Here again we simplify the concept for choices without history tracking.

[1] To guarantee that $\mathcal{C} \subseteq$ Term, one can for example extend the syntax to include a designated constructor for choices, or require a coercion $\mathcal{C} \rightarrow$ Term. We opted for the latter in our formalization.

▶ **Definition 3** (Restrictions). *A restriction $r \in \mathsf{Res}$ is a pair $\langle res, d\rangle$ consisting of a function $res \in \mathcal{C} \rightarrow \mathbb{P}$ and a default choice $d \in \mathcal{C}$ such that $(res\ d)$ holds. Given such a pair r, we write $r_{\cdot \mathsf{d}}$ for d.*

Intuitively, *res* specifies a restriction on the choices that can be made at any point in time and d provides a default choice that meets this restriction (e.g., for reference cells, this default choice is used to initialize a cell). For example, the restriction $\langle \lambda\kappa.\kappa \in \mathbb{N}, 0\rangle$ requires choices to be numbers and provides 0 as a default value. To reason about restrictions, we require the existence of a "compatibility" predicate as follows.

▶ **Definition 4** (Compatibility). *We say that C is compatible if there exists a predicate $\mathsf{comp} \in \mathcal{N} \rightarrow \mathcal{W} \rightarrow \mathsf{Res} \rightarrow \mathbb{P}$, intended to guarantee that restrictions are satisfied, and which is preserved by $\sqsubseteq$: $\forall(\delta : \mathcal{N})(w_1, w_2 : \mathcal{W})(r : \mathsf{Res}).w_1 \sqsubseteq w_2 \rightarrow \mathsf{comp}(\delta, w_1, r) \rightarrow \mathsf{comp}(\delta, w_2, r)$.*

$\mathrm{TT}_{\mathcal{C}}^{\square}$ further requires the ability to create new choice names as follows.

▶ **Definition 5** (Extendability). *We say that C is extendable if there exists a function $\nu\mathcal{C} \in \mathcal{W} \rightarrow \mathcal{N}$, where $\nu\mathcal{C}(w)$ is intended to return a new choice name not present in w, and a function $\mathsf{start}\nu\mathcal{C} \in \mathcal{W} \rightarrow \mathsf{Res} \rightarrow \mathcal{W}$, where $\mathsf{start}\nu\mathcal{C}(w, r)$ is intended to return an extension of w with the new choice name $\nu\mathcal{C}(w)$ with restriction r, satisfying the following properties:*
- *Starting a new choice extends the current world:* $\forall(w : \mathcal{W})(r : \mathsf{Res}).w \sqsubseteq \mathsf{start}\nu\mathcal{C}(w, r)$
- *Initially, the only possible choice is the default value of the given restriction, i.e.:*
  $\forall(r : \mathsf{Res})(w : \mathcal{W})(\kappa : \mathcal{C}).\mathsf{choice?}(\mathsf{start}\nu\mathcal{C}(w, r), \nu\mathcal{C}(w)) = \kappa \rightarrow \kappa = r_{\cdot \mathsf{d}}$
- *A choice is initially compatible with its restriction:*
  $\forall(w : \mathcal{W})(r : \mathsf{Res}).\mathsf{comp}(\nu\mathcal{C}(w), \mathsf{start}\nu\mathcal{C}(w, r), r)$

Lastly, $\mathrm{TT}_{\mathcal{C}}^{\square}$ requires the ability to update a choice as follows.

▶ **Definition 6** (Mutability). *We say that C is mutable if there exists a function $\mathsf{update} \in \mathcal{W} \rightarrow \mathcal{N} \rightarrow \mathcal{C} \rightarrow \mathcal{W}$ such that if $w \in \mathcal{W}$, $\delta \in \mathcal{N}$, $\kappa \in \mathcal{C}$, then $w \sqsubseteq \mathsf{update}(w, \delta, \kappa)$.*

From this point on, we will only discuss choices C that are compatible, extendable and mutable. The abstract notion of choice operators has many concrete instances. This paper focuses on one concrete instance – mutable references.

▶ **Example 7** (References). Reference cells, which are values that allow a program to indirectly access a particular object, are choice operators since they can point to different objects over their lifetime. Formally, we define references to numbers, Ref, as follows (see worldInstanceRef.lagda for details):

Non-trivial Choices Let $\mathcal{N} := \mathbb{N}$ and $\mathcal{C} := \mathbb{N}$, which is non-trivial, e.g., take $\boldsymbol{\kappa_0} := \underline{0}$ and $\boldsymbol{\kappa_1} := \underline{1}$.

Worlds Worlds are lists of cells, where a cell is a quadruple of (1) a choice name, (2) a restriction, (3) a choice, and (4) a Boolean indicating whether the cell is mutable. $\sqsubseteq$ is the reflexive transitive closure of two operations that allow (i) creating a new reference cell, and (ii) updating an existing reference cell. We define $\mathsf{choice?}(w, \delta)$ so that it simply accesses the content of the δ cell in w.

Compatible $\mathsf{comp}(\delta, w, r)$ states that a reference cell named δ with restriction r was created in the world w (using operation of type (i) described above), and that the current value of the cell satisfies r.

Extendable $\nu\mathcal{C}(w)$ returns a reference name not occurring in w; and $\mathsf{start}\nu\mathcal{C}(w, r)$ adds a new reference cell to w with name $\nu\mathcal{C}(w)$ and restriction r (using operation of type (i) mentioned above).

Mutable $\mathsf{update}(w, \delta, \kappa)$ updates the reference δ with the choice κ if δ occurs in w, and otherwise returns w (using operation of type (ii) mentioned above).

3 Instantiating $\mathrm{TT}_{\mathcal{C}}^{\Box}$

To validate continuity, we need to internalize some semantical properties of $\mathrm{TT}_{\mathcal{C}}^{\Box}$ that were introduced in [6] and recalled in Sec. 2. Concretely, we instantiate (and extend) $\mathrm{TT}_{\mathcal{C}}^{\Box}$ with the following components, which are formally defined next.

- An operator that allows us to make a choice. This has far-reaching consequences, as a computation can now modify its current world. We generalize $\mathrm{TT}_{\mathcal{C}}^{\Box}$'s semantics accordingly. This is an internalization of the mutability requirement.
- An operator to generate a "fresh" choice name. This is an internalization of the extendability requirement.
- A type that states the "purity" of an expression, i.e., that the expression has no side effects. This will allow us to formalize the variant of the continuity principle we validate. Sec. 4.1 provides further details.

3.1 Syntax & Operational Semantics

We extend $\mathrm{TT}_{\mathcal{C}}^{\Box}$'s syntax as follows:

$$\begin{aligned}
t \in \mathsf{Term} &::= \cdots \boxed{\mid \texttt{choose}(t_1, t_2)} \boxed{\mid \boldsymbol{\nu} x.t} \mid \texttt{let}\ x = t_1\ \texttt{in}\ t_2 \\
&\quad\ \mid \texttt{if}\ t_1\ \texttt{<}\ t_2\ \texttt{then}\ t_3\ \texttt{else}\ t_4 \mid t_1 + t_2 \\
vt \in \mathsf{Type} &::= \cdots \boxed{\mid \texttt{pure}} \mid t_1 \cap t_2 \mid \downharpoonleft t
\end{aligned}$$

As in Sec. 2.3, the blue boxes highlight the time-related component. The term `pure` is the type of "pure" terms, i.e. terms that do no contain choice names. The term $t_1 \cap t_2$ is an intersection type, which is inhabited by the inhabitants of both t_1 and t_2. Finally, $\downharpoonleft t$ turns a type t into a subsingleton type that equates all elements of t. The expression $\texttt{choose}(\delta, t)$ makes the δ-choice t; while $\boldsymbol{\nu} x.t$ creates a "fresh" choice name w.r.t. t, thereby internalizing the notion of extendability presented in Def. 5. The term $\texttt{let}\ x = t_1\ \texttt{in}\ t_2$ is a call-by-value operator that allows evaluating t_1 to a value before proceeding with t_2. We write $t_1;t_2$ for $\texttt{let}\ x = t_1\ \texttt{in}\ t_2$ where x does not occur free in t_2.

We extend $\mathrm{TT}_{\mathcal{C}}^{\Box}$'s operational semantics as follows. We turn the ternary relation $a \Downarrow_w b$ into a four-place relations $a \Downarrow_{w_2}^{w_1} b$ which captures that a computes to b starting from the world w_1 and updating it so that the resulting world is w_2 at the end of the computation. Most computations do not modify the current world except $\texttt{choose}(t_1, t_2)$.

$$\begin{array}{lll}
(\lambda x.t)\ u \mapsto_w^w t[x \backslash u] & \quad & \texttt{let}\ x, y = \langle t_1, t_2 \rangle\ \texttt{in}\ t \mapsto_w^w t[x \backslash t_1; y \backslash t_2] \\
\texttt{fix}(v) \mapsto_w^w v\ \texttt{fix}(v) & & \texttt{case}\ \texttt{inl}(t)\ \texttt{of}\ \texttt{inl}(x) \Rightarrow t_1 \mid \texttt{inr}(y) \Rightarrow t_2 \mapsto_w^w t_1[x \backslash t] \\
!\delta \mapsto_w^w \mathsf{choice?}(w, \delta) & & \texttt{case}\ \texttt{inr}(t)\ \texttt{of}\ \texttt{inl}(x) \Rightarrow t_1 \mid \texttt{inr}(y) \Rightarrow t_2 \mapsto_w^w t_2[y \backslash t]
\end{array}$$

In addition we now have the following computations:

$$\begin{array}{ll}
\texttt{if}\ \underline{n}\ \texttt{<}\ \underline{m}\ \texttt{then}\ t_1\ \texttt{else}\ t_2 & \mapsto_w^w t_1, \text{if } n < m \\
\texttt{if}\ \underline{n}\ \texttt{<}\ \underline{m}\ \texttt{then}\ t_1\ \texttt{else}\ t_2 & \mapsto_w^w t_2, \text{if } m \leq n \\
\underline{n} + \underline{m} & \mapsto_w^w \underline{n + m} \\
\texttt{let}\ x = v\ \texttt{in}\ t_2 & \mapsto_w^w t_2[x \backslash v]
\end{array}$$

The semantics of $\texttt{choose}(t_1, t_2)$ is defined as follows:

$$\texttt{choose}(\delta, t) \mapsto_{\mathsf{update}(w, \delta, t)}^{w} \star$$

Choosing a δ-choice t using $\texttt{choose}(\delta, t)$ results in a corresponding update of the current world, namely $\mathsf{update}(w, \delta, t)$. The computation returns $\star$, which is reminiscent of reference updates in OCaml for example, which are of type $\texttt{unit}$. As mentioned in Def. 2, we require $\mathcal{C} \subseteq \mathsf{Term}$ so that choices can be included in computations. In addition, because $\mathsf{update} \in \mathcal{W} \to \mathcal{N} \to \mathcal{C} \to \mathcal{W}$, for $\mathsf{update}(w, \delta, t)$ to be well-defined for $t \in \mathsf{Term}$, we require a coercion from Term to $\mathcal{C}$ so that t can be turned into a choice, and update can be applied to that choice. This coercion is left implicit for readability. We further require that applying this coercion to a choice κ returns κ, which is used to validate the assumption Ass_3 discussed in Sec. 4.2.

▶ **Example 8.** We saw in Ex. 7, that $\mathcal{C}$ can for example be instantiated to be $\mathbb{N}$. A coercion from Term to $\mathcal{C}$ can then turn $\underline{n}$ into n and all other terms to 0, which satisfies the requirement that choices are mapped to the same choices.

Finally, we describe how $\boldsymbol{\nu}x.t$ computes. Intuitively, it selects a "fresh" choice name δ and instantiate the variable x with δ. Formally, it computes as follows:

$$\boldsymbol{\nu}x.t \mapsto^{w}_{\mathsf{start}\nu\mathcal{C}(w,\mathtt{r})} t[x\backslash\nu\mathcal{C}(w)]$$

where $\mathtt{r}$ is the restriction $\langle\lambda c.(c \in \mathbb{N}), 0\rangle$, which constrains the choices to be numbers, with default value 0. Other restrictions could be supported, for example by adding different $\boldsymbol{\nu}$ symbols to the language and by selecting during computation the appropriate restriction based on the $\boldsymbol{\nu}$ operator at hand. This is however left for future work as we especially focus here on the choices presented in Ex. 8.

▶ Remark 9 (Freshness). The fresh operator used in [24] computes $\boldsymbol{\nu}x.a$ by reducing a to b, and then returning $\boldsymbol{\nu}x.b$, thereby never generating new fresh names. As opposed to that fresh operator, which was based on nominal sets, the one introduced in this paper cannot put back the "fresh" constructor at each step of the small step derivation, otherwise a multi-step computation would not be able to use a choice name to keep track of the modulus of continuity of a function across multiple computation steps by recording it in the current world. One consequence of this is that this fresh operator cannot guarantee that it generates a truly "fresh" name that does not occur anywhere else (therefore, it does not satisfy the nominal axioms). For example $(\boldsymbol{\nu}x.x)\ \delta$ might generate the name δ because it does not occur in the local expression $\boldsymbol{\nu}x.x$.

Formally, $a \Downarrow^{w_1}_{w_2} b$ is the reflexive and transitive closure of $\mapsto$, i.e., it holds if a in world w_1 computes to b in world w_2 in 0 or more steps. Thanks to the properties of $\mathsf{start}\nu\mathcal{C}$ presented in Def. 5, and the properties of update presented in Def. 6, computations respect $\sqsubseteq$:

▶ **Lemma 10** (Computations respect $\sqsubseteq$)**.** *If $a \Downarrow^{w_1}_{w_2} b$ then $w_1 \sqsubseteq w_2$.*

3.2 Forcing Interpretation

$\mathrm{TT}^{\square}_{\mathcal{C}}$'s semantics is similar to the one presented in [6], which we recall and extend in Fig. 2. It is interpreted via a forcing interpretation in which the forcing conditions are worlds. This interpretation is defined using induction-recursion as follows: (1) the inductive relation $w \vDash T_1{\equiv}T_2$ expresses type equality in the world w; (2) the recursive function $w \vDash t_1{\equiv}t_2{\in}T$ expresses equality in a type. We further use the following abstractions: $w \vDash \texttt{type}(T)$ for $w \vDash T{\equiv}T$, $w \vDash t{\in}T$ for $w \vDash t{\equiv}t{\in}T$, and $w \vDash T$ for $\exists(t : \mathsf{Term}).w \vDash t{\in}T$. Note that a major difference is that while $a \Downarrow_w b$ is still defined as $\forall^{\sqsubseteq}_{w}(w'.a \Downarrow_{w'} b)$ as in [6], $a \Downarrow_{w'} b$ is now defined as $\exists(w'' : \mathcal{W}).a \Downarrow^{w'}_{w''} b$ to account for the fact that computations can now update the current

Numbers:

- $w \vDash \mathsf{Nat}\equiv\mathsf{Nat} \iff \texttt{True}$
- $w \vDash t\equiv t'\in\mathsf{Nat} \iff \Box_w(w'.\exists(n:\mathbb{N}).t \Downarrow_{w'} \underline{n} \wedge t' \Downarrow_{w'} \underline{n})$

Products:

- $w \vDash \mathbf{\Pi}x{:}A_1.B_1\equiv\mathbf{\Pi}x{:}A_2.B_2 \iff \mathsf{Fam}_w(A_1,A_2,B_1,B_2)$
- $w \vDash f\equiv g\in\mathbf{\Pi}x{:}A.B \iff \Box_w(w'.\forall(a_1,a_2:\mathsf{Term}).w' \vDash a_1\equiv a_2\in A \rightarrow w' \vDash f\ a_1\equiv g\ a_2\in B[x\backslash a_1])$

Sums:

- $w \vDash \mathbf{\Sigma}x{:}A_1.B_1\equiv\mathbf{\Sigma}x{:}A_2.B_2 \iff \mathsf{Fam}_w(A_1,A_2,B_1,B_2)$
- $w \vDash p_1\equiv p_2\in\mathbf{\Sigma}x{:}A.B \iff \Box_w(w'.\exists(a_1,a_2,b_1,b_2:\mathsf{Term}).w' \vDash a_1\equiv a_2\in A \wedge w' \vDash b_1\equiv b_2\in B[x\backslash a_1] \wedge p_1 \Downarrow_{w'} \langle a_1,b_1\rangle \wedge p_2 \Downarrow_{w'} \langle a_2,b_2\rangle)$

Sets:

- $w \vDash \{x:A_1 \mid B_1\}\equiv\{x:A_2 \mid B_2\} \iff \mathsf{Fam}_w(A_1,A_2,B_1,B_2)$
- $w \vDash a_1\equiv a_2\in\{x:A \mid B\} \iff \Box_w(w'.\exists(b_1,b_2:\mathsf{Term}).w' \vDash a_1\equiv a_2\in A \wedge w' \vDash b_1\equiv b_2\in B[x\backslash a_1])$

Disjoint unions:

- $w \vDash A_1+B_1\equiv A_2+B_2 \iff w \vDash A_1\equiv A_2 \wedge w \vDash B_1\equiv B_2$
- $w \vDash a_1\equiv a_2\in A+B \iff \Box_w(w'.\exists(u,v:\mathsf{Term}).(a_1 \Downarrow_{w'} \texttt{inl}(u) \wedge a_2 \Downarrow_{w'} \texttt{inl}(v) \wedge w' \vDash u\equiv v\in A) \vee (a_1 \Downarrow_{w'} \texttt{inr}(u) \wedge a_2 \Downarrow_{w'} \texttt{inr}(v) \wedge w' \vDash u\equiv v\in B))$

Equalities:

- $w \vDash (a_1{=}b_1\in A)\equiv(a_2{=}b_2\in B) \iff w \vDash A\equiv B \wedge \forall_w^{\sqsubseteq}(w'.w' \vDash a_1\equiv a_2\in A) \wedge \forall_w^{\sqsubseteq}(w'.w' \vDash b_1\equiv b_2\in B)$
- $w \vDash a_1\equiv a_2\in(a{=}b\in A) \iff \Box_w(w'.w' \vDash a\equiv b\in A)$
  (note that a_1 and a_2 can be any term here)

Time-Quotiented types:

- $w \vDash \wr A\equiv\wr B \iff w \vDash A\equiv B$
- $w \vDash a\equiv b\in\wr A \iff \Box_w(w'.(\lambda a,b.\exists(c,d:\mathsf{Value}).a \sim_w c \wedge b \sim_w d \wedge w \vDash c\equiv d\in A)^+\ a\ b)$

Subsingletons:

- $w \vDash \downharpoonleft A\equiv\downharpoonleft B \iff w \vDash A\equiv B$
- $w \vDash a\equiv b\in\downharpoonleft A \iff \Box_w(w'.w' \vDash a\equiv a\in A \wedge w' \vDash b\equiv b\in A)$

Purity:

- $w \vDash \texttt{pure}\equiv\texttt{pure} \iff \texttt{True}$
- $w \vDash a_1\equiv a_2\in\texttt{pure} \iff \texttt{namefree}(a_1) \wedge \texttt{namefree}(a_2)$

Binary intersections:

- $w \vDash A_1 \cap B_1\equiv A_2 \cap B_2 \iff w \vDash A_1\equiv A_2 \wedge w \vDash B_1\equiv B_2$
- $w \vDash a_1\equiv a_2\in A \cap B \iff \Box_w(w'.w' \vDash a_1\equiv a_2\in A \wedge w' \vDash a_1\equiv a_2\in B)$

Modality closure:

- $w \vDash T_1\equiv T_2 \iff \Box_w(w'.\exists(T_1',T_2':\mathsf{Term}).T_1 \Downarrow_{w'} T_1' \wedge T_2 \Downarrow_{w'} T_2' \wedge w' \vDash T_1'\equiv T_2')$
- $w \vDash t_1\equiv t_2\in T \iff \Box_w(w'.\exists(T':\mathsf{Term}).T \Downarrow_{w'} T' \wedge w' \vDash t_1\equiv t_2\in T')$

Figure 2 Forcing Interpretation.

world. We also define $a \Downarrow_{!w} b$ as $\forall_w^{\sqsubseteq}(w'.a \Downarrow_{w'}^{w'} b)$, capturing the fact that the computation does not change the initial world (this is used in Thm. 12). Fig. 2 defines in particular the semantics of $\texttt{pure}$, which is inhabited by name-free terms, where $\texttt{namefree}(t)$ is defined recursively over t and returns false iff t contains a choice name δ or a fresh operator of the form $\boldsymbol{\nu}x.t$. There, we write R^+ for R's transitive closure, which is used to prove the transitivity of time-quotiented types, in the sense of Thm. 12. We also write $\mathsf{Fam}_w(A_1,A_2,B_1,B_2)$ for $w \vDash A_1\equiv A_2 \wedge \forall_w^{\sqsubseteq}(w'.\forall(a_1,a_2:\mathsf{Term}).w' \vDash a_1\equiv a_2\in A_1 \rightarrow w' \vDash B_1[x\backslash a_1]\equiv B_2[x\backslash a_2])$.

This forcing interpretation is parameterized by a family of abstract modalities $\square$, which we sometimes refer to simply as a modality, which is a function that takes a world w to its modality $\square_w \in \mathcal{P}_w \to \mathbb{P}$. We often write $\square_w(w'.P)$ for $\square_w \lambda w'.P$. As in [6], to guarantee that this interpretation yields a standard type system in the sense of Thm. 12, we require that the modalities satisfy certain properties reminiscent of standard modal axiom schemata [10], which we repeat here for ease of read:

▶ **Definition 11** (Equality modality). *The modality $\square$ is called an* equality modality *if it satisfies the following properties:*

- $\square_1$ *(monotonicity of $\square$):* $\forall(w : \mathcal{W})(P : \mathcal{P}_w).\forall w' \sqsupseteq w.\, \square_w P \to \square_{w'} P$.
- $\square_2$ *(K, distribution axiom):* $\forall(w : \mathcal{W})(P, Q : \mathcal{P}_w).\, \square_w (w'.P\ w' \to Q\ w') \to \square_w P \to \square_w Q$
- $\square_3$ *(C4, i.e., $\square$ follows from $\square\square$):* $\forall(w : \mathcal{W})(P : \mathcal{P}_w).\, \square_w (w'.\, \square_{w'} P) \to \square_w P$
- $\square_4$*:* $\forall(w : \mathcal{W})(P : \mathcal{P}_w).\forall^{\sqsubseteq}_{w}(P) \to \square_w P$
- $\square_5$ *(T, reflexivity axiom):* $\forall(w : \mathcal{W})(P : \mathbb{P}).\, \square_w (w'.P) \to P$

▶ **Theorem 12.** *Given a computation system with choices $\mathcal{C}$ and an equality modality $\square$, $TT^{\square}_{\mathcal{C}}$ is a standard type system in the sense that its forcing interpretation induced by $\square$ satisfy the following properties (where free variables are universally quantified):*

transitivity:	$w \vDash T_1 \equiv T_2 \to w \vDash T_2 \equiv T_3 \to w \vDash T_1 \equiv T_3$	$w \vDash t_1 \equiv t_2 \in T \to w \vDash t_2 \equiv t_3 \in T \to w \vDash t_1 \equiv t_3 \in T$
symmetry:	$w \vDash T_1 \equiv T_2 \to w \vDash T_2 \equiv T_1$	$w \vDash t_1 \equiv t_2 \in T \to w \vDash t_2 \equiv t_1 \in T$
computation:	$w \vDash T \equiv T \to T \Downarrow_{!w} T' \to w \vDash T \equiv T'$	$w \vDash t \equiv t \in T \to t \Downarrow_{!w} t' \to w \vDash t \equiv t' \in T$
monotonicity:	$w \vDash T_1 \equiv T_2 \to w \sqsubseteq w' \to w' \vDash T_1 \equiv T_2$	$w \vDash t_1 \equiv t_2 \in T \to w \sqsubseteq w' \to w' \vDash t_1 \equiv t_2 \in T$
locality:	$\square_w(w'.w' \vDash T_1 \equiv T_2) \to w \vDash T_1 \equiv T_2$	$\square_w(w'.w' \vDash t_1 \equiv t_2 \in T) \to w \vDash t_1 \equiv t_2 \in T$
consistency:	$\neg w \vDash t \in \texttt{False}$	

Proof. The proof relies on the properties of the equality modality. For example: $\square_1$ is used to prove monotonicity when $w \vDash T_1 \equiv T_2$ is derived by closing under $\square_w$; $\square_2$ and $\square_4$ are used, e.g., to prove the symmetry and transitivity of $w \vDash t \equiv t' \in \mathsf{Nat}$; $\square_3$ is used to prove locality; and $\square_5$ is used to prove consistency. See props3.lagda for further details. ◀

As indicated in Thm. 12, and as opposed to the counterpart of the theorem in [6], $w \vDash T \equiv T$ and $w \vDash t \equiv t \in T$ are no longer closed under all computations. For example, when $T := \mathsf{Nat}$, if $t \Downarrow_w t'$ and $t \Downarrow_w \underline{n}$, does not necessarily give us that $t' \Downarrow_w \underline{n}$. An example is $t := (\texttt{choose}(\delta, \underline{1}); \texttt{if } !\delta \texttt{ < } \underline{1} \texttt{ then } \underline{0} \texttt{ else } \underline{1})$, which reduces to $t' := (\texttt{if } !\delta \texttt{ < } \underline{1} \texttt{ then } \underline{0} \texttt{ else } \underline{1})$ and also to $\underline{1}$ in all worlds, but t' does not reduce to $\underline{1}$ in all worlds, because δ could be initialized differently in different worlds. However, the following holds by transitivity of $\Downarrow_w$: $t' \Downarrow_w t \to w \vDash t \equiv t \in \mathsf{Nat} \to w \vDash t \equiv t' \in \mathsf{Nat}$. Similarly, the following also holds by transitivity of $\Downarrow_w$: $w \vDash T \equiv T \to T' \Downarrow_w T \to w \vDash T \equiv T'$. Finally, note that, as indicated in Thm. 12, this semantics is closed under β-reduction, as β-reduction does not modify the current world.

4 Proof of Continuity

We can now state the version of Brouwer's continuity principle that we validate in this paper, along with its realizer. For this we first introduce the following notation: $\mathbf{\Pi_p} a{:}A.B := \mathbf{\Pi} a{:}(A \cap \texttt{pure}).B$, which quantifies over pure elements of type A.

▶ **Theorem 13** (Continuity Principle). *The following continuity principle, referred to as* $\mathsf{CONT_p}$*, is valid w.r.t. the semantics presented in Sec. 3.2:*

$$\mathbf{\Pi_p} F{:}\mathcal{B} \to \mathsf{Nat}.\mathbf{\Pi_p} \alpha{:}\mathcal{B}.\downharpoonleft \mathbf{\Sigma} n{:}\mathsf{Nat}.\mathbf{\Pi_p} \beta{:}\mathcal{B}.(\alpha{=}\beta \in \mathcal{B}_n) \to (F(\alpha){=}F(\beta) \in \mathsf{Nat}) \qquad (1)$$

and is inhabited by

$$\lambda F.\lambda\alpha.\langle \texttt{mod}(F,\alpha), \lambda\beta.\lambda e.\star\rangle \tag{2}$$

where $\texttt{mod}(F,\alpha)$ *is the modulus of continuity of the function* $F \in \mathcal{B} \to \mathsf{Nat}$ *at* $\alpha \in \mathcal{B}$ *and is computed by the following expression:*

$$\begin{aligned}\texttt{mod}(F,\alpha) &\coloneqq \boldsymbol{\nu} x.(\texttt{choose}(x,\underline{0});F(\texttt{upd}(x,\alpha));!x + \underline{1})\\ \texttt{upd}(\delta,\alpha) &\coloneqq \lambda x.(\texttt{let}\ y = x\ \texttt{in}\ ((\texttt{if}\ !\delta\ <\ y\ \texttt{then}\ \texttt{choose}(\delta,y)\ \texttt{else}\ \star);\alpha(y)))\end{aligned}$$

More precisely, the following is true for any world w:

$$w \vDash \lambda F.\lambda\alpha.\langle \texttt{mod}(F,\alpha), \lambda\beta.\lambda e.\star\rangle \in \mathsf{CONT}_{\mathsf{p}}$$

The rest of this section describes the proof of this theorem (see `continuity` in continuity7.lagda for details). First, we intuitively explain how $\texttt{mod}(F,\alpha)$ computes the modulus of continuity of a function F at a point α. This is done using the following steps:

1. selecting, using $\boldsymbol{\nu}$, a fresh choice name δ (the variable x gets replaced with the freshly generated name δ when computing `mod`), with the appropriate restriction (here a restriction that requires choices to be numbers as mentioned in Sec. 3.1);
2. setting δ to 0 using $\texttt{choose}(x,\underline{0})$ (where x is δ when this expression computes);
3. applying F to a modified version of α, namely $\texttt{upd}(\delta,\alpha)$, which computes as α, except that in addition it increases δ's value every time α is applied to a number larger than the last chosen one;
4. returning the last chosen number using $!x$ (again x is δ when this expression computes), increased by one in order to return a number higher than any number F applies α to.

We divide the proof of the validity of the continuity principle, i.e., that it is inhabited by the expression presented in Eq. (2), into the following three components, where $F \in \mathcal{B} \to \mathsf{Nat}$ and $\alpha \in \mathcal{B}$:

- Proving that the modulus is a number, i.e., $\texttt{mod}(F,\alpha) \in \mathsf{Nat}$;
- Proving that $\texttt{mod}(F,\alpha)$ returns the highest number that α is applied to in the computation it performs;
- Given $\beta \in \mathcal{B}$, proving that $F(\alpha)$ and $F(\beta)$ return the same number if α and β agree up to $\texttt{mod}(F,\alpha)$.

4.1 Purity

According to Nat's semantics, to prove that $\texttt{mod}(F,\alpha) \in \mathsf{Nat}$ w.r.t. a world w, we have to prove it computes to the same number in all extensions of w. However, this will not be the case if F or α have side effects. For example, if F is $\lambda f.f(!\delta_0);0$, for some choice name δ_0, then it could happen that f gets applied to 0 in some world w_1 if $!\delta_0$ returns 0, and to 1 in some world $w_2 \sqsupseteq w_1$ if $!\delta_0$ returns 1. As $\texttt{mod}(F,\alpha)$ returns the highest number that F applies its argument to, then $\texttt{mod}(F,\alpha)$ would in this instance return different numbers in different extensions, and would therefore not inhabit Nat.

Therefore, to validate a version of continuity which requires the modulus of continuity to be time-invariant as in Eq. (1), one can require that both F and α are pure (i.e., name-free) terms. Thanks to $\boldsymbol{\Pi_p}$, we get to assume that both F and α are in `pure` and therefore are name-free. Note that it would not be enough to use the following pattern: $\boldsymbol{\Pi} F{:}\mathcal{B} \to \mathsf{Nat}.(F{=}F{\in}\texttt{pure}) \to \ldots$, because then for the continuity principle to even be a type, we would have to prove that F is name-free to prove that $F{=}F{\in}\texttt{pure}$ is a type, only knowing that $F \in \mathcal{B} \to \mathsf{Nat}$, which is not true in general.

Let us now mention a potential solution to avoid such a purity requirement, as well as some difficulties it involves, which we leave investigating to future work. One could try to validate instead the following version of the continuity axiom, where $\mathcal{B}_{\wr n} = \{x : \mathsf{Nat} \mid x <_{\wr} n\} \rightarrow \mathsf{Nat}$, assuming the existence of some type $x <_{\wr} n$ that can relate an $x \in \mathsf{Nat}$ with an $n \in \wr\mathsf{Nat}$:

$$\mathbf{\Pi} F{:}\mathcal{B} \rightarrow \mathsf{Nat}.\mathbf{\Pi}\alpha{:}\mathcal{B}.\downarrow\mathbf{\Sigma} n{:}\wr\mathsf{Nat}.\mathbf{\Pi}\beta{:}\mathcal{B}.(\alpha{=}\beta{\in}\mathcal{B}_{\wr n}) \rightarrow (F(\alpha){=}F(\beta){\in}\mathsf{Nat})$$

A first difficulty with this is the type $x <_{\wr} n$, which to prove that it holds in some world w would require proving that x is equal to all possible values that n can take in extensions of w. Another related difficulty is that it is at present unclear whether this rule can be validated constructively. More precisely, proving its validity would require:

(1) Proving that $\mathtt{mod}(F, \alpha) \in \wr\mathsf{Nat}$, which is now straightforward.

(2) Next, we have to prove that $\mathbf{\Pi}\beta{:}\mathcal{B}.(\alpha{=}\beta{\in}\mathcal{B}_{\wr\mathtt{mod}(F,\alpha)}) \rightarrow (F(\alpha){=}F(\beta){\in}\mathsf{Nat})$, i.e., given $\beta \in \mathcal{B}$ such that $\alpha{=}\beta{\in}\mathcal{B}_{\wr\mathtt{mod}(F,\alpha)}$, we have to prove $F(\alpha){=}F(\beta){\in}\mathsf{Nat}$. The assumption $\alpha{=}\beta{\in}\mathcal{B}_{\wr\mathtt{mod}(F,\alpha)}$ tells us that given $k \in \mathsf{Nat}$ such that $k <_{\wr} \mathtt{mod}(F, \alpha)$, $\alpha(k){=}\beta(k){\in}\mathsf{Nat}$. As mentioned above, for $k <_{\wr} \mathtt{mod}(F, \alpha)$ to be true, it must be that k is less than $\mathtt{mod}(F, \alpha)$ in all extensions of the current world. However, without the purity constraint, $\mathtt{mod}(F, \alpha)$ can compute to different numbers in different extensions.

Going back to our goal $F(\alpha){=}F(\beta){\in}\mathsf{Nat}$, given the semantics of Nat presented in Fig. 2, to prove this it is enough to assume that $F(\mathtt{upd}(\delta, \alpha))$ computes to some number $\underline{m}$ in some world w, and to prove that $F(\beta)$ also computes to $\underline{m}$ in w. We can then inspect the computation $F(\mathtt{upd}(\delta, \alpha)) \Downarrow^{w}_{w_1} \underline{k}$, where δ is the name generated by $\mathtt{mod}(F, \alpha)$, and show that it can be converted into a computation $F(\beta) \Downarrow^{w}_{w_2} \underline{k}$, by replacing $\alpha(\underline{i})$ with $\beta(\underline{i})$, whenever we encounter such an expression. To do this, we need to know that $\alpha(\underline{i})$ and $\beta(\underline{i})$ compute to the same number using $\alpha{=}\beta{\in}\mathcal{B}_{\wr\mathtt{mod}(F,\alpha)}$. However, we only know that $\underline{i}$ is less than $\mathtt{mod}(F, \alpha)$ in w, which is not enough to use this assumption, as $\underline{i}$ might be greater than $\mathtt{mod}(F, \alpha)$ in some other world w'. We can address this issue using classical logic to prove that there exists a $w' \sqsupseteq w$ such that for all $w'' \sqsupseteq w$, the smallest number that α is applied to in the computation of $\mathtt{mod}(F, \alpha)$ w.r.t. w' is less than the number that $\mathtt{mod}(F, \alpha)$ computes to w.r.t. w''. In the argument sketched above we can then use w' instead of w.

4.2 Assumptions

Before we prove that the continuity principle is inhabited, we will summarize here the assumptions we will be making to prove this result, where r is a restriction that requires choices to be numbers (see continuity-conds.lagda for details):

$$(\mathrm{Ass}_1)\quad \forall(w : \mathcal{W})(P : \mathcal{P}_w).\ \Box_w\, P \rightarrow \exists^{\sqsubseteq}_{w}(P)$$

$$(\mathrm{Ass}_2)\quad \forall(\delta : \mathcal{N})(w : \mathcal{W})(n : \mathbb{N}).\ \mathsf{comp}(\delta, w, r) \rightarrow \forall^{\sqsubseteq}_{\mathsf{update}(w,\delta,\underline{n})}(w'.\exists(k : \mathbb{N}).\mathsf{choice?}(w', \delta) = \underline{k})$$

$$(\mathrm{Ass}_3)\quad \forall(\delta : \mathcal{N})(w : \mathcal{W})(k : \mathbb{N}).\mathsf{comp}(\delta, w, r) \rightarrow \mathsf{choice?}(\mathsf{update}(w, \delta, \underline{k}), \delta) = k$$

Ass_1 requires the modality $\Box$ to be non-empty in the sense that for $\Box_w P$ to be true, it has to be true for at least one extension of w. This is true about all topological bar spaces (see →∃𝕎 in mod.lagda), and therefore about the Kripke, Beth, and Open modalities which are derived from such spaces [6, Sec.6.2]. Ass_2 requires that the "last" choice of a r-compatible choice name δ is indeed a number. Ass_3 guarantees that retrieving a choice that was just made will return that choice.

The last two assumptions are true about Ref, the formalization of references to numbers presented in Ex. 7 (see for example contInstanceKripkeRef.lagda for the proof that $\mathrm{TT}^{\square}_{\mathcal{C}}$ instantiated with a Kriple modality and references satisfies these properties). In addition both are true about another kind of stateful computations, namely a variant of the formalization of free choice sequences [15, 31, 27, 26, 17, 32, 21] presented in [6, Ex.5], where new choices are pre-pended as opposed to being appended in [6] (see for example contInstanceKripkeCS.lagda for the proof that $\mathrm{TT}^{\square}_{\mathcal{C}}$ instantiated with a Kriple modality and choice sequences satisfies these properties).

4.3 The Modulus is a Number

In this section we prove that $\mathtt{mod}(F, \alpha) \in \mathsf{Nat}$. More precisely, we prove the following (see `testM-NAT` in continuity1.lagda for details):

▶ **Theorem 14** (The Modulus is a Number). *If* $\mathtt{namefree}(F)$, $\mathtt{namefree}(\alpha)$, $w \vDash F{\in}\mathsf{Nat}^{\mathcal{B}}$, *and* $w \vDash \alpha{\in}\mathcal{B}$, *for some world* w, *then*

$$\square_w(w'.\exists(n:\mathbb{N}).\mathtt{mod}(F,\alpha) \Downarrow_{w'} \underline{n}) \tag{3}$$

To prove the above, we will make use of the fact that $w \vDash \mathtt{upd}(\delta, \alpha){\in}\mathcal{B}$ and therefore also $w \vDash F(\mathtt{upd}(\delta, \alpha)){\in}\mathsf{Nat}$, i.e., by the semantics of Nat presented in Fig. 2, we have for some fresh name δ:

$$\square_w(w'.\exists(n:\mathbb{N}).F(\mathtt{upd}(\delta,\alpha)) \Downarrow_{w'} \underline{n}). \tag{4}$$

But for this we first need to start computing $\mathtt{mod}(F, \alpha)$ to generate a fresh name δ according to the current world. If that current world is some world $w' \sqsupseteq w$ (obtained, for example, using $\square_4$ from Def. 11 on Eq. (3)), then we need to be able to get that $F(\mathtt{upd}(\delta, \alpha))$ computes to a number w.r.t. w', which Eq. (4) might not provide. This is the reason for assumption Ass_1.

Going back to the proof of Eq. (3), we use $\square_4$, and have to prove $\exists(n : \mathbb{N}).\mathtt{mod}(F, \alpha) \Downarrow_{w_1} \underline{n}$ for some $w_1 \sqsupseteq w$. We then:

- (A) first have to find a number n such that $\mathtt{mod}(F, \alpha)$ computes to $\underline{n}$ w.r.t. w_1,
- (B) and then that it does so also for all $w_1' \sqsupseteq w_1$.

Let us prove (A) first. We now start computing $\mathtt{mod}(F, \alpha)$ w.r.t. w_1. We generate a fresh name $\delta := \nu\mathcal{C}(w_1)$, and have to prove that $\mathtt{choose}(\delta, \underline{0});F(\mathtt{upd}(\delta, \alpha));!\delta + \underline{1}$ computes to a number w.r.t. $w_2 := \mathsf{start}\nu\mathcal{C}(w_1, r)$ that satisfies $\mathsf{comp}(\delta, w_2, r)$ (by the properties of $\mathsf{start}\nu\mathcal{C}$ presented in Def. 5). We keep computing this expression and have to prove that $F(\mathtt{upd}(\delta, \alpha));!\delta + \underline{1}$ computes to a number w.r.t. $w_3 := \mathsf{update}(w_2, \delta, \underline{0})$.

From Ass_1 and Eq. (4), we obtain $w_5 \sqsupseteq w$ and $n \in \mathbb{N}$ such that $F(\mathtt{upd}(\delta, \alpha)) \Downarrow_{w_5} \underline{n}$, from which we obtain by definition that there exists a w_6 such that $F(\mathtt{upd}(\delta, \alpha)) \Downarrow^{w_5}_{w_6} \underline{n}$. Now, because F and α are name-free, we can derive that there exists a w_4 such that $F(\mathtt{upd}(\delta, \alpha)) \Downarrow^{w_3}_{w_4} \underline{n}$ (see `differNF⇓APPLY-upd` in terms7.lagda). It now remains to prove that $\underline{n};!\delta + \underline{1}$, computes to a number w.r.t. w_4. It is then enough to prove that $!\delta$ computes to a number $\underline{k}$ w.r.t. w_4, in which case $\underline{n};!\delta + \underline{1}$ computes to $\underline{k{+}1}$ w.r.t. w_4. To prove this we make use of Ass_2 which states that r constrains the δ-choices to be numbers. Using this and the facts that $\mathsf{comp}(\delta, w_2, r)$ and $w_2 \sqsubseteq w_4$ (by $\sqsubseteq$'s transitivity since $w_3 \sqsubseteq w_4$ by Lem. 10 and $w_2 \sqsubseteq w_3$ by Def. 6), we deduce that there exists a $k \in \mathbb{N}$ such that $\mathsf{choice?}(w_4, \delta) = \underline{k}$, and therefore $!\delta$ computes to $\underline{k}$ w.r.t. w_4, and $\underline{n};!\delta + \underline{1}$ computes to $\underline{k{+}1}$ w.r.t. w_4, which concludes the proof of (A).

To prove $\exists(n : \mathbb{N}).\mathtt{mod}(F,\alpha) \Downarrow_{w_1} \underline{n}$, we then instantiate the formula with $k{+}1$, and have to prove $\mathtt{mod}(F,\alpha) \Downarrow_{w_1} \underline{k{+}1}$. We already know that $\mathtt{mod}(F,\alpha) \Downarrow_{w_4}^{w_1} \underline{k{+}1}$ (i.e., part (A)), and we now prove our statement labeled (B) above, i.e., that it does so in all extensions of w_1 too.

To prove (B) we assume a $w_1' \sqsupseteq w_1$ and have to prove that $\mathtt{mod}(F,\alpha)$ computes to $\underline{k{+}1}$ w.r.t. w_1'. As before, we start computing $\mathtt{mod}(F,\alpha)$ w.r.t. w_1', and generate a fresh name $\delta' := \nu\mathcal{C}(w_1')$, and have to prove that $F(\mathtt{upd}(\delta',\alpha));!\delta' + \underline{1}$ computes to $\underline{k{+}1}$ w.r.t. $w_3' := \mathtt{update}(w_2',\delta',\underline{0})$, where $w_2' := \mathtt{start}\nu\mathcal{C}(w_1',r)$. As F and α are name-free, $t_1 := F(\mathtt{upd}(\delta,\alpha))$ and $t_2 := F(\mathtt{upd}(\delta',\alpha))$ behave the same except that when t_1 updates δ with a number, t_2 updates δ' with that number.

Using a syntactic simulation method, we will prove that because t_1 and t_2 are "similar" (which is captured by Def. 15 below), $\mathtt{choice?}(w_3,\delta) = \mathtt{choice?}(w_3',\delta')$, and $t_1 \Downarrow_{w_4}^{w_3} t_1'$, then $t_2 \Downarrow_{w_4'}^{w_3'} t_2'$ such that t_1' and t_2' are also "similar" and $\mathtt{choice?}(w_4,\delta) = \mathtt{choice?}(w_4',\delta')$. Note that $\mathtt{choice?}(w_3,\delta)$ and $\mathtt{choice?}(w_3',\delta')$ return the same choice because $\mathtt{choice?}(w_3,\delta) = \mathtt{choice?}(\mathtt{update}(w_2,\delta,\underline{0}),\delta) = 0$ and $\mathtt{choice?}(w_3',\delta') = \mathtt{choice?}(\mathtt{update}(w_2',\delta',\underline{0}),\delta') = 0$. To derive these equalities, we need assumption Ass_3 that relates $\mathtt{choice?}$ and $\mathtt{update}$.

Let us now define the simulation mentioned above (see `differ` in terms3.lagda for details):

▶ **Definition 15.** *The similarity relation $t_1 \sim_{\delta_1,\delta_2,\alpha} t_2$ is true iff*

$$\begin{aligned}
&(t_1 = \mathtt{upd}(\delta_1,\alpha) \land t_2 = \mathtt{upd}(\delta_2,\alpha))\\
\lor\ &(t_1 = x \land t_2 = x) \lor (t_1 = \star \land t_2 = \star) \lor (t_1 = \underline{n} \land t_2 = \underline{n})\\
\lor\ &(t_1 = \lambda x.a \land t_2 = \lambda x.b \land a \sim_{\delta_1,\delta_2,\alpha} b)\\
\lor\ &(t_1 = (a_1\ b_1) \land t_2 = (a_2\ b_2) \land a_1 \sim_{\delta_1,\delta_2,\alpha} a_2 \land b_1 \sim_{\delta_1,\delta_2,\alpha} b_2)\\
\lor\ &\ldots
\end{aligned}$$

Most cases are omitted in this definition as they similar to the ones presented above. Note however that crucially terms of the form δ or $\boldsymbol{\nu}x.t$ are not related, and that those are the only expressions not related, thereby ruling out names except when occurring inside $\mathtt{upd}$ *through the first clause.*

As discussed above, a key property of this relation is then (see `differ⇓` in terms6.lagda for details):

▶ **Lemma 16.** *If $t_1 \sim_{\delta_1,\delta_2,\alpha} t_2$, $\mathtt{choice?}(w_1,\delta_1) = \mathtt{choice?}(w_2,\delta_2)$, $t_1 \Downarrow_{w_1'}^{w_1} t_1'$, $\mathtt{namefree}(\alpha)$, $\mathtt{comp}(\delta_1,w_1,r)$, and $\mathtt{comp}(\delta_2,w_2,r)$, then there exist w_2' and t_2' such that $t_2 \Downarrow_{w_2'}^{w_2} t_2'$, $t_1' \sim_{\delta_1,\delta_2,\alpha} t_2'$, and $\mathtt{choice?}(w_1',\delta_1) = \mathtt{choice?}(w_2',\delta_2)$.*

which we prove by induction on the computation $t_1 \Downarrow_{w_1'}^{w_1} t_1'$.

We therefore obtain that there exist t_2' and w_4' such that $F(\mathtt{upd}(\delta',\alpha)) \Downarrow_{w_4'}^{w_3'} t_2'$, $\underline{n} \sim_{\delta,\delta',\alpha} t_2'$ and $\mathtt{choice?}(w_4',\delta') = \mathtt{choice?}(w_4,\delta) = \underline{k}$. Furthermore, by definition of the similarity relation, $t_2' = \underline{n}$. We obtain that $F(\mathtt{upd}(\delta',\alpha));!\delta'{+}\underline{1} \Downarrow_{w_4'}^{w_3'} \underline{n};!\delta'{+}\underline{1}$ and so $F(\mathtt{upd}(\delta',\alpha));!\delta'{+}\underline{1} \Downarrow_{w_4'}^{w_3'} !\delta'{+}\underline{1}$. Because $\mathtt{choice?}(w_4',\delta') = \underline{k}$, we finally obtain $F(\mathtt{upd}(\delta',\alpha));!\delta' + \underline{1} \Downarrow_{w_4'}^{w_3'} \underline{k{+}1}$, which concludes the proof of (B), and therefore that $\mathtt{mod}(F,\alpha) \in \mathsf{Nat}$.

4.4 The Modulus is the Highest Number

We now prove that $\mathtt{mod}(F,\alpha)$ returns the highest number that α is applied to in the computation it performs (see `steps-sat-isHighestℕ` in continuity3.lagda for details):

▶ **Theorem 17** (The Modulus is the Highest Number). *If* $\texttt{mod}(F,\alpha) \Downarrow_{w'}^{w} \underline{n}$ *such that* $\texttt{mod}(F,\alpha)$ *generates a fresh name* δ *and* $\mathsf{choice?}(w',\delta) = \underline{i}$*, then for any world* w_0 *occurring along this computation, it must be that* $\mathsf{choice?}(w_0,\delta) = \underline{j}$ *such that* $j \leq i$*.*

As shown above, we know that for any world w_1 there exist $w_2 \in \mathcal{W}$ and $k \in \mathbb{N}$ such that $\texttt{mod}(F,\alpha) \Downarrow_{w_2}^{w_1} \underline{k{+}1}$. As in Sec. 4.3, we start computing $\texttt{mod}(F,\alpha)$ w.r.t. the current world w_1, and generate a fresh name $\delta := \nu\mathcal{C}(w_1)$, and deduce that

$$F(\texttt{upd}(\delta,\alpha));!\delta + \underline{1} \Downarrow_{w_2}^{w_1''} \underline{k{+}1} \tag{5}$$

where $w_1' := \mathsf{start}\nu\mathcal{C}(w_1,r)$ and $w_1'' := \mathsf{update}(w_1',\delta,\underline{0})$. Furthermore, by Ass_2, there must be a $n \in \mathbb{N}$ such that $\mathsf{choice?}(w_2,\delta) = \underline{n}$.

We now want to show that if $n < m$, for some $m \in \mathbb{N}$ (which we will instantiate with $k{+}1$), then it must also be that for any world w along the computation in Eq. (5), if $\mathsf{choice?}(w,\delta) = \underline{i}$ then $i < m$. This is not true about any computation, but it is true about the above because $\texttt{upd}$ only makes a choice if that choice is higher than the "current" one. To capture this, we define the property $\texttt{Upd}_{\delta,\alpha}(t)$, which captures that the only place where δ occurs in t is wrapped inside $\texttt{upd}(\delta,\alpha)$. That is, $\texttt{Upd}_{\delta,\alpha}(t)$ is true iff $t \sim_{\delta,\delta,\alpha} t$. We can then prove the following result by induction on the computation (see continuity3.lagda):

▶ **Lemma 18.** *Let* α *be a closed name-free term, and* t *be a term such that* $\texttt{Upd}_{\delta,\alpha}(t)$ *and* $t \Downarrow_{w_2}^{w_1} u$*, and let* $\mathsf{choice?}(w_2,\delta) = \underline{n}$*, such that* $n < m$*, then for any world* w *along the computation* $t \Downarrow_{w_2}^{w_1} u$ *if* $\mathsf{choice?}(w,\delta) = \underline{i}$ *then* $i < m$*.*

Applying this result to $F(\texttt{upd}(\delta,\alpha));!\delta + \underline{1} \Downarrow_{w_2}^{w_1''} \underline{k{+}1}$ and instantiating m with $k{+}1$, we obtain that for any world w along that computation if $\mathsf{choice?}(w,\delta) = \underline{i}$ then $i < k{+}1$.

4.5 The Modulus is the Modulus

We now prove the crux of continuity, namely that F returns the same number on functions that agree up to $\texttt{mod}(F,\alpha)$ (see $\texttt{eqfg}$ in continuity6.lagda for details):

▶ **Theorem 19** (The Modulus is the Modulus). *If* $w \vDash \alpha{\equiv}\beta{\in}\mathcal{B}_n$ *then* $w \vDash F(\alpha){\equiv}F(\beta){\in}\mathsf{Nat}$*.*

First, we prove that $w \vDash F(\alpha){\equiv}F(\texttt{upd}(\delta,\alpha)){\in}\mathsf{Nat}$, which follows from the semantics of $\mathbf{\Pi}$ and Nat presented in Fig. 2, and in particular the fact that $w \vDash \alpha{\equiv}\texttt{upd}(\delta,\alpha){\in}\mathcal{B}$. It is therefore enough to prove that $F(\texttt{upd}(\delta,\alpha))$ and $F(\beta)$ are equal in Nat. Relating $F(\texttt{upd}(\delta,\alpha))$ and $F(\beta)$ instead of $F(\alpha)$ and $F(\beta)$ allows getting access to the values that α gets applied to in the computation $F(\alpha)$ as they are recorded using the choice name δ. We can then use these values to prove that $F(\texttt{upd}(\delta,\alpha))$ and $F(\beta)$ behave similarly up to applications of α in the first computation, which are applications of β in the second, and that these applications reduce to the same numbers because the arguments, recorded using δ, are less than $\texttt{mod}(F,\alpha)$.

However, even though $\texttt{upd}(\delta,\alpha)$ and α are equal in $\mathcal{B}$, they behave slightly differently computationally as $\texttt{upd}(\delta,\alpha)$ turns the call-by-name computations $\alpha(t)$ into call-by-value computations by first reducing t into an expression of the form $\underline{i}$. By typing, we know that $F(\texttt{upd}(\delta,\alpha))$ and $F(\beta)$ compute to numbers, and to relate the two computations to prove that they compute to the same number, we first apply a similar transformation to $F(\beta)$. Let $\texttt{cbv}$ be defined as follows:

$$\texttt{cbv}(f) := \lambda x.\texttt{let}\ y = x\ \texttt{in}\ f(y).$$

It is straightforward to derive that $w \vDash F(\beta){\equiv}F(\mathtt{cbv}(\beta)){\in}\mathsf{Nat}$ from the semantics of $\boldsymbol{\Pi}$ and Nat presented in Fig. 2. It is therefore enough to prove that $F(\mathtt{upd}(\delta,\alpha))$ and $F(\mathtt{cbv}(\beta))$ are equal in Nat.

Because $F(\mathtt{upd}(\delta,\alpha)) \Downarrow_{w'}^{w} \underline{n}$, by Lem. 18 for any world w_0 along this computation if $\mathsf{choice?}(w_0,\delta) = \underline{i}$ then $i < k{+}1$, where $k{+}1$ is the number computed by $\mathtt{mod}(F,\alpha)$.

We now prove that $F(\mathtt{upd}(\delta,\alpha))$ and $F(\mathtt{cbv}(\beta))$ both compute to $\underline{n}$ through another simulation proof that relies on the following relation (see `updRel` in continuity4.lagda for details):

▶ **Definition 20.** *The similarity relation* $t_1 \approx_{\delta,\alpha,\beta} t_2$ *is true iff*

$$\begin{aligned} & (t_1 = \mathtt{upd}(\delta,\alpha) \wedge t_2 = \mathtt{cbv}(\beta)) \\ \vee\ & (t_1 = x \wedge t_2 = x) \vee (t_1 = \star \wedge t_2 = \star) \vee (t_1 = \underline{n} \wedge t_2 = \underline{n}) \\ \vee\ & (t_1 = \lambda x.a \wedge t_2 = \lambda x.b \wedge a \approx_{\delta,\alpha,\beta} b) \\ \vee\ & (t_1 = (a_1\ b_1) \wedge t_2 = (a_2\ b_2) \wedge a_1 \approx_{\delta,\alpha,\beta} a_2 \wedge b_1 \approx_{\delta,\alpha,\beta} b_2) \\ \vee\ & \ldots \end{aligned}$$

Most cases are omitted in this definition as they similar to the ones presented above. Note however that crucially terms of the form δ or $\boldsymbol{\nu}x.t$ are not related, and that those are the only expressions not related, thereby ruling out names except when occurring inside upd *through the first clause.*

A key property of this relation is as follows, which captures the fact that $t_1 \approx_{\delta,\alpha,\beta} t_2$ is preserved by computations, and which we prove by induction on the computation (see `steps-updRel` in continuity5.lagda for details):

▶ **Lemma 21.** *If $t_1 \approx_{\delta,\alpha,\beta} t_2$, α and β agree up to k, $t_1 \Downarrow_{w'}^{w} t_1'$ and for any world w_0 along this computation if $\mathsf{choice?}(w_0,\delta) = \underline{i}$ then $i < k{+}1$, then $t_2 \Downarrow_{w}^{w} t_2'$ such that $t_1' \approx_{\delta,\alpha,\beta} t_2'$.*

Therefore, because $F(\mathtt{upd}(\delta,\alpha)) \approx_{\delta,\alpha,\beta} F(\mathtt{cbv}(\beta))$ (as F is name-free) and $F(\mathtt{upd}(\delta,\alpha))$ computes to $\underline{n}$, it must be that $F(\mathtt{cbv}(\beta))$ also computes to $\underline{n}$, which concludes our proof of Thm. 13.

5 Conclusion and Related Works

We have shown in this paper how to validate a continuity principle for a subset of the $\mathrm{TT}_{\mathcal{C}}^{\square}$ family of type theories, such that the modulus of continuity of functions is internalized, i.e., computed using an expression of the underlying computation system. In particular, we have used stateful computations, and have discussed some of the challenges arising from such impure computations. As mentioned in the introduction, and as recalled below, this is not the first proof of continuity, however to the best of our knowledge, this is the first proof of an "internal" validity proof of continuity that relies on stateful computations. Furthermore, the proof presented above relies on an "internal" notion of probing through the use of stateful computations internal to the computation language of the type theory, while approaches such as [8, 9, 34] rely on a meta-theoretic (or "external") notion of probing.

Troelstra proved in [29, p.158] that every closed term $t \in \mathbb{N}^{\mathcal{B}}$ of N-HA$^{\omega}$ has a provable modulus of continuity in N-HA$^{\omega}$ – see also [4] for similar proofs of the consistency of continuity with various constructive theories.

Coquand and Jaber [8, 9] proved the *uniform* continuity of a Martin-Löf-like intensional type theory using forcing. Their method consists in adding a generic element `f` as a constant to the language that stands for a Cohen real of type $2^{\mathbb{N}}$, and defining the forcing conditions as approximations of `f`. They then define a suitable *computability* predicate that expresses when a term is a computable term of some type up to approximations given by the forcing conditions. The key steps are to (1) first prove that `f` is computable and then (2) prove that well-typed terms are computable, from which they derive uniform continuity: the uniform modulus of continuity is given by the approximations.

Similarly, Escardó and Xu [34] proved that the definable functionals of Gödel's system T [14] are uniformly continuous on the Cantor space $\mathcal{C}$ (without assuming classical logic or the Fan Theorem). For that, they developed the C-Space category, which internalizes continuity, and has a *Fan functional* which computes the modulus of uniform continuity of functions in $\mathcal{C} \to \mathbb{N}$. Relating C-Space and the standard set-theoretical model of system T, they show that all System T functions on the Cantor space are uniformly continuous. Furthermore, using this model, they show how to extract computational content from proofs in HA^{ω} extended with a uniform continuity axiom, which is realized by the Fan functional.

In [13], Escardó proves that all System T functions are continuous on the Baire space and uniformly continuous on the Cantor space without using forcing. Instead, he provides an alternative interpretation of system T, where a number is interpreted by a *dialogue tree*, which captures the computation of a function w.r.t. an oracle of type $\mathcal{B}$. Escardó first proves that such computations are continuous, and then by defining a suitable relation between the standard interpretation and the alternative one, that relates the interpretations of all system T terms, derives that for all system T functions on the Baire space are continuous.

In [24, 25], the authors proved that Brouwer's continuity principle is consistent with Nuprl [7, 2] by realizing the modulus of continuity of functions on the Baire space also using Longley's method [20], but using exceptions instead of references. The realizer there is more complicated than the one presented in this paper as it involves an effectful computation that repeatedly checks whether a given number is at least as high as the modulus of continuity, and increasing that number until the modulus of continuity is reached. We do not require such a loop, and can directly extract the modulus of continuity of a function.

In [3] the authors prove that all BTT [22] functions are continuous by generalizing the method used in [13]. Their model is built in three steps as follows: an axiom model/translation adds an oracle to the theory at hand; a branching model/translation interprets types as intensional D-algebras, i.e., as types equipped with pythias; and an algebraic parametricity model/translation that relates the two previous translations by relating the calls to the pythia to the oracle. Their models allows deriving that all functions are continuous, but does not allow "internalizing" the continuity principle, which is the goal of this paper.

References

1 Agda wiki. URL: `http://wiki.portal.chalmers.se/agda/pmwiki.php`.

2 Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. *J. Applied Logic*, 4(4):428–469, 2006. `doi:10.1016/j.jal.2005.10.005`.

3 Martin Baillon, Assia Mahboubi, and Pierre-Marie Pédrot. Gardening with the pythia A model of continuity in a dependent setting. In Florin Manea and Alex Simpson, editors, *CSL*, volume 216 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.CSL.2022.5`.

4 Michael J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.

5 Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. London Mathematical Society Lecture Notes Series. Cambridge University Press, 1987. `doi:10.1017/CBO9780511565663`.

6 Liron Cohen and Vincent Rahli. Constructing unprejudiced extensional type theories with choices via modalities. In Amy P. Felty, editor, *FSCD*, volume 228 of *LIPIcs*, pages 10:1–10:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.FSCD.2022.10`.

7 Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, Robert W. Harper, Douglas J. Howe, Todd B. Knoblock, Nax P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing mathematics with the Nuprl proof development system*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

8 Thierry Coquand and Guilhem Jaber. A note on forcing and type theory. *Fundam. Inform.*, 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 *Epistemology versus Ontology*, 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 M. J. Cresswell and G. E. Hughes. *A New Introduction to Modal Logic*. Routledge, 1996.

11 Michael A. E. Dummett. *Elements of Intuitionism*. Clarendon Press, second edition, 2000.

12 Martín H. Escardó and Chuangjie Xu. The inconsistency of a Brouwerian continuity principle with the Curry-Howard interpretation. In *TLCA 2015*, volume 38 of *LIPIcs*, pages 153–164. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. `doi:10.4230/LIPIcs.TLCA.2015.153`.

13 Martín Hötzel Escardó. Continuity of Gödel's system T definable functionals via effectful forcing. *Electr. Notes Theor. Comput. Sci.*, 298:119–141, 2013. `doi:10.1016/j.entcs.2013.09.010`.

14 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, 1989.

15 Stephen C. Kleene and Richard E. Vesley. *The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions*. North-Holland Publishing Company, 1965.

16 Georg Kreisel. On weak completeness of intuitionistic predicate logic. *J. Symb. Log.*, 27(2):139–158, 1962. `doi:10.2307/2964110`.

17 Georg Kreisel and Anne S. Troelstra. Formal systems for some branches of intuitionistic analysis. *Annals of Mathematical Logic*, 1(3):229–387, 1970. `doi:10.1016/0003-4843(70)90001-X`.

18 Saul A. Kripke. Semantical analysis of modal logic i. normal propositional calculi. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 9(5-6):67–96, 1963. `doi:10.1002/malq.19630090502`.

19 Saul A. Kripke. Semantical analysis of intuitionistic logic i. In J.N. Crossley and M.A.E. Dummett, editors, *Formal Systems and Recursive Functions*, volume 40 of *Studies in Logic and the Foundations of Mathematics*, pages 92–130. Elsevier, 1965. `doi:10.1016/S0049-237X(08)71685-9`.

20 John Longley. When is a functional program not a functional program? In *ICFP'99*, pages 1–7. ACM, 1999. `doi:10.1145/317636.317775`.

21 Joan R. Moschovakis. An intuitionistic theory of lawlike, choice and lawless sequences. In *Logic Colloquium'90: ASL Summer Meeting in Helsinki*, pages 191–209. Association for Symbolic Logic, 1993.

22 Pierre-Marie Pédrot and Nicolas Tabareau. An effectful way to eliminate addiction to dependence. In *LICS 2017*, pages 1–12. IEEE Computer Society, 2017. `doi:10.1109/LICS.2017.8005113`.

23 Andrew M Pitts. Nominal sets: Names and symmetry in computer science, volume 57 of cambridge tracts in theoretical computer science, 2013.

24 Vincent Rahli and Mark Bickford. A nominal exploration of intuitionism. In Jeremy Avigad and Adam Chlipala, editors, *CPP 2016*, pages 130–141. ACM, 2016. Extended version: `http://www.nuprl.org/html/Nuprl2Coq/continuity-long.pdf`. `doi:10.1145/2854065.2854077`.

25 Vincent Rahli and Mark Bickford. Validating brouwer's continuity principle for numbers using named exceptions. *Mathematical Structures in Computer Science*, pages 1–49, 2017. `doi:10.1017/S0960129517000172`.

26 Anne S. Troelstra. *Choice sequences: a chapter of intuitionistic mathematics.* Clarendon Press Oxford, 1977.

27 Anne S. Troelstra. Choice sequences and informal rigour. *Synthese*, 62(2):217–227, 1985.

28 Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics An Introduction*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1988.

29 A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis.* New York, Springer, 1973.

30 A.S. Troelstra. A note on non-extensional operations in connection with continuity and recursiveness. *Indagationes Mathematicae*, 39(5):455–462, 1977. doi:10.1016/1385-7258(77)90060-9.

31 Mark van Atten and Dirk van Dalen. Arguments for the continuity principle. *Bulletin of Symbolic Logic*, 8(3):329–347, 2002. URL: http://www.math.ucla.edu/~asl/bsl/0803/0803-001.ps, doi:10.2178/bsl/1182353892.

32 Wim Veldman. Understanding and using Brouwer's continuity principle. In *Reuniting the Antipodes — Constructive and Nonstandard Views of the Continuum*, volume 306 of *Synthese Library*, pages 285–302. Springer Netherlands, 2001. doi:10.1007/978-94-015-9757-9_24.

33 Chuangjie Xu. *A continuous computational interpretation of type theories.* PhD thesis, University of Birmingham, UK, 2015. URL: http://etheses.bham.ac.uk/5967/.

34 Chuangjie Xu and Martín Hötzel Escardó. A constructive model of uniform continuity. In *TLCA 2013*, volume 7941 of *LNCS*, pages 236–249. Springer, 2013. doi:10.1007/978-3-642-38946-7_18.