




Realizing Continuity Using Stateful Computations

Liron Cohen   

Ben-Gurion University, 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 \rightarrow Type theory; Theory of computation \rightarrow 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.2023.35](https://doi.org/10.4230/LIPIcs.CSL.2023.35)

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 [16; 12; 5; 6; 31]. The continuity principle for numbers, sometimes referred to as the weak continuity principle, states that all functions on the Baire space (i.e., $\mathcal{B} := \text{Nat}^{\text{Nat}}$) have a modulus of continuity. More concretely, given a function F of type $\mathcal{B} \rightarrow \text{Nat}$ 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 $\mathcal{B}_n := \{x : \text{Nat} \mid x < n\} \rightarrow \text{Nat}$ is the set of finite sequences of length n :

$$\text{WCP} = \prod F : \mathcal{B} \rightarrow \text{Nat} . \prod \alpha : \mathcal{B} . \downarrow \sum n : \text{Nat} . \prod \beta : \mathcal{B} . (\alpha = \beta \in \mathcal{B}_n) \rightarrow (F(\alpha) = F(\beta) \in \text{Nat})$$

A number of theories have been shown to satisfy Brouwer's continuity principle, or uniform variants, such as N-HA $^\omega$ by Troelstra [28, p.158], MLTT by Coquand and Jaber [10; 9], System T by Escardó [14], CTT by Rahli and Bickford [25], BTT by Baillon, Mahboubi and Pedrot [4], to cite only a few (see Sec. 5 for further details). These proofs often rely on a semantical forcing-based approach [10; 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 [14; 4]).



© Liron Cohen and Vincent Rahli;
licensed under Creative Commons License CC-BY 4.0
31st EACSL Annual Conference on Computer Science Logic (CSL 2023).
Editors: Bartek Klin and Elaine Pimentel; Article No. 35; pp. 35:1–35:18



Leibniz International Proceedings in Informatics
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

31 Not only can functions on the Baire space be proved to be continuous, but using effectful
 32 computations, as for example described in [21], one can *compute* the modulus of continuity
 33 of such a function. However, as shown for example by Kreisel [17, p.154], Troelstra [27,
 34 Thm.IIA], and Escardó and Xu [13; 33], continuity is not an extensional property in the
 35 sense that two equal functions might have different moduli of continuity. Therefore, to realize
 36 continuity, the existence of a modulus of continuity has to be truncated as explained, e.g.,
 37 in [13; 33; 25; 26], which is what the \downarrow operator achieves in WCP. Following the effectful
 38 approach, continuity was shown to be realizable in [25; 26] using exceptions.

39 Instead of using exceptions, a more straightforward way to compute the modulus of
 40 continuity of a function on the Baire space is to use reference cells. This was explained, e.g.,
 41 in [21], where the use of references can be seen as the programming counterparts of the more
 42 logical forcing conditions. The computation using references is more efficient than when
 43 using exceptions as it allows computing the modulus of continuity of a function F at a point
 44 α simply by executing F on α , while recording the highest argument that α is applied to,
 45 while using exceptions requires repeatedly searching for the modulus of continuity.

46 Following this line of work, in this paper we show how to use stateful computations to
 47 realize a continuity principle. This allows deriving constructive type theories that include
 48 continuity axioms where the modulus of continuity is internalized in the sense that it is
 49 computed by an expression of the underlying programming language. Concretely, we do so for
 50 $\text{TT}_{\mathcal{C}}^{\square}$ [7], which is a family of extensional type theories parameterized by a type modality \square ,
 51 and a choice type \mathcal{C} , which are presented in more details in Sec. 2. More precisely, we prove
 52 in this paper that all $\text{TT}_{\mathcal{C}}^{\square}$ functions are continuous for some instances of \square and \mathcal{C} : namely
 53 for “non-empty” equality modalities, and reference-like stateful choice operators. Our proof
 54 is for a variant of the weak continuity principle (see Thm. 13), which we show to be inhabited
 55 by a program that relies on a choice operator to keep track of the modulus of continuity of a
 56 given function, following Longley’s method [21]. This variant is restricted to “pure” functions
 57 F , α , and β without side effects, and Sec. 4.1 discusses issues arising with impure functions.

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

66 **2 Background**

67 This section recalls $\text{TT}_{\mathcal{C}}^{\square}$, a family of type theories parameterized by a choice operator \mathcal{C} ,
 68 and a metatheoretical modality \square , which allows typing the choice operators. See [7] for
 69 further details. The choice operators are time-progressing elements that we will in particular
 70 instantiate with references. Sec. 3 carves out a sub-family for which we can validate
 71 computationally relevant continuity rules as shown in Sec. 4.

72 **2.1 Metatheory**

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

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

$v \in \text{Value} ::= vt$ (type)	$\lambda x.t$ (lambda)	\star (constant)
\underline{n} (number)	$\text{inl}(t)$ (left injection)	δ (choice name)
$\langle t_1, t_2 \rangle$ (pair)	$\text{inr}(t)$ (right injection)	
$vt \in \text{Type} ::= \Pi x:t_1.t_2$ (product)	$\{x : t_1 \mid t_2\}$ (set)	$t_1 + t_2$ (disjoint union)
$\Sigma x:t_1.t_2$ (sum)	$t_1 = t_2 \in t$ (equality)	$\Downarrow t$ (time truncation)
\mathbb{U}_i (universe)	Nat (numbers)	
$t \in \text{Term} ::= x$ (variable)	v (value)	$!t$ (read)
$t_1 t_2$ (application)	$\text{fix}(t)$ (fixpoint)	$\text{let } x, y = t_1 \text{ in } t_2$ (pair destructor)
$\text{case } t \text{ of } \text{inl}(x) \Rightarrow t_1 \mid \text{inr}(y) \Rightarrow t_2$ (injection destructor)		
$(\lambda x.t) u \mapsto_{\square} t[x \setminus u]$	$\text{let } x, y = \langle t_1, t_2 \rangle \text{ in } t \mapsto_{\square} t[x \setminus t_1; y \setminus t_2]$	
$\text{fix}(v) \mapsto_{\square} v \text{ fix}(v)$	$\text{case } \text{inl}(t) \text{ of } \text{inl}(x) \Rightarrow t_1 \mid \text{inr}(y) \Rightarrow t_2 \mapsto_{\square} t_1[x \setminus t]$	
$!\delta \mapsto_w \text{choice?}(w, \delta)$	$\text{case } \text{inr}(t) \text{ of } \text{inl}(x) \Rightarrow t_1 \mid \text{inr}(y) \Rightarrow t_2 \mapsto_{\square} t_2[y \setminus t]$	

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 TT_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, TT_C^\square is parameterized by a Kripke frame [20; 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 TT_C^\square ’s Syntax and Operational Semantics

Fig. 1 recalls TT_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 $t[x \setminus u]$ 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 $\Pi x:t_1.t_2$ and

35:4 Realizing Continuity Using Stateful Computations

105 dependent sums of the form $\Sigma x:t_1.t_2$. For convenience we write $t_1 \rightarrow t_2$ for the non-dependent
 106 Π type; True for $0=0 \in \text{Nat}$; False for $0=1 \in \text{Nat}$; and $\neg T$ for $(T \rightarrow \text{False})$.

107 Fig. 1's lower part presents TT_C^\square 's small-step operational semantics, where $t_1 \mapsto_w t_2$
 108 expresses that t_1 reduces to t_2 in one step of computation *w.r.t. the world w* . We omit the
 109 congruence rules that allow computing within terms such as: if $t_1 \mapsto_w t_2$ then $t_1(u) \mapsto_w t_2(u)$.
 110 We denote by \Downarrow the reflexive transitive closure of \mapsto , i.e., $a \Downarrow_w b$ states that a computes
 111 to b in ≥ 0 steps. We also write $a \Downarrow_w b$ if a computes to b in all extensions of w , i.e., if
 112 $\forall_w^\exists (w'.a \Downarrow_{w'} b)$. We write \sim_w for the symmetric and transitive closure of \Downarrow_w .

113 TT_C^\square includes time-progressing notions that rely on worlds to record choices and provides
 114 operators to access the choices stored in a world, which we now recall. Choices are referred to
 115 through their names. A concrete example of such choices are reference cells in programming
 116 languages, where a variable name pointing to a reference cell is the name of the corresponding
 117 reference cell. To this end, TT_C^\square 's computation system is parameterized by a set \mathcal{N} of *choice*
 118 *names*, that is equipped with a decidable equality, and an operator that given a list of names,
 119 returns a name not in the list. This can be given by, e.g., nominal sets [24]. In what follows
 120 we let δ range over \mathcal{N} , and take \mathcal{N} to be \mathbb{N} for simplicity. TT_C^\square is further parameterized
 121 over abstract operators and properties recalled in Defs. 2 and 4–6, which we show how to
 122 instantiate in Ex. 7. Definitions such as Def. 2 provide axiomatizations of operators, and in
 123 addition informally indicate their intended use. Choices are defined abstractly as follows:

124 **► Definition 2 (Choices).** *Let $\mathcal{C} \subseteq \text{Term}$ be a set of choices,¹ and let κ range over \mathcal{C} . We*
 125 *say that a computation system contains $\langle \mathcal{N}, \mathcal{C} \rangle$ -choices if there exists a partial function*
 126 *$\text{choice?} \in \mathcal{W} \rightarrow \mathcal{N} \rightarrow \mathcal{C}$. Given $w \in \mathcal{W}$ and $\delta \in \mathcal{N}$, the returned choice, if it exists, is meant*
 127 *to be the last choice made for δ according to w . \mathcal{C} is said to be **non-trivial** if it contains two*
 128 *values κ_0 and κ_1 , which are computationally different, i.e., such that $\neg(\kappa_0 \sim_w \kappa_1)$ for all w .*

129 A choice name δ can be used in a computation to access choices from a world as follows:
 130 $! \delta \mapsto_w \text{choice?}(w, \delta)$ (as shown in Fig. 1). This allows getting the last δ -choice from the
 131 current world w . The quotienting type operator \Downarrow allows assigning types to such expressions
 132 that compute to different values in different worlds. For example, as defined in Fig. 2, while
 133 Nat is the type of expressions that when they compute to \underline{i} in w_1 must also compute to \underline{i}
 134 in $w_2 \sqsupseteq w_1$, $\Downarrow \text{Nat}$ is the type of expressions that can compute to \underline{i} in w_1 and to another
 135 number \underline{j} in $w_2 \sqsupseteq w_1$. This is used to assign types to computations involving choices. For
 136 example, $! \delta$ inhabits $\Downarrow \text{Nat}$ when its choices are numbers.

137 Note that the above definition of choice? is a slight simplification of the more general
 138 notion of choices presented in [7]. There, the choice? function was of type $\mathcal{W} \rightarrow \mathcal{N} \rightarrow \mathbb{N} \rightarrow \mathcal{C}$.
 139 The additional \mathbb{N} component enables a more general notion of choice operators, including ones
 140 in which the history is recorded. In references, which is the notion of choices we especially
 141 focus on in this paper, one only maintains the latest update and so the \mathbb{N} component becomes
 142 moot. Thus, for simplicity of presentation, we elide the \mathbb{N} component in this paper, but full
 143 details are available in the Agda implementation.

144 TT_C^\square also includes the notion of a *restriction*, which allows assuming that the choices
 145 made for a given choice name all satisfy a pre-defined constraint. Here again we simplify the
 146 concept for choices without history tracking.

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

147 ► **Definition 3** (Restrictions). A restriction $r \in \mathbf{Res}$ is a pair $\langle res, d \rangle$ consisting of a function
 148 $res \in \mathcal{C} \rightarrow \mathbb{P}$ and a default choice $d \in \mathcal{C}$ such that $(res\ d)$ holds. Given such a pair r , we
 149 write $r_{\mathbf{a}}$ for d .

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

155 ► **Definition 4** (Compatibility). We say that \mathcal{C} is **compatible** if there exists a predicate
 156 $\mathbf{comp} \in \mathcal{N} \rightarrow \mathcal{W} \rightarrow \mathbf{Res} \rightarrow \mathbb{P}$, intended to guarantee that restrictions are satisfied, and which is
 157 preserved by \sqsubseteq : $\forall(\delta : \mathcal{N})(w_1, w_2 : \mathcal{W})(r : \mathbf{Res}).w_1 \sqsubseteq w_2 \rightarrow \mathbf{comp}(\delta, w_1, r) \rightarrow \mathbf{comp}(\delta, w_2, r)$.

158 $\mathbf{TT}_{\mathcal{C}}^{\square}$ further requires the ability to create new choice names as follows.

159 ► **Definition 5** (Extendability). We say that \mathcal{C} is **extendable** if there exists a function
 160 $\nu\mathcal{C} \in \mathcal{W} \rightarrow \mathcal{N}$, where $\nu\mathcal{C}(w)$ is intended to return a new choice name not present in w , and
 161 a function $\mathbf{start}\nu\mathcal{C} \in \mathcal{W} \rightarrow \mathbf{Res} \rightarrow \mathcal{W}$, where $\mathbf{start}\nu\mathcal{C}(w, r)$ is intended to return an extension
 162 of w with the new choice name $\nu\mathcal{C}(w)$ with restriction r , satisfying the following properties:

163 ■ Starting a new choice extends the current world: $\forall(w : \mathcal{W})(r : \mathbf{Res}).w \sqsubseteq \mathbf{start}\nu\mathcal{C}(w, r)$

164 ■ Initially, the only possible choice is the default value of the given restriction, i.e.:

165 $\forall(r : \mathbf{Res})(w : \mathcal{W})(\kappa : \mathcal{C}).\mathbf{choice}?(w, r, \nu\mathcal{C}(w)) = \kappa \rightarrow \kappa = r_{\mathbf{a}}$

166 ■ A choice is initially compatible with its restriction:

167 $\forall(w : \mathcal{W})(r : \mathbf{Res}).\mathbf{comp}(\nu\mathcal{C}(w), \mathbf{start}\nu\mathcal{C}(w, r), r)$

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

169 ► **Definition 6** (Mutability). We say that \mathcal{C} is **mutable** if there exists a function $\mathbf{update} \in$
 170 $\mathcal{W} \rightarrow \mathcal{N} \rightarrow \mathcal{C} \rightarrow \mathcal{W}$ such that if $w \in \mathcal{W}$, $\delta \in \mathcal{N}$, $\kappa \in \mathcal{C}$, then $w \sqsubseteq \mathbf{update}(w, \delta, \kappa)$.

171 From this point on, we will only discuss choices \mathcal{C} that are **compatible**, **extendable** and
 172 **mutable**. The abstract notion of choice operators has many concrete instances. This paper
 173 focuses on one concrete instance — mutable references.

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

178 **Non-trivial Choices** Let $\mathcal{N} := \mathbb{N}$ and $\mathcal{C} := \mathbb{N}$, which is **non-trivial**, e.g., take $\kappa_0 := \underline{0}$ and
 179 $\kappa_1 := \underline{1}$.

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

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

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

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

193 **3** Instantiating \mathbb{TT}_C^\square

194 To validate continuity, we need to internalize some semantical properties of \mathbb{TT}_C^\square that were
 195 introduced in [7] and recalled in Sec. 2. Concretely, we instantiate (and extend) \mathbb{TT}_C^\square with
 196 the following components, which are formally defined next.

- 197 ■ An operator that allows us to make a choice. This has far-reaching consequences, as a
 198 computation can now modify its current world. We generalize \mathbb{TT}_C^\square 's semantics accordingly.
 199 This is an internalization of the **mutability** requirement.
- 200 ■ An operator to generate a “fresh” choice name. This is an internalization of the **extend-**
 201 **ability** requirement.
- 202 ■ A type that states the “purity” of an expression, i.e., that the expression has no side
 203 effects. This will allow us to formalize the variant of the continuity principle we validate.
 204 Sec. 4.1 provides further details.

205 **3.1 Syntax & Operational Semantics**

We extend \mathbb{TT}_C^\square 's syntax as follows:

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

206 As in Sec. 2.3, the blue boxes highlight the time-related component. The term **pure** is
 207 the type of “pure” terms, i.e. terms that do not contain choice names. The term $t_1 \cap t_2$ is an
 208 intersection type, which is inhabited by the inhabitants of both t_1 and t_2 . Finally, $\Downarrow t$ turns a
 209 type t into a subsingleton type that equates all elements of t . The expression **choose**(δ, t)
 210 makes the δ -choice t ; while $\nu x.t$ creates a “fresh” choice name w.r.t. t , thereby internalizing
 211 the notion of extendability presented in Def. 5. The term **let** $x = t_1$ **in** t_2 is a call-by-value
 212 operator that allows evaluating t_1 to a value before proceeding with t_2 . We write $t_1; t_2$ for
 213 **let** $x = t_1$ **in** t_2 where x does not occur free in t_2 .

We extend \mathbb{TT}_C^\square '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 **choose**(t_1, t_2).

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

In addition we now have the following computations:

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

The semantics of **choose**(t_1, t_2) is defined as follows:

$$\text{choose}(\delta, t) \mapsto_w^w \text{update}_{(w, \delta, t)} \star$$

214 Choosing a δ -choice t using `choose`(δ, t) results in a corresponding update of the current
 215 world, namely `update`(w, δ, t). The computation returns \star , which is reminiscent of reference
 216 updates in OCaml for example, which are of type `unit`. As mentioned in Def. 2, we
 217 require $\mathcal{C} \subseteq \mathbf{Term}$ so that choices can be included in computations. In addition, because
 218 `update` $\in \mathcal{W} \rightarrow \mathcal{N} \rightarrow \mathcal{C} \rightarrow \mathcal{W}$, for `update`(w, δ, t) to be well-defined for $t \in \mathbf{Term}$, we require a
 219 coercion from \mathbf{Term} to \mathcal{C} so that t can be turned into a choice, and `update` can be applied to
 220 that choice. This coercion is left implicit for readability. We further require that applying
 221 this coercion to a choice κ returns κ , which is used to validate the assumption `Ass3` discussed
 222 in Sec. 4.2.

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

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

$$\nu x.t \mapsto_{\text{start}\nu\mathcal{C}(w,x)}^w t[x \setminus \nu\mathcal{C}(w)]$$

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

231 ► **Remark 9 (Freshness).** The fresh operator used in [25] computes $\nu x.a$ by reducing a to
 232 b , and then returning $\nu x.b$, thereby never generating new fresh names. As opposed to that
 233 fresh operator, which was based on nominal sets, the one introduced in this paper cannot
 234 put back the “fresh” constructor at each step of the small step derivation, otherwise a
 235 multi-step computation would not be able to use a choice name to keep track of the modulus
 236 of continuity of a function across multiple computation steps by recording it in the current
 237 world. One consequence of this is that this fresh operator cannot guarantee that it generates
 238 a truly “fresh” name that does not occur anywhere else (therefore, it does not satisfy the
 239 nominal axioms). For example $(\nu x.x) \delta$ might generate the name δ because it does not occur
 240 in the local expression $\nu x.x$.

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

244 ► **Lemma 10 (Computations respect \sqsubseteq).** *If $a \Downarrow_{w_2}^{w_1} b$ then $w_1 \sqsubseteq w_2$.*

245 3.2 Forcing Interpretation

246 $\mathbf{TT}_{\mathcal{C}}^{\square}$'s semantics is similar to the one presented in [7], which we recall and extend in Fig. 2.
 247 It is interpreted via a forcing interpretation in which the forcing conditions are worlds. This
 248 interpretation is defined using induction-recursion as follows: (1) the inductive relation
 249 $w \models T_1 \equiv T_2$ expresses type equality in the world w ; (2) the recursive function $w \models t_1 \equiv t_2 \in T$
 250 expresses equality in a type. We further use the following abstractions: $w \models \text{type}(T)$ for
 251 $w \models T \equiv T$, $w \models t \in T$ for $w \models t \equiv t \in T$, and $w \models T$ for $\exists(t : \mathbf{Term}). w \models t \in T$. Note that a major
 252 difference is that while $a \Downarrow_w b$ is still defined as $\forall_w^{\sqsubseteq}(w'. a \Downarrow_{w'} b)$ as in [7], $a \Downarrow_{w'} b$ is now defined
 253 as $\exists(w'' : \mathcal{W}). a \Downarrow_{w''}^{w'} b$ to account for the fact that computations can now update the current

Figure 2 Forcing Interpretation

Numbers:

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

Products:

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

Sums:

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

Sets:

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

Disjoint unions:

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

Equalities:

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

Time-Quotiented types:

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

Subsingletons:

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

Purity:

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

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:

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

254 world. We also define $a \Downarrow_{!w} b$ as $\forall_w^E(w'. a \Downarrow_{w'}^w b)$, capturing the fact that the computation
 255 does not change the initial world (this is used in Thm. 12). Fig. 2 defines in particular the
 256 semantics of `pure`, which is inhabited by name-free terms, where `namefree`(t) is defined
 257 recursively over t and returns false iff t contains a choice name δ or a fresh operator of the form
 258 $\nu x.t$. There, we write R^+ for R 's transitive closure, which is used to prove the transitivity
 259 of time-quotiented types, in the sense of Thm. 12. We also write $\text{Fam}_w(A_1, A_2, B_1, B_2)$
 260 for $w \Vdash A_1 \equiv A_2 \wedge \forall_w^E(w'. \forall (a_1, a_2 : \text{Term}). w' \Vdash a_1 \equiv a_2 \in A_1 \rightarrow w' \Vdash B_1[x \setminus a_1] \equiv B_2[x \setminus a_2])$.
 261 This forcing interpretation is parameterized by a family of abstract modalities \Box , which we
 262 sometimes refer to simply as a modality, which is a function that takes a world w to its
 263 modality $\Box_w \in \mathcal{P}_w \rightarrow \mathbb{P}$. We often write $\Box_w(w'. P)$ for $\Box_w \lambda w'. P$. As in [7], to guarantee that
 264 this interpretation yields a standard type system in the sense of Thm. 12, we require that
 265 the modalities satisfy certain properties reminiscent of standard modal axiom schemata [11],
 266 which we repeat here for ease of read:

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

- 269 ■ \Box_1 (monotonicity of \Box): $\forall(w : \mathcal{W})(P : \mathcal{P}_w). \forall w' \sqsupseteq w. \Box_w P \rightarrow \Box_{w'} P$.
- 270 ■ \Box_2 (K , distribution axiom): $\forall(w : \mathcal{W})(P, Q : \mathcal{P}_w). \Box_w (w'.P \ w' \rightarrow Q \ w') \rightarrow \Box_w P \rightarrow \Box_w Q$
- 271 ■ \Box_3 (CA , i.e., \Box follows from $\Box\Box$): $\forall(w : \mathcal{W})(P : \mathcal{P}_w). \Box_w (w'.\Box_{w'} P) \rightarrow \Box_w P$
- 272 ■ \Box_4 : $\forall(w : \mathcal{W})(P : \mathcal{P}_w). \forall_w^\Xi(P) \rightarrow \Box_w P$
- 273 ■ \Box_5 (T , reflexivity axiom): $\forall(w : \mathcal{W})(P : \mathbb{P}). \Box_w (w'.P) \rightarrow P$

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

transitivity:	$w \Vdash T_1 \equiv T_2 \rightarrow w \Vdash T_2 \equiv T_3 \rightarrow w \Vdash T_1 \equiv T_3$	$w \Vdash t_1 \equiv t_2 \in T \rightarrow w \Vdash t_2 \equiv t_3 \in T \rightarrow w \Vdash t_1 \equiv t_3 \in T$
symmetry:	$w \Vdash T_1 \equiv T_2 \rightarrow w \Vdash T_2 \equiv T_1$	$w \Vdash t_1 \equiv t_2 \in T \rightarrow w \Vdash t_2 \equiv t_1 \in T$
computation:	$w \Vdash T \equiv T' \rightarrow T \Downarrow_{!w} T' \rightarrow w \Vdash T \equiv T'$	$w \Vdash t \equiv t' \in T \rightarrow t \Downarrow_{!w} t' \rightarrow w \Vdash t \equiv t' \in T$
monotonicity:	$w \Vdash T_1 \equiv T_2 \rightarrow w \sqsubseteq w' \rightarrow w' \Vdash T_1 \equiv T_2$	$w \Vdash t_1 \equiv t_2 \in T \rightarrow w \sqsubseteq w' \rightarrow w' \Vdash t_1 \equiv t_2 \in T$
locality:	$\Box_w (w'.w' \Vdash T_1 \equiv T_2) \rightarrow w \Vdash T_1 \equiv T_2$	$\Box_w (w'.w' \Vdash t_1 \equiv t_2 \in T) \rightarrow w \Vdash t_1 \equiv t_2 \in T$
consistency:	$\neg w \Vdash t \in \text{False}$	

274 **Proof.** The proof relies on the properties of the equality modality. For example: \Box_1 is used
 275 to prove monotonicity when $w \Vdash T_1 \equiv T_2$ is derived by closing under \Box_w ; \Box_2 and \Box_4 are used,
 276 e.g., to prove the symmetry and transitivity of $w \Vdash t \equiv t' \in \text{Nat}$; \Box_3 is used to prove locality;
 277 and \Box_5 is used to prove consistency. See [props3.lagda](#) for further details. ◀

278 As indicated in Thm. 12, and as opposed to the counterpart of the theorem in [7],
 279 $w \Vdash T \equiv T$ and $w \Vdash t \equiv t \in T$ are no longer closed under all computations. For example, when
 280 $T := \text{Nat}$, 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
 281 $t := (\text{choose}(\delta, \underline{1}); \text{if } !\delta < \underline{1} \text{ then } \underline{0} \text{ else } \underline{1})$, which reduces to $t' := (\text{if } !\delta < \underline{1} \text{ then } \underline{0} \text{ else } \underline{1})$
 282 and also to $\underline{1}$ in all worlds, but t' does not reduce to $\underline{1}$ in all worlds, because δ could be
 283 initialized differently in different worlds. However, the following holds by transitivity of \Downarrow_w :
 284 $t' \Downarrow_w t \rightarrow w \Vdash t \equiv t \in \text{Nat} \rightarrow w \Vdash t \equiv t' \in \text{Nat}$. Similarly, the following also holds by transitivity
 285 of \Downarrow_w : $w \Vdash T \equiv T \rightarrow T' \Downarrow_w T \rightarrow w \Vdash T \equiv T'$. Finally, note that, as indicated in Thm. 12, this
 286 semantics is closed under β -reduction, as β -reduction does not modify the current world.

287 4 Proof of Continuity

288 We can now state the version of Brouwer's continuity principle that we validate in this
 289 paper, along with its realizer. For this we first introduce the following notation: $\Pi_{\mathcal{P}} a : A.B :=$
 290 $\Pi a : (A \cap \text{pure}).B$, which quantifies over pure elements of type A .

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

$$293 \quad \Pi_{\mathcal{P}} F : \mathcal{B} \rightarrow \text{Nat}. \Pi_{\mathcal{P}} \alpha : \mathcal{B}. \downarrow \Sigma n : \text{Nat}. \Pi_{\mathcal{P}} \beta : \mathcal{B}. (\alpha = \beta \in \mathcal{B}_n) \rightarrow (F(\alpha) = F(\beta) \in \text{Nat}) \quad (1)$$

294 and is inhabited by

$$295 \quad \lambda F. \lambda \alpha. \langle \text{mod}(F, \alpha), \lambda \beta. \lambda e. \star \rangle \quad (2)$$

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

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

35:10 Realizing Continuity Using Stateful Computations

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

$$w \models \lambda F. \lambda \alpha. \langle \text{mod}(F, \alpha), \lambda \beta. \lambda e. \star \rangle \in \text{CONT}_p$$

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

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

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

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

316 4.1 Purity

317 According to Nat 's semantics, to prove that `mod(F, α) ∈ Nat` w.r.t. a world w , we have to
 318 prove it computes to the same number in all extensions of w . However, this will not be the
 319 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 ,
 320 then it could happen that f gets applied to 0 in some world w_1 if $!\delta_0$ returns 0, and to 1 in
 321 some world $w_2 \sqsupseteq w_1$ if $!\delta_0$ returns 1. As `mod(F, α)` returns the highest number that F applies
 322 its argument to, then `mod(F, α)` would in this instance return different numbers in different
 323 extensions, and would therefore not inhabit Nat .

324 Therefore, to validate a version of continuity which requires the modulus of continuity
 325 to be time-invariant as in Eq. (1), one can require that both F and α are pure (i.e.,
 326 name-free) terms. Thanks to $\mathbf{\Pi}_p$, we get to assume that both F and α are in `pure` and
 327 therefore are name-free. Note that it would not be enough to use the following pattern:
 328 $\mathbf{\Pi} F: \mathcal{B} \rightarrow \text{Nat}. (F = F \in \text{pure}) \rightarrow \dots$, because then for the continuity principle to even be a
 329 type, we would have to prove that F is name-free to prove that $F = F \in \text{pure}$ is a type, only
 330 knowing that $F \in \mathcal{B} \rightarrow \text{Nat}$, 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 $\mathcal{B}_{\leq n} = \{x : \text{Nat} \mid x <_{\leq} n\} \rightarrow \text{Nat}$,
 assuming the existence of some type $x <_{\leq} n$ that can relate an $x \in \text{Nat}$ with an $n \in \mathcal{N}\text{at}$:

$$\mathbf{\Pi} F: \mathcal{B} \rightarrow \text{Nat}. \mathbf{\Pi} \alpha: \mathcal{B}. \downarrow \Sigma n: \mathcal{N}\text{at}. \mathbf{\Pi} \beta: \mathcal{B}. (\alpha = \beta \in \mathcal{B}_{\leq n}) \rightarrow (F(\alpha) = F(\beta) \in \text{Nat})$$

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

- 335 (1) Proving that $\text{mod}(F, \alpha) \in \mathfrak{z}\text{Nat}$, which is now straightforward.
 336 (2) Next, we have to prove that $\prod \beta : \mathcal{B}. (\alpha = \beta \in \mathcal{B}_{\mathfrak{z}\text{mod}(F, \alpha)}) \rightarrow (F(\alpha) = F(\beta) \in \text{Nat})$, i.e.,
 337 given $\beta \in \mathcal{B}$ such that $\alpha = \beta \in \mathcal{B}_{\mathfrak{z}\text{mod}(F, \alpha)}$, we have to prove $F(\alpha) = F(\beta) \in \text{Nat}$. The assumption
 338 $\alpha = \beta \in \mathcal{B}_{\mathfrak{z}\text{mod}(F, \alpha)}$ tells us that given $k \in \text{Nat}$ such that $k <_{\mathfrak{z}} \text{mod}(F, \alpha)$, $\alpha(k) = \beta(k) \in \text{Nat}$. As
 339 mentioned above, for $k <_{\mathfrak{z}} \text{mod}(F, \alpha)$ to be true, it must be that k is less than $\text{mod}(F, \alpha)$ in
 340 all extensions of the current world. However, without the purity constraint, $\text{mod}(F, \alpha)$ can
 341 compute to different numbers in different extensions.

342 Going back to our goal $F(\alpha) = F(\beta) \in \text{Nat}$, given the semantics of Nat presented in Fig. 2,
 343 to prove this it is enough to assume that $F(\text{upd}(\delta, \alpha))$ computes to some number \underline{m} in
 344 some world w , and to prove that $F(\beta)$ also computes to \underline{m} in w . We can then inspect
 345 the computation $F(\text{upd}(\delta, \alpha)) \Downarrow_{w_1}^w \underline{k}$, where δ is the name generated by $\text{mod}(F, \alpha)$, and
 346 show that it can be converted into a computation $F(\beta) \Downarrow_{w_2}^w \underline{k}$, by replacing $\alpha(\underline{i})$ with $\beta(\underline{i})$,
 347 whenever we encounter such an expression. To do this, we need to know that $\alpha(\underline{i})$ and $\beta(\underline{i})$
 348 compute to the same number using $\alpha = \beta \in \mathcal{B}_{\mathfrak{z}\text{mod}(F, \alpha)}$. However, we only know that \underline{i} is less
 349 than $\text{mod}(F, \alpha)$ in w , which is not enough to use this assumption, as \underline{i} might be greater than
 350 $\text{mod}(F, \alpha)$ in some other world w' . We can address this issue using classical logic to prove
 351 that there exists a $w' \sqsupseteq w$ such that for all $w'' \sqsupseteq w$, the smallest number that α is applied to
 352 in the computation of $\text{mod}(F, \alpha)$ w.r.t. w' is less than the number that $\text{mod}(F, \alpha)$ computes
 353 to w.r.t. w'' . In the argument sketched above we can then use w' instead of w .

354 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):

$$\begin{aligned}
 (\text{Ass}_1) \quad & \forall (w : \mathcal{W})(P : \mathcal{P}_w). \Box_w P \rightarrow \exists_w^E(P) \\
 (\text{Ass}_2) \quad & \forall (\delta : \mathcal{N})(w : \mathcal{W})(n : \mathbb{N}). \text{comp}(\delta, w, r) \\
 & \rightarrow \forall_{\text{update}(w, \delta, \underline{n})}^E (w'. \exists (k : \mathbb{N}). \text{choice?}(w', \delta) = k) \\
 (\text{Ass}_3) \quad & \forall (\delta : \mathcal{N})(w : \mathcal{W})(k : \mathbb{N}). \text{comp}(\delta, w, r) \rightarrow \text{choice?}(\text{update}(w, \delta, \underline{k}), \delta) = k
 \end{aligned}$$

355 Ass_1 requires the modality \Box to be non-empty in the sense that for $\Box_w P$ to be true, it
 356 has to be true for at least one extension of w . This is true about all topological bar spaces
 357 (see $\rightarrow \exists \mathcal{W}$ in `mod.lagda`), and therefore about the Kripke, Beth, and Open modalities which are
 358 derived from such spaces [7, Sec.6.2]. Ass_2 requires that the “last” choice of a r -compatible
 359 choice name δ is indeed a number. Ass_3 guarantees that retrieving a choice that was just
 360 made will return that choice.

361 The last two assumptions are true about `Ref`, the formalization of references to numbers
 362 presented in Ex. 7 (see for example `contInstanceKripkeRef.lagda` for the proof that TT_C^\square instan-
 363 tiated with a Kripke modality and references satisfies these properties). In addition both are
 364 true about another kind of stateful computations, namely a variant of the formalization of
 365 free choice sequences [16; 3; 29; 30; 18; 32; 22] presented in [7, Ex.5], where new choices are
 366 pre-pended as opposed to being appended in [7] (see for example `contInstanceKripkeCS.lagda`
 367 for the proof that TT_C^\square instantiated with a Kripke modality and choice sequences satisfies
 368 these properties).

369 **4.3 The Modulus is a Number**

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

372 **► Theorem 14** (The Modulus is a Number). *If $\text{namefree}(F)$, $\text{namefree}(\alpha)$, $w \models F \in \text{Nat}^{\mathcal{B}}$,
 373 and $w \models \alpha \in \mathcal{B}$, for some world w , then*

$$374 \quad \Box_w(w'. \exists(n : \mathbb{N}). \text{mod}(F, \alpha) \Downarrow_{w'} \underline{n}) \quad (3)$$

375 To prove the above, we will make use of the fact that $w \models \text{upd}(\delta, \alpha) \in \mathcal{B}$ and therefore also
 376 $w \models F(\text{upd}(\delta, \alpha)) \in \text{Nat}$, i.e., by the semantics of Nat presented in Fig. 2, we have for some
 377 fresh name δ :

$$378 \quad \Box_w(w'. \exists(n : \mathbb{N}). F(\text{upd}(\delta, \alpha)) \Downarrow_{w'} \underline{n}). \quad (4)$$

379 But for this we first need to start computing $\text{mod}(F, \alpha)$ to generate a fresh name δ according
 380 to the current world. If that current world is some world $w' \sqsupseteq w$ (obtained, for example, using
 381 \Box_4 from Def. 11 on Eq. (3)), then we need to be able to get that $F(\text{upd}(\delta, \alpha))$ computes to a
 382 number w.r.t. w' , which Eq. (4) might not provide. This is the reason for assumption `Ass1`.

383 Going back to the proof of Eq. (3), we use \Box_4 , and have to prove $\exists(n : \mathbb{N}). \text{mod}(F, \alpha) \Downarrow_{w_1} \underline{n}$
 384 for some $w_1 \sqsupseteq w$. We then:

- 385 ■ (A) first have to find a number n such that $\text{mod}(F, \alpha)$ computes to \underline{n} w.r.t. w_1 ,
- 386 ■ (B) and then that it does so also for all $w'_1 \sqsupseteq w_1$.

387 Let us prove (A) first. We now start computing $\text{mod}(F, \alpha)$ w.r.t. w_1 . We generate a
 388 fresh name $\delta := \nu\mathcal{C}(w_1)$, and have to prove that $\text{choose}(\delta, \underline{0}); F(\text{upd}(\delta, \alpha)); !\delta + \underline{1}$ computes
 389 to a number w.r.t. $w_2 := \text{start}\nu\mathcal{C}(w_1, r)$ that satisfies $\text{comp}(\delta, w_2, r)$ (by the properties of
 390 `start` $\nu\mathcal{C}$ presented in Def. 5). We keep computing this expression and have to prove that
 391 $F(\text{upd}(\delta, \alpha)); !\delta + \underline{1}$ computes to a number w.r.t. $w_3 := \text{update}(w_2, \delta, \underline{0})$.

392 From `Ass1` and Eq. (4), we obtain $w_5 \sqsupseteq w$ and $n \in \mathbb{N}$ such that $F(\text{upd}(\delta, \alpha)) \Downarrow_{w_5} \underline{n}$,
 393 from which we obtain by definition that there exists a w_6 such that $F(\text{upd}(\delta, \alpha)) \Downarrow_{w_6}^{w_5} \underline{n}$.
 394 Now, because F and α are name-free, we can derive that there exists a w_4 such that
 395 $F(\text{upd}(\delta, \alpha)) \Downarrow_{w_4}^{w_3} \underline{n}$ (see `differNF` \Downarrow `APPLY-upd` in `terms7.lagda`). It now remains to prove that
 396 $\underline{n}; !\delta + \underline{1}$, computes to a number w.r.t. w_4 . It is then enough to prove that $!\delta$ computes to
 397 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
 398 make use of `Ass2` which states that r constrains the δ -choices to be numbers. Using this and
 399 the facts that $\text{comp}(\delta, w_2, r)$ and $w_2 \sqsubseteq w_4$ (by \sqsubseteq 's transitivity since $w_3 \sqsubseteq w_4$ by Lem. 10 and
 400 $w_2 \sqsubseteq w_3$ by Def. 6), we deduce that there exists a $k \in \mathbb{N}$ such that $\text{choice?}(w_4, \delta) = \underline{k}$, and
 401 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
 402 the proof of (A).

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

406 To prove (B) we assume a $w'_1 \sqsupseteq w_1$ and have to prove that $\text{mod}(F, \alpha)$ computes to $\underline{k+1}$
 407 w.r.t. w'_1 . As before, we start computing $\text{mod}(F, \alpha)$ w.r.t. w'_1 , and generate a fresh name
 408 $\delta' := \nu\mathcal{C}(w'_1)$, and have to prove that $F(\text{upd}(\delta', \alpha)); !\delta' + \underline{1}$ computes to $\underline{k+1}$ w.r.t. $w'_3 :=$
 409 $\text{update}(w'_2, \delta', \underline{0})$, where $w'_2 := \text{start}\nu\mathcal{C}(w'_1, r)$. As F and α are name-free, $t_1 := F(\text{upd}(\delta, \alpha))$
 410 and $t_2 := F(\text{upd}(\delta', \alpha))$ behave the same except that when t_1 updates δ with a number, t_2
 411 updates δ' with that number.

412 Using a syntactic simulation method, we will prove that because t_1 and t_2 are “similar”
 413 (which is captured by Def. 15 below), $\text{choice?}(w_3, \delta) = \text{choice?}(w'_3, \delta')$, and $t_1 \Downarrow_{w_4}^{w_3} t'_1$, then
 414 $t_2 \Downarrow_{w'_4}^{w'_3} t'_2$ such that t'_1 and t'_2 are also “similar” and $\text{choice?}(w_4, \delta) = \text{choice?}(w'_4, \delta')$. Note
 415 that $\text{choice?}(w_3, \delta)$ and $\text{choice?}(w'_3, \delta')$ return the same choice because $\text{choice?}(w_3, \delta) =$
 416 $\text{choice?}(\text{update}(w_2, \delta, \underline{0}), \delta) = 0$ and $\text{choice?}(w'_3, \delta') = \text{choice?}(\text{update}(w'_2, \delta', \underline{0}), \delta') = 0$. To
 417 derive these equalities, we need assumption Ass_3 that relates choice? and update .

418 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*

$$\begin{aligned} & (t_1 = \text{upd}(\delta_1, \alpha) \wedge t_2 = \text{upd}(\delta_2, \alpha)) \\ & \vee (t_1 = x \wedge t_2 = x) \vee (t_1 = \star \wedge t_2 = \star) \vee (t_1 = \underline{n} \wedge t_2 = \underline{n}) \\ & \vee (t_1 = \lambda x.a \wedge t_2 = \lambda x.b \wedge a \sim_{\delta_1, \delta_2, \alpha} b) \\ & \vee (t_1 = (a_1 \ b_1) \wedge t_2 = (a_2 \ b_2) \wedge a_1 \sim_{\delta_1, \delta_2, \alpha} a_2 \wedge b_1 \sim_{\delta_1, \delta_2, \alpha} b_2) \\ & \vee \dots \end{aligned}$$

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

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

425 ► **Lemma 16.** *If $t_1 \sim_{\delta_1, \delta_2, \alpha} t_2$, $\text{choice?}(w_1, \delta_1) = \text{choice?}(w_2, \delta_2)$, $t_1 \Downarrow_{w'_1}^{w_1} t'_1$, `namefree`(α),*
 426 *`comp`(δ_1, w_1, r), and `comp`(δ_2, w_2, r), then there exist w'_2 and t'_2 such that $t_2 \Downarrow_{w'_2}^{w_2} t'_2$, $t'_1 \sim_{\delta_1, \delta_2, \alpha}$*
 427 *t'_2 , and $\text{choice?}(w'_1, \delta_1) = \text{choice?}(w'_2, \delta_2)$.*

428 which we prove by induction on the computation $t_1 \Downarrow_{w'_1}^{w_1} t'_1$.

429 We therefore obtain that there exist t'_2 and w'_4 such that $F(\text{upd}(\delta', \alpha)) \Downarrow_{w'_4}^{w'_3} t'_2$, $\underline{n} \sim_{\delta, \delta', \alpha} t'_2$
 430 and $\text{choice?}(w'_4, \delta') = \text{choice?}(w_4, \delta) = \underline{k}$. Furthermore, by definition of the similarity relation,
 431 $t'_2 = \underline{n}$. We obtain that $F(\text{upd}(\delta', \alpha)); !\delta' + \underline{1} \Downarrow_{w'_4}^{w'_3} \underline{n}; !\delta' + \underline{1}$ and so $F(\text{upd}(\delta', \alpha)); !\delta' + \underline{1} \Downarrow_{w'_4}^{w'_3} !\delta' + \underline{1}$.
 432 Because $\text{choice?}(w'_4, \delta') = \underline{k}$, we finally obtain $F(\text{upd}(\delta', \alpha)); !\delta' + \underline{1} \Downarrow_{w'_4}^{w'_3} \underline{k} + \underline{1}$, which concludes
 433 the proof of (B), and therefore that $\text{mod}(F, \alpha) \in \text{Nat}$.

4.4 The Modulus is the Highest Number

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

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

440 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
 441 $\text{mod}(F, \alpha) \Downarrow_{w_2}^{w_1} \underline{k} + \underline{1}$. As in Sec. 4.3, we start computing $\text{mod}(F, \alpha)$ w.r.t. the current world w_1 ,
 442 and generate a fresh name $\delta := \nu \mathcal{C}(w_1)$, and deduce that

$$443 \quad F(\text{upd}(\delta, \alpha)); !\delta + \underline{1} \Downarrow_{w_2}^{w_1} \underline{k} + \underline{1} \tag{5}$$

35:14 Realizing Continuity Using Stateful Computations

444 where $w_1' := \text{start}\nu\mathcal{C}(w_1, r)$ and $w_1'' := \text{update}(w_1', \delta, \underline{0})$. Furthermore, by [Ass₂](#), there must be
 445 a $n \in \mathbb{N}$ such that $\text{choice?}(w_2, \delta) = \underline{n}$.

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

453 **► Lemma 18.** *Let α be a closed name-free term, and t be a term such that $\text{Upd}_{\delta, \alpha}(t)$ and
 454 $t \Downarrow_{w_2}^{w_1} u$, and let $\text{choice?}(w_2, \delta) = \underline{n}$, such that $n < m$, then for any world w along the
 455 computation $t \Downarrow_{w_2}^{w_1} u$ if $\text{choice?}(w, \delta) = \underline{i}$ then $i < m$.*

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

458 4.5 The Modulus is the Modulus

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

461 **► Theorem 19** (The Modulus is the Modulus). *If $w \models \alpha \equiv \beta \in \mathcal{B}_n$ then $w \models F(\alpha) \equiv F(\beta) \in \mathbb{N}\text{at}$.*

462 First, we prove that $w \models F(\alpha) \equiv F(\text{upd}(\delta, \alpha)) \in \mathbb{N}\text{at}$, which follows from the semantics of **Π** and
 463 **Nat** presented in Fig. 2, and in particular the fact that $w \models \alpha \equiv \text{upd}(\delta, \alpha) \in \mathcal{B}$. It is therefore
 464 enough to prove that $F(\text{upd}(\delta, \alpha))$ and $F(\beta)$ are equal in **Nat**. Relating $F(\text{upd}(\delta, \alpha))$ and
 465 $F(\beta)$ instead of $F(\alpha)$ and $F(\beta)$ allows getting access to the values that α gets applied to in
 466 the computation $F(\alpha)$ as they are recorded using the choice name δ . We can then use these
 467 values to prove that $F(\text{upd}(\delta, \alpha))$ and $F(\beta)$ behave similarly up to applications of α in the
 468 first computation, which are applications of β in the second, and that these applications
 469 reduce to the same numbers because the arguments, recorded using δ , are less than $\text{mod}(F, \alpha)$.

However, even though $\text{upd}(\delta, \alpha)$ and α are equal in \mathcal{B} , they behave slightly differently
 computationally as $\text{upd}(\delta, \alpha)$ 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
 $F(\text{upd}(\delta, \alpha))$ 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
cbv be defined as follows:

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

470 It is straightforward to derive that $w \models F(\beta) \equiv F(\text{cbv}(\beta)) \in \mathbb{N}\text{at}$ from the semantics of **Π** and
 471 **Nat** presented in Fig. 2. It is therefore enough to prove that $F(\text{upd}(\delta, \alpha))$ and $F(\text{cbv}(\beta))$
 472 are equal in **Nat**.

473 Because $F(\text{upd}(\delta, \alpha)) \Downarrow_{w'}^w \underline{n}$, by Lem. 18 for any world w_0 along this computation if
 474 $\text{choice?}(w_0, \delta) = \underline{i}$ then $i < k+1$, where $k+1$ is the number computed by $\text{mod}(F, \alpha)$.

475 We now prove that $F(\text{upd}(\delta, \alpha))$ and $F(\text{cbv}(\beta))$ both compute to \underline{n} through another
 476 simulation proof that relies on the following relation (see [updRel](#) in [continuity4.lagda](#) for details):
 477

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

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

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

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

485 ► **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*
 486 *this computation if `choice?`(w_0, δ) = i then $i < k+1$, then $t_2 \Downarrow_w^w t_2'$ such that $t_1' \approx_{\delta, \alpha, \beta} t_2'$.*

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

490 5 Conclusion and Related Works

491 We have shown in this paper how to validate a continuity principle for a subset of the $\mathbb{T}\mathbb{T}_{\mathcal{C}}^{\square}$
 492 family of type theories, such that the modulus of continuity of functions is internalized, i.e.,
 493 computed using an expression of the underlying computation system. In particular, we have
 494 used stateful computations, and have discussed some of the challenges arising from such
 495 impure computations. As mentioned in the introduction, and as recalled below, this is not
 496 the first proof of continuity, however to the best of our knowledge, this is the first proof of an
 497 “internal” validity proof of continuity that relies on stateful computations. Furthermore, the
 498 proof presented above relies on an “internal” notion of probing through the use of stateful
 499 computations internal to the computation language of the type theory, while approaches
 500 such as [10; 9; 34] rely on a meta-theoretic (or “external”) notion of probing.

501 Troelstra proved in [28, p.158] that every closed term $t \in \mathbb{N}^{\mathbb{B}}$ of $\mathbf{N}\text{-HA}^{\omega}$ has a provable
 502 modulus of continuity in $\mathbf{N}\text{-HA}^{\omega}$ —see also [5] for similar proofs of the consistency of continuity
 503 with various constructive theories.

504 Coquand and Jaber [10; 9] proved the *uniform* continuity of a Martin-Löf-like intensional
 505 type theory using forcing. Their method consists in adding a generic element \mathbf{f} as a constant
 506 to the language that stands for a Cohen real of type $2^{\mathbb{N}}$, and defining the forcing conditions
 507 as approximations of \mathbf{f} . They then define a suitable *computability* predicate that expresses
 508 when a term is a computable term of some type up to approximations given by the forcing
 509 conditions. The key steps are to (1) first prove that \mathbf{f} is computable and then (2) prove that
 510 well-typed terms are computable, from which they derive uniform continuity: the uniform
 511 modulus of continuity is given by the approximations.

512 Similarly, Escardó and Xu [34] proved that the definable functionals of Gödel’s system \mathbf{T} [15]
 513 are uniformly continuous on the Cantor space \mathcal{C} (without assuming classical logic or the Fan
 514 Theorem). For that, they developed the $\mathbf{C}\text{-Space}$ category, which internalizes

515 continuity, and has a *Fan functional* which computes the modulus of uniform continuity of
 516 functions in $\mathcal{C} \rightarrow \mathbb{N}$. Relating C-Space and the standard set-theoretical model of system T,
 517 they show that all System T functions on the Cantor space are uniformly continuous. Fur-
 518 thermore, using this model, they show how to extract computational content from proofs in
 519 HA^ω extended with a uniform continuity axiom, which is realized by the Fan functional.

520 In [14], Escardó proves that all System T functions are continuous on the Baire space
 521 and uniformly continuous on the Cantor space without using forcing. Instead, he provides
 522 an alternative interpretation of system T, where a number is interpreted by a *dialogue tree*,
 523 which captures the computation of a function w.r.t. an oracle of type \mathcal{B} . Escardó first proves
 524 that such computations are continuous, and then by defining a suitable relation between
 525 the standard interpretation and the alternative one, that relates the interpretations of all
 526 system T terms, derives that for all system T functions on the Baire space are continuous.

527 In [25; 26], the authors proved that Brouwer’s continuity principle is consistent with
 528 Nuprl [8; 2] by realizing the modulus of continuity of functions on the Baire space also using
 529 Longley’s method [21], but using exceptions instead of references. The realizer there is more
 530 complicated than the one presented in this paper as it involves an effectful computation that
 531 repeatedly checks whether a given number is at least as high as the modulus of continuity,
 532 and increasing that number until the modulus of continuity is reached. We do not require
 533 such a loop, and can directly extract the modulus of continuity of a function.

534 In [4] the authors prove that all BTT [23] functions are continuous by generalizing the
 535 method used in [14]. Their model is built in three steps as follows: an axiom model/translation
 536 adds an oracle to the theory at hand; a branching model/translation interprets types as
 537 intensional D-algebras, i.e., as types equipped with pythias; and an algebraic parametricity
 538 model/translation that relates the two previous translations by relating the calls to the
 539 pythia to the oracle. Their models allows deriving that all functions are continuous, but does
 540 not allow “internalizing” the continuity principle, which is the goal of this paper.

541 References

- 542 [1] *Agda Wiki*. <http://wiki.portal.chalmers.se/agda/pmwiki.php>.
- 543 [2] Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori
 544 Lorigo and Evan Moran. “Innovations in computational type theory using Nuprl”. In: *J.*
 545 *Applied Logic* 4.4 (2006). <http://www.nuprl.org/>, pp. 428–469.
- 546 [3] Mark van Atten and Dirk van Dalen. “Arguments for the continuity principle”. In: *Bulletin of*
 547 *Symbolic Logic* 8.3 (2002), pp. 329–347.
- 548 [4] Martin Baillon, Assia Mahboubi and Pierre-Marie Pédro. “Gardening with the Pythia A
 549 Model of Continuity in a Dependent Setting”. In: *CSL*. Vol. 216. LIPIcs. Schloss Dagstuhl -
 550 Leibniz-Zentrum für Informatik, 2022, 5:1–5:18. DOI: [10.4230/LIPIcs.CSL.2022.5](https://doi.org/10.4230/LIPIcs.CSL.2022.5).
- 551 [5] Michael J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.
- 552 [6] Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. London Mathe-
 553 matical Society Lecture Notes Series. Cambridge University Press, 1987.
- 554 [7] Liron Cohen and Vincent Rahli. “Constructing Unprejudiced Extensional Type Theories with
 555 Choices via Modalities”. In: *FSCD*. Vol. 228. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für
 556 Informatik, 2022, 10:1–10:23. DOI: [10.4230/LIPIcs.FSCD.2022.10](https://doi.org/10.4230/LIPIcs.FSCD.2022.10).
- 557 [8] Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, Robert
 558 W. Harper, Douglas J. Howe, Todd B. Knoblock, Nax P. Mendler, Prakash Panangaden, James
 559 T. Sasaki and Scott F. Smith. *Implementing mathematics with the Nuprl proof development*
 560 *system*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.
- 561 [9] Thierry Coquand and Guilhem Jaber. “A Computational Interpretation of Forcing in Type
 562 Theory”. In: *Epistemology versus Ontology*. Vol. 27. Logic, Epistemology, and the Unity of
 563 Science. Springer, 2012, pp. 203–213. DOI: [10.1007/978-94-007-4435-6_10](https://doi.org/10.1007/978-94-007-4435-6_10).

- 564 [10] Thierry Coquand and Guilhem Jaber. “A Note on Forcing and Type Theory”. In: *Fundam.*
565 *Inform.* 100.1-4 (2010), pp. 43–52. DOI: [10.3233/FI-2010-262](https://doi.org/10.3233/FI-2010-262).
- 566 [11] M. J. Cresswell and G. E. Hughes. *A New Introduction to Modal Logic*. Routledge, 1996.
- 567 [12] Michael A. E. Dummett. *Elements of Intuitionism*. Second. Clarendon Press, 2000.
- 568 [13] Martín H. Escardó and Chuangjie Xu. “The Inconsistency of a Brouwerian Continuity Principle
569 with the Curry-Howard Interpretation”. In: *TLCA 2015*. Vol. 38. LIPIcs. Schloss Dagstuhl -
570 Leibniz-Zentrum fuer Informatik, 2015, pp. 153–164. DOI: [10.4230/LIPIcs.TLCA.2015.153](https://doi.org/10.4230/LIPIcs.TLCA.2015.153).
- 571 [14] Martín Hötzel Escardó. “Continuity of Gödel’s System T Definable Functionals via Effectful
572 Forcing”. In: *Electr. Notes Theor. Comput. Sci.* 298 (2013), pp. 119–141. DOI: [10.1016/j.
573 entcs.2013.09.010](https://doi.org/10.1016/j.entcs.2013.09.010).
- 574 [15] Jean-Yves Girard, Paul Taylor and Yves Lafont. *Proofs and Types*. Cambridge University
575 Press, 1989.
- 576 [16] Stephen C. Kleene and Richard E. Vesley. *The Foundations of Intuitionistic Mathematics,*
577 *especially in relation to recursive functions*. North-Holland Publishing Company, 1965.
- 578 [17] Georg Kreisel. “On weak completeness of intuitionistic predicate logic”. In: *J. Symb. Log.* 27.2
579 (1962), pp. 139–158. DOI: <http://dx.doi.org/10.2307/2964110>.
- 580 [18] Georg Kreisel and Anne S. Troelstra. “Formal systems for some branches of intuitionistic
581 analysis”. In: *Annals of Mathematical Logic* 1.3 (1970), pp. 229–387. DOI: [http://dx.doi.
582 org/10.1016/0003-4843\(70\)90001-X](http://dx.doi.org/10.1016/0003-4843(70)90001-X).
- 583 [19] Saul A. Kripke. “Semantical Analysis of Intuitionistic Logic I”. In: *Formal Systems and*
584 *Recursive Functions*. Vol. 40. Studies in Logic and the Foundations of Mathematics. Elsevier,
585 1965, pp. 92–130. DOI: [https://doi.org/10.1016/S0049-237X\(08\)71685-9](https://doi.org/10.1016/S0049-237X(08)71685-9).
- 586 [20] Saul A. Kripke. “Semantical Analysis of Modal Logic I. Normal Propositional Calculi”. In:
587 *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 9.5-6 (1963), pp. 67–96.
588 DOI: [10.1002/malq.19630090502](https://doi.org/10.1002/malq.19630090502).
- 589 [21] John Longley. “When is a Functional Program Not a Functional Program?” In: *ICFP’99*.
590 ACM, 1999, pp. 1–7. DOI: [10.1145/317636.317775](https://doi.org/10.1145/317636.317775).
- 591 [22] Joan R. Moschovakis. “An intuitionistic theory of lawlike, choice and lawless sequences”. In:
592 *Logic Colloquium’90: ASL Summer Meeting in Helsinki*. Association for Symbolic Logic. 1993,
593 pp. 191–209.
- 594 [23] Pierre-Marie Pédrot and Nicolas Tabareau. “An effectful way to eliminate addiction to
595 dependence”. In: *LICS 2017*. IEEE Computer Society, 2017, pp. 1–12. DOI: [10.1109/LICS.
596 2017.8005113](https://doi.org/10.1109/LICS.2017.8005113).
- 597 [24] Andrew M Pitts. *Nominal Sets: Names and Symmetry in Computer Science, volume 57 of*
598 *Cambridge Tracts in Theoretical Computer Science*. 2013.
- 599 [25] Vincent Rahli and Mark Bickford. “A nominal exploration of intuitionism”. In: *CPP 2016*.
600 Extended version: <http://www.nuprl.org/html/Nuprl2Coq/continuity-long.pdf>. ACM,
601 2016, pp. 130–141. DOI: [10.1145/2854065.2854077](https://doi.org/10.1145/2854065.2854077).
- 602 [26] Vincent Rahli and Mark Bickford. “Validating Brouwer’s continuity principle for numbers
603 using named exceptions”. In: *Mathematical Structures in Computer Science* (2017), pp. 1–49.
604 DOI: [10.1017/S0960129517000172](https://doi.org/10.1017/S0960129517000172).
- 605 [27] A.S. Troelstra. “A Note on Non-Extensional Operations in Connection With Continuity and
606 Recursiveness”. In: *Indagationes Mathematicae* 39.5 (1977), pp. 455–462. DOI: [10.1016/1385-
607 7258\(77\)90060-9](https://doi.org/10.1016/1385-7258(77)90060-9).
- 608 [28] A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. New
609 York, Springer, 1973.
- 610 [29] Anne S. Troelstra. “Choice Sequences and Informal Rigour”. In: *Synthese* 62.2 (1985), pp. 217–
611 227.
- 612 [30] Anne S. Troelstra. *Choice sequences: a chapter of intuitionistic mathematics*. Clarendon Press
613 Oxford, 1977.
- 614 [31] Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics An Introduction*.
615 Vol. 121. Studies in Logic and the Foundations of Mathematics. Elsevier, 1988.
- 616 [32] Wim Veldman. “Understanding and Using Brouwer’s Continuity Principle”. In: *Reuniting*
617 *the Antipodes — Constructive and Nonstandard Views of the Continuum*. Vol. 306. Synthese
618 Library. Springer Netherlands, 2001, pp. 285–302. DOI: [10.1007/978-94-015-9757-9_24](https://doi.org/10.1007/978-94-015-9757-9_24).

35:18 REFERENCES

- 619 [33] Chuangjie Xu. “A continuous computational interpretation of type theories”. PhD thesis.
620 University of Birmingham, UK, 2015.
- 621 [34] Chuangjie Xu and Martín Hötzel Escardó. “A Constructive Model of Uniform Continuity”. In:
622 *TLCA 2013*. Vol. 7941. LNCS. Springer, 2013, pp. 236–249. doi: [10.1007/978-3-642-38946-](https://doi.org/10.1007/978-3-642-38946-7_18)
623 [7_18](https://doi.org/10.1007/978-3-642-38946-7_18).