Inductive Continuity via Brouwer Trees

Liron Cohen
Ben-Gurion University, Israel

Bruno da Rocha Paiva
University of Birmingham, UK

Vincent Rahli
University of Birmingham, UK

Ayberk Tosun
University of Birmingham, UK

Abstract

Continuity is a key principle of intuitionistic logic that is generally accepted by constructivists but is inconsistent with classical logic. Most commonly, continuity states that a function from the Baire space to numbers, only needs approximations of the points in the Baire space to compute. More recently, another formulation of the continuity principle was put forward. It states that for any function F from the Baire space to numbers, there exists a (dialogue) tree that contains the values of F at its leaves and such that the modulus of F at each point of the Baire space is given by the length of the corresponding branch in the tree. In this paper we provide the first internalization of this "inductive" continuity principle within a computational setting. Concretely, we present a class of intuitionistic theories that validate this formulation of continuity thanks to computations that construct such dialogue trees internally to the theories using effectful computations. We further demonstrate that this inductive continuity principle implies other forms of continuity principles.

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

Keywords and phrases Continuity, Dialogue trees, Stateful computations, Intuitionistic Logic, Extensional Type Theory, Constructive Type Theory, Realizability, Theorem proving, Agda

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.35

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

Acknowledgements We would like to thank Martin Escardo, Martin Baillon, and Yannick Forster for useful discussions about continuity and dialogue trees.

1 Introduction

The continuity principle is a cornerstone in intuitionistic theories which is generally accepted by constructivists but contradicts classical mathematics. In essence, the principle states that functions on the Baire space (i.e., $\mathfrak{B} \equiv \mathsf{Nat} \to \mathsf{Nat}$) only need finite inputs, i.e., initial segments of points of the Baire space, to produce outputs. Different variants of the continuity principle have been developed to capture different levels of strictness in the notion of continuity and different computational aspects. Perhaps the most common continuity principle is the continuity principle for numbers, sometimes referred to as the weak continuity principle (WCP) [24; 15; 4; 7; 38]. WCP states that given a function $F \in \mathfrak{B} \to \mathsf{Nat}$ and an point α of the Baire space $\mathfrak{B}$, $F(\alpha)$ 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 $\mathfrak{B}_n \equiv \{x : \mathsf{Nat} \mid x < n\} \to \mathsf{Nat}$ is the set of finite sequences of length n:

$$\mathtt{WCP} \equiv \mathbf{\Pi} F{:}\mathfrak{B} \to \mathsf{Nat}.\mathbf{\Pi}\alpha{:}\mathfrak{B}.\|\mathbf{\Sigma} n{:}\mathsf{Nat}.\mathbf{\Pi}\beta{:}\mathfrak{B}.(\alpha{=}\beta{\in}\mathfrak{B}_n) \to (F(\alpha){=}F(\beta){\in}\mathsf{Nat})\|$$

However, as shown, e.g., by Kreisel [25, p.154], Troelstra [36, Thm.IIA], and Escardó and Xu [18; 40], continuity is not an extensional property in the sense that two (extensionally) equal functions might have different moduli of continuity. Therefore, to computationally realize continuity, the existence of a modulus of continuity has to be truncated as explained, e.g., in [18; 40; 32; 33], which is what the $\|\_\|$ operator achieves in `WCP`'s definition above.

Brouwer used WCP, along with a consequence of Bar Induction called the Fan Theorem, to derive the following uniform continuity principle (UCP) [7, p.113], which he then used to prove that all real-valued function on the unit interval are uniformly continuous [24; 15; 4; 7; 38], where $\mathfrak{C} \equiv \mathsf{Nat} \to \mathsf{Bool}$ is the Cantor space and $\mathfrak{C}_n \equiv \{x : \mathsf{Nat} \mid x < n\} \to \mathsf{Bool}$:

$$\texttt{UCP} \equiv \mathbf{\Pi} F{:}\mathfrak{C} \to \mathsf{Nat}.\mathbf{\Sigma} n{:}\mathsf{Nat}.\mathbf{\Pi}\alpha, \beta{:}\mathfrak{C}.(\alpha{=}\beta{\in}\mathfrak{C}_n) \to (F(\alpha){=}F(\beta){\in}\mathsf{Nat})$$

Note that `UCP` does not need to be truncated as shown for example in [18].

Another version of the continuity principle, which originates from the completeness of Brouwer's bar thesis and implies both `WCP` and `UCP`, has been recently studied [20; 22; 21; 19]. This principle, referred to here as the Inductive Continuity Principle (ICP), relies on a notion of dialogue trees related to Brouwer trees [38] and reminiscent of Kleene trees [23]. This tree-based technique of capturing continuity information, pioneered in [20; 22; 21; 19], and reused for example in [9; 35; 3], consists in computing a tree that, given a function F from a subset of $\mathfrak{B}$ to numbers, contains the values of F at its leaves, and such that the amount of information needed to compute these values, i.e., the modulus of continuity of F at each point, is given by its branches. This can be formalized as follows where $\mathfrak{B}_{\mathsf{SNat}} \equiv \mathsf{Nat} \to \mathsf{SNat}$ for SNat a subtype of Nat (Bt and $\texttt{follow}(d, \alpha)$ are made formal in Sec. 3.2).[1]

$$\texttt{ICP} \equiv \mathbf{\Pi} F{:}\mathfrak{B}_{\mathsf{SNat}} \to \mathsf{Nat}.\|\mathbf{\Sigma} d{:}\mathsf{Bt}.\mathbf{\Pi}\alpha{:}\mathfrak{B}_{\mathsf{SNat}}.\texttt{follow}(d, \alpha){=}F(\alpha){\in}\mathsf{Nat}\|$$

A number of theories have been shown to satisfy Brouwer's continuity principle, or uniform variants, such as N-HA$^\omega$ by Troelstra [37, p.158], MLTT by Coquand and Jaber [13; 12], System T by Escardó [19], MLTT by Xu [40], CTT by Rahli and Bickford [32], BTT by Baillon, Mahboubi and Pedrot [3], among others (see Sec. 6 for details). These proofs often rely on a semantic forcing-based approach [13; 12], 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 [41]) or exhibit continuous behavior (e.g., dialogue trees in [19; 3]).

Not only can functions on the Baire space be proved to be continuous, but using effectful computations one can in fact *compute* their modulus of continuity [28]. The $\mathrm{TT}_{\mathcal{C}}^{\square}$ family of effectful extensional type theories, recalled in Sec. 2, was shown to be consistent with a version of `WCP` using a family of realizability models that allow validating this principle using effectful computations, and in particular using reference cells [11]. Building on this result, in this paper we identify a family of effectful type theories that are consistent with a variant of `ICP`, and prove this consistency result using effectful computations, namely references.

Importantly, in addition to validating the continuity of $\mathrm{TT}_{\mathcal{C}}^{\square}$ functions using dialogue trees, our work provides the first internalization of the principle into a computational system in the sense that we extend $\mathrm{TT}_{\mathcal{C}}^{\square}$ with a variant of `ICP` in Sec. 3, and exhibit in Sec. 5 an effectful $\mathrm{TT}_{\mathcal{C}}^{\square}$ program that realizes this axiom. The most challenging aspect of internalizing this dialogue-based technique is in proving termination of the computation of such trees. We further show in Sec. 4 that `ICP` encompasses both weak and uniform continuity. It is however still unknown whether `ICP` is in fact strictly stronger than the other principles.

---

1 We use here Brouwer trees, which are equivalent to dialogue trees for functions on the Baire space [17].

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

$v \in$ Value ::= vt	(type)	$\mid \lambda x.t$	(lambda)	$\mid \star$	(constant)
$\mid \underline{n}$	(number)	$\mid \mathtt{inl}(t)$	(left injection)	$\mid \boxed{\delta}$	(choice name)
$\mid \langle t_1, t_2 \rangle$	(pair)	$\mid \mathtt{inr}(t)$	(right injection)		
$vt \in$ Type ::= $\mathbf{\Pi} x{:}t_1.t_2$	(product)	$\mid \{x : t_1 \mid t_2\}$	(set)	$\mid t_1 + t_2$	(disjoint union)
$\mid \mathbf{\Sigma} x{:}t_1.t_2$	(sum)	$\mid t_1 = t_2 \in t$	(equality)	$\mid \lVert t \rVert$	(truncation)
$\mid \mathbb{U}_i$	(universe)	$\mid$ Nat	(numbers)	$\mid \boxed{\mathtt{pure}}$	(pure)
$\mid t_1 \cap t_2$	(intersection)				
$t \in$ Term ::= x	(variable)	$\mid \boxed{!t}$	(read)	$\mid t_1 <?\ t_2$	(less than)
$\mid v$	(value)	$\mid \boxed{\boldsymbol{\nu} x.t}$	(fresh)	$\mid t_1 =?\ t_2$	(equality)
$\mid t_1\ t_2$	(application)	$\mid \boxed{t_1 := t_2}$	(write)	$\mid \mathtt{let}\ x = t_1\ \mathtt{in}\ t_2$	(call-by-value)
$\mid \mathtt{fix}(t)$	(fixpoint)	$\mid t_1 + t_2$	(addition)	$\mid \mathtt{let}\ x, y = t_1\ \mathtt{in}\ t_2$	(pair destructor)
$\mid \mathtt{case}\ t\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2$	(injection destructor)				

$$
\begin{array}{lll}
(\lambda x.t)\ u & {}_w{\mapsto}_w & t[x\backslash u] \\
\mathtt{fix}(v) & {}_w{\mapsto}_w & v\ \mathtt{fix}(v) \\
\mathtt{let}\ x = v\ \mathtt{in}\ t_2 & {}_w{\mapsto}_w & t_2[x\backslash v] \\
\mathtt{let}\ x, y = \langle t_1, t_2 \rangle\ \mathtt{in}\ t & {}_w{\mapsto}_w & t[x\backslash t_1; y\backslash t_2]
\end{array}
\qquad
\begin{array}{lll}
\underline{n} <?\ \underline{m} & {}_w{\mapsto}_w & \mathtt{inl}(\star), \text{if } n < m \\
\underline{n} <?\ \underline{m} & {}_w{\mapsto}_w & \mathtt{inr}(\star), \text{if } n \not< m \\
\underline{n} =?\ \underline{m} & {}_w{\mapsto}_w & \mathtt{inl}(\star), \text{if } n = m \\
\underline{n} =?\ \underline{m} & {}_w{\mapsto}_w & \mathtt{inr}(\star), \text{if } n \neq m \\
\underline{n} + \underline{m} & {}_w{\mapsto}_w & \underline{n + m}
\end{array}
$$

$$
\begin{array}{lll}
!\delta & {}_w{\mapsto}_w & \mathsf{read}(w, \delta) \\
\delta := t & {}_w{\mapsto}_{\mathsf{write}(w,\delta,t)} & \star \\
\boldsymbol{\nu} x.t & {}_w{\mapsto}_{\mathsf{start}\nu\mathcal{C}(w)} & t[x\backslash \nu\mathcal{C}(w)]
\end{array}
\qquad
\begin{array}{lll}
\mathtt{case}\ \mathtt{inl}(t)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 & {}_w{\mapsto}_w & t_1[x\backslash t] \\
\mathtt{case}\ \mathtt{inr}(t)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 & {}_w{\mapsto}_w & t_2[y\backslash t]
\end{array}
$$

2 Background

This section reviews $\mathrm{TT}_{\mathcal{C}}^{\Box}$ [10] — a family of extensional type theories parameterized by a choice operator $\mathcal{C}$ and a metatheoretical modality $\Box$, which allows typing the choice operators.

2.1 Metatheory

Our metatheory is Agda's type theory [2]. The results presented in this paper have been formalized in Agda: https://github.com/vrahli/opentt/. We use $\forall, \exists, \wedge, \vee, \rightarrow, \neg$ in place of Agda's logical connectives in this paper, and use $\top$ for True and $\bot$ for False. Agda provides a 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 $\mathrm{TT}_{\mathcal{C}}^{\Box}$'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. 2.3, where we use function extensionality to interpret universes. We also use classical reasoning twice in the proof presented in Sec. 5.

2.2 $\mathrm{TT}_{\mathcal{C}}^{\Box}$'s Syntax and Operational Semantics

Fig. 1 recalls $\mathrm{TT}_{\mathcal{C}}^{\Box}$'s syntax and operational semantics, where the blue boxes highlight the effecful components, and where x belongs to a set of variables Var. For simplicity, numbers are considered to be primitive and the constant $\star$ is used in place of a term when the particular term used is irrelevant. We use all letters as metavariables for terms and denote by $t[x\backslash u]$ the capture-avoiding substitution of all the free occurrences of x in t by u. We write $\mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3$ for $\mathtt{case}\ t_1\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_2 \mid \mathtt{inr}(x) \Rightarrow t_3$, where x does not occur in t_2 or t_3, and $t_1;t_2$ for $\mathtt{let}\ x = t_1\ \mathtt{in}\ t_2$ where x does not occur free in t_2.

Types are syntactic forms that are given semantics in Sec. 2.3 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 $\mathbf{\Sigma} x{:}t_1.t_2$. We write $t_1 \rightarrow t_2$ for the non-dependent $\mathbf{\Pi}$ type; Unit for $\underline{0}{=}\underline{0}{\in}$Nat; Void for $\underline{0}{=}\underline{1}{\in}$Nat; $\neg T$ for $(T \rightarrow$ Void); and Bool for Unit+Unit.

To capture the time progression notion which underlines choice operators, $\mathrm{TT}_{\mathcal{C}}^{\square}$ is parameterized by a Kripke frame [27; 26], consisting 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 $\forall w' \sqsupseteq w.\mathbb{P}$. Due to $\sqsubseteq$'s transitivity, if $P \in \mathcal{P}_w$ then for every $w' \sqsupseteq w$ it naturally extends to a predicate in $\mathcal{P}_{w'}$. Let $\forall_w^{\sqsubseteq}(P)$ stand for the fact that $P \in \mathcal{P}_w$ is true for all extensions of w, i.e., $P\ w'$ holds for all $w' \sqsupseteq w$. We sometime write $\forall_w^{\sqsubseteq}(w'.P)$ instead of $\forall_w^{\sqsubseteq}(\lambda w'.P)$.

Fig. 1's lower part presents $\mathrm{TT}_{\mathcal{C}}^{\square}$'s small-step call-by-name operational semantics, where $t_1\ {}_{w_1}{\mapsto}_{w_2}\ t_2$ expresses that t_1 reduces to t_2 in one step of computation from the world w_1 and potentially updating it so that the resulting world is w_2. We omit the congruence rules such as: if $t_1\ {}_{w_1}{\mapsto}_{w_2}\ t_2$ then $t_1(u)\ {}_{w_1}{\mapsto}_{w_2}\ t_2(u)$. We denote by $\mapsto^*$ the reflexive transitive closure of $\mapsto$, i.e., $a\ {}_{w_1}{\mapsto}_{w_2}^*\ b$ states that a computes to b in 0 or more steps. We write $a \mapsto_w^* b$ for $\exists(w' : \mathcal{W}).a\ {}_{w}{\mapsto}_{w'}^*\ b$, and $a \Mapsto_w b$ for $\forall_w^{\sqsubseteq}(w'.a \mapsto_{w'}^* b)$.

$\mathrm{TT}_{\mathcal{C}}^{\square}$ includes effecful notions that rely on worlds to record choices and provides operators to access and update choices. In this paper, for conciseness of presentation, we focus on one instance of choice operators as mutable references to natural numbers. Reference cells, which allow a program to indirectly access a particular object, are choice operators since they can point to different objects over their lifetime. See [10] for the general notion of choice operators. To define references to numbers[2], we let the set of choices $\mathcal{C} \subseteq \mathsf{Term}$ to be $\mathbb{N}$. A choice stored in a reference cell is referred to through the reference's name. To this end, $\mathrm{TT}_{\mathcal{C}}^{\square}$'s computation system is parameterized by a set $\mathcal{N}$ of choice names, ranged over by δ, equipped with a decidable equality, and an operator that given a list of names, returns a name not in the list ($\mathcal{N} \coloneqq \mathbb{N}$ for simplicity). This can be given by nominal sets [30]. We take worlds to be lists of cells, where a cell is a pair of a choice name and a choice, and $\sqsubseteq$ is the reflexive transitive closure of two operations that allow creating and updating reference cells.

As shown in Fig. 1, a choice name δ can be used in a computation to access choices from a world using $!\delta\ {}_{w}{\mapsto}_{w}\ \mathsf{read}(w, \delta)$, where the partial function $\mathsf{read} \in \mathcal{W} \to \mathcal{N} \to \mathcal{C}$ accesses the content of the δ-cell in w if that cell exists.[3] Choices can be made using $(\delta \coloneqq t)\ {}_{w}{\mapsto}_{\mathsf{write}(w,\delta,t)}\ \star$, where $\mathsf{write}(w, \delta, t)$ updates the reference δ with the choice t if δ occurs in w, and otherwise returns w, and therefore $w \sqsubseteq \mathsf{write}(w, \delta, t)$. The computation returns $\star$, which is reminiscent of reference updates in OCaml for example, which are of type `unit`. Finally, new choice names can be generated using $\boldsymbol{\nu}x.t\ {}_{w}{\mapsto}_{\mathsf{start}\nu\mathcal{C}(w)}\ t[x \backslash \nu\mathcal{C}(w)]$, where $\nu\mathcal{C}(w)$ returns a "fresh" name not occurring in the list w, which x gets replaced with in the expression above, and $\mathsf{start}\nu\mathcal{C}(w)$ returns the list w extended with the pair $\langle \nu\mathcal{C}(w), 0\rangle$, where 0 is the default value with which reference cells are filled, and therefore $\forall(w : \mathcal{W}).w \sqsubseteq \mathsf{start}\nu\mathcal{C}(w)$.[4]

2.3 Forcing Interpretation

$\mathrm{TT}_{\mathcal{C}}^{\square}$'s semantics is similar to the one presented in [10], which we recall and extend in Fig. 2. Types are interpreted via a forcing interpretation defined using induction-recursion [16] as follows, where the forcing conditions are worlds: (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 also define $a \Mapsto_{!w} b$ as $\forall_w^{\sqsubseteq}(w'.a\ {}_{w'}{\mapsto}_{w'}^*\ b)$, capturing the fact that the computation

[2] Only relevant components of the choice operator are discussed. See worldInstanceRef.lagda for details.
[3] In general, read, $\nu\mathcal{C}$, $\mathsf{start}\nu\mathcal{C}$, and write are all parameters of $\mathrm{TT}_{\mathcal{C}}^{\square}$, as described in [10]. Here they too are instantiated with references to numbers.
[4] $\mathrm{TT}_{\mathcal{C}}^{\square}$ also contains a quotienting type operator ≀ used to assign types to computations that can compute to different values in different worlds, such as choices $!\delta$ [11]. For readability, we elide it here.

Figure 2 Forcing Interpretation

Numbers:
- $w \vDash \mathsf{Nat}{\equiv}\mathsf{Nat} \iff \texttt{True}$
- $w \vDash t{\equiv}t'{\in}\mathsf{Nat} \iff \square_w(w'.\exists(n:\mathbb{N}).t \Mapsto_{w'} \underline{n} \land t' \Mapsto_{w'} \underline{n})$

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

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

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

Disjoint unions:
- $w \vDash A_1{+}B_1{\equiv}A_2{+}B_2 \iff w \vDash A_1{\equiv}A_2 \land w \vDash B_1{\equiv}B_2$
- $w \vDash a_1{\equiv}a_2{\in}A{+}B \iff \square_w(w'.\exists(u, v : \mathsf{Term}).(a_1 \Mapsto_{w'} \texttt{inl}(u) \land a_2 \Mapsto_{w'} \texttt{inl}(v) \land w' \vDash u{\equiv}v{\in}A) \lor (a_1 \Mapsto_{w'} \texttt{inr}(u) \land a_2 \Mapsto_{w'} \texttt{inr}(v) \land w' \vDash u{\equiv}v{\in}B))$

Equalities:
- $w \vDash (a_1{=}b_1{\in}A){\equiv}(a_2{=}b_2{\in}B) \iff w \vDash A{\equiv}B \land w \vDash a_1{\equiv}a_2{\in}A \land w \vDash b_1{\equiv}b_2{\in}B$
- $w \vDash a_1{\equiv}a_2{\in}(a{=}b{\in}A) \iff \square_w(w'.w' \vDash a{\equiv}b{\in}A)$

Subsingletons:
- $w \vDash \|A\|{\equiv}\|B\| \iff w \vDash A{\equiv}B$
- $w \vDash a{\equiv}b{\in}\|A\| \iff \square_w(w'.w' \vDash a{\equiv}a{\in}A \land w' \vDash b{\equiv}b{\in}A)$

Purity:
- $w \vDash \texttt{pure}{\equiv}\texttt{pure} \iff \top$
- $w \vDash a_1{\equiv}a_2{\in}\texttt{pure} \iff \texttt{namefree}(a_1) \land \texttt{namefree}(a_2)$

Binary intersections:
- $w \vDash A_1 \cap B_1{\equiv}A_2 \cap B_2 \iff w \vDash A_1{\equiv}A_2 \land w \vDash B_1{\equiv}B_2$
- $w \vDash a_1{\equiv}a_2{\in}A \cap B \iff \square_w(w'.w' \vDash a_1{\equiv}a_2{\in}A \land w' \vDash a_1{\equiv}a_2{\in}B)$

Modality closure:
- $w \vDash T_1{\equiv}T_2 \iff \square_w(w'.\exists(T_1', T_2' : \mathsf{Term}).T_1 \Mapsto_{w'} T_1' \land T_2 \Mapsto_{w'} T_2' \land w' \vDash T_1'{\equiv}T_2')$
- $w \vDash t_1{\equiv}t_2{\in}T \iff \square_w(w'.\exists(T' : \mathsf{Term}).T \Mapsto_{w'} T' \land w' \vDash t_1{\equiv}t_2{\in}T')$

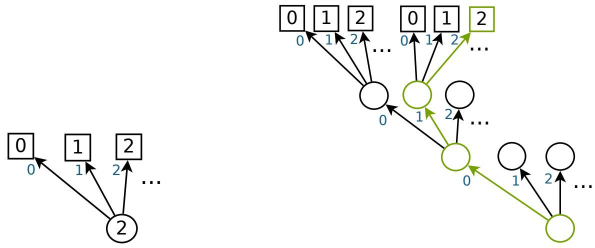
can read using $!\delta$ but not write, and therefore does not change the initial world (this is used in Thm. 1). Fig. 2 defines in particular the semantics of `pure`, which is inhabited by name-free terms, where $\texttt{namefree}(t)$ is defined recursively over t and returns false iff t contains a choice name δ or a fresh operator of the form $\boldsymbol{\nu}x.t$. We also write $\mathsf{Fam}_w(A_1, A_2, B_1, B_2)$ for $w \vDash A_1{\equiv}A_2 \land \forall_w^{\sqsubseteq}(w'.\forall(a_1, a_2 : \mathsf{Term}).w' \vDash a_1{\equiv}a_2{\in}A_1 \to w' \vDash B_1(a_1){\equiv}B_2(a_2))$. 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 $\square_w \lambda w'.P$. To guarantee that this interpretation yields a type system in the sense of Thm. 1, we require that the modalities satisfy certain properties detailed in [10] and reminiscent of standard modal axiom schemata [14].

▶ **Theorem 1** ([10]). *$TT_{\mathcal{C}}^{\square}$ is a standard type system in the sense that its forcing interpretation induced by $\square$ satisfies the following properties (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:	$w \vDash T{\equiv}T \to T \Mapsto_{!w} T' \to w \vDash T{\equiv}T'$	$w \vDash t{\equiv}t{\in}T \to t \Mapsto_{!w} t' \to w \vDash t{\equiv}t'{\in}T$
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:	$\neg w \vDash t{\equiv}t{\in}\mathsf{Void}$	

Note that due to effects, types are not closed under all computations. For example, when $T \equiv \mathsf{Nat}$, $t' \Mapsto_w \underline{n}$ does not necessarily follow from $t \Mapsto_w t'$ and $t \Mapsto_w \underline{n}$. An example is $t \equiv (\delta \coloneqq \underline{1};\texttt{if}\ !\delta\ <\ \underline{1}\ \texttt{then}\ \underline{0}\ \texttt{else}\ \underline{1})$, which reduces to $t' \equiv (\texttt{if}\ !\delta\ <\ \underline{1}\ \texttt{then}\ \underline{0}\ \texttt{else}\ \underline{1})$ 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 $\Mapsto_w$:

Figure 3 Examples of dialogue (left) and Brouwer (right) trees for $\lambda\alpha.\alpha(\underline{2})$



$t' \mapsto_w t \to w \vDash t{\equiv}t{\in}\mathsf{Nat} \to w \vDash t{\equiv}t'{\in}\mathsf{Nat}$. Similarly, the following also holds by transitivity of $\mapsto_w$: $w \vDash T{\equiv}T \to T' \mapsto_w T \to w \vDash T{\equiv}T'$. Finally, note that, as indicated in Thm. 1, this semantics is closed under β-reduction, as β-reduction does not modify the current world.

2.4 $\text{TT}_{\mathcal{C}}^{\square}$'s Inference Rules

$\text{TT}_{\mathcal{C}}^{\square}$'s inference rules are standard and they reflect the semantics of the types, which is given meaning through a forcing interpretation presented in Sec. 2.3. Concetely, sequents in $\text{TT}_{\mathcal{C}}^{\square}$ are of the form $h_1, \ldots, h_n \vdash t : T$. Such a sequent denotes that, assuming $h_1, \ldots, h_n$, T is a type inhabited by t. An hypothesis h is of the form $x{:}A$, where the variable x stands for the name of the hypothesis and A its type. We write $a{\in}A$ for $a{=}a{\in}A$. To illustrate the naturality of the typing rules and their correspondence to the forcing interpretation, we provide examples of $\text{TT}_{\mathcal{C}}^{\square}$'s inference rules for $\mathbf{\Pi}$ types. The following rules are the standard $\mathbf{\Pi}$-elimination, $\mathbf{\Pi}$-introduction, type equality for $\mathbf{\Pi}$ types, and λ-introduction rules, respectively.

$$\frac{H, f{:}\mathbf{\Pi}x{:}A.B, J \vdash a{\in}A \quad H, f{:}\mathbf{\Pi}x{:}A.B, J, z{:}f(a){\in}B[x\backslash a] \vdash e : C}{H, f{:}\mathbf{\Pi}x{:}A.B, J \vdash e[z\backslash\star] : C} \qquad \frac{H, z{:}A \vdash b : B[x\backslash z] \quad H \vdash A{\in}\mathbb{U}_i}{H \vdash \lambda z.b : \mathbf{\Pi}x{:}A.B}$$

$$\frac{H \vdash A_1{=}A_2{\in}\mathbb{U}_i \quad H, y{:}A_1 \vdash B_1[x_1\backslash y]{=}B_2[x_2\backslash y]{\in}\mathbb{U}_i}{H \vdash \mathbf{\Pi}x_1{:}A_1.B_1{=}\mathbf{\Pi}x_2{:}A_2.B_2{\in}\mathbb{U}_i} \qquad \frac{H, z{:}A \vdash t_1[x_1\backslash z]{=}t_2[x_2\backslash z]{\in}B[x\backslash z] \quad H \vdash A{\in}\mathbb{U}_i}{H \vdash \lambda x_1.t_1{=}\lambda x_2.t_2{\in}\mathbf{\Pi}x{:}A.B}$$

The following rules are the standard function extensionality and β-reduction rules, resp.:

$$\frac{H, z{:}A \vdash f_1(z){=}f_2(z){\in}B[x\backslash z] \quad H \vdash A{\in}\mathbb{U}_i}{H \vdash f_1{=}f_2{\in}\mathbf{\Pi}x{:}A.B} \qquad \frac{H \vdash t[x\backslash s]{=}u{\in}T}{H \vdash (\lambda x.t)\ s{=}u{\in}T}$$

3 Inductive Continuity via Brouwer Trees

This section states a dialogue tree-based continuity principle, referred to as the inductive continuity principle, since it relies on trees to capture functions. As we show in Sec. 4, it implies both Brouwer's continuity principle for numbers and his uniform continuity principle on the Cantor space. Furthermore, it is still unknown whether the inductive continuity principle is strictly stronger than Brouwer's continuity principle for numbers. Sec. 5 internally validates this inductive principle. In particular, Thm. 4 shows that, given a pure function $F \in \mathfrak{B} \to \mathsf{Nat}$, $\text{TT}_{\mathcal{C}}^{\square}$ provides a computation, introduced in Sec. 5.1, that builds a dialogue tree capturing F's continuity.

As mentioned above, we rely here on Brouwer trees, which are a simple form of dialogue trees. Let us provide an example of how dialogue and Brouwer trees work. Consider the function $F \equiv \lambda\alpha.\alpha(\underline{2}) \in \mathfrak{B} \to \mathsf{Nat}$. Fig. 3 (left) shows its dialogue tree, where the internal (root) node is labeled with the value α is applied to, and the leaves contain the values of F for all possible inputs. For example if F is applied to $\alpha \equiv \lambda x.x$, then starting from the root, we apply α to the node's value, i.e., $\underline{2}$, which gives us $\underline{2}$, and we therefore follow the

2nd path, which leads to the leaf labeled 2, the value of $F(\alpha)$. If $\alpha \equiv \lambda x.\underline{0}$, then $\alpha(\underline{2})$ is now $\underline{0}$, and following the 0th path leads to the leaf labeled 0, which is the value of $F(\alpha)$. Fig. 3 (right) shows F's Brouwer tree, where as opposed to dialogue trees, internal nodes are not labeled, and as for dialogue trees, the leaves contain the values of F for all inputs. For example if F is applied to $\alpha \equiv \lambda x.x$, because $\alpha(\underline{0})$ is $\underline{0}$, we first follow the 0the branch; then because $\alpha(\underline{1})$ is $\underline{1}$, we follow the 1st branch, and finally because $\alpha(\underline{2})$ is $\underline{2}$, we follow the 2nd branch, leading to a leaf labeled 2 (following the green path in Fig. 3). If $\alpha \equiv \lambda x.\underline{0}$, then we instead always follow the 0th branch, leading to a leaf labeled with 0.

In the dialogue tree, the modulus of continuity of F at some point α is given by the maximum value of the internal nodes followed using α, while in the Brouwer tree, the modulus is the length of the branch followed using α. Note that, in general, the values of the internal nodes of a dialogue tree of a function $F \in \mathfrak{B} \to \mathsf{Nat}$ are used to “ask questions” to an argument $\alpha \in \mathfrak{B}$ to decide what branch to take in the tree (by applying α to those values), while in a Brouwer tree, “dialogues” happen by asking all the values of an initial segment of α.

3.1 Extending $\mathrm{TT}_{\mathcal{C}}^{\Box}$ with (Co-)W Types and Infinite Sequences

In order to state the inductive continuity principle, we make use of the notion of a Brouwer tree, which we define in $\mathrm{TT}_{\mathcal{C}}^{\Box}$ using W types [1, Sec.5.2], which is a standard way of representing inductive types. Additionally, we use co-W types (also called M types) [1, Sec.5.2], the dual notion to that of a W type, to prove the validity of the principle. Thus, we add W and M types to $\mathrm{TT}_{\mathcal{C}}^{\Box}$, using `sup` as a W type and M type constructor and `wrec` as a W type recursor.

$$\begin{aligned} vt \in \mathsf{Type} &::= \cdots \mid W(t_1, t_2) \mid M(t_1, t_2) \\ t \in \mathsf{Term} &::= \cdots \mid \mathsf{sup}(t_1, t_2) \mid \mathtt{wrec}(t_1, t_2) \\ v \in \mathsf{Value} &::= \cdots \mid \lceil \boldsymbol{s} \rfloor, \text{ where } \boldsymbol{s} \text{ is a metatheoretical function in } \mathbb{N} \to \mathbb{N} \end{aligned}$$

where $\mathtt{wrec}(t_1, t_2)$ and $\lceil \boldsymbol{s} \rfloor$ compute as follows:

$$\mathtt{wrec}(\mathsf{sup}(a, f), g) \ {}_{w}{\mapsto}_{w} \ g\ a\ f\ (\lambda b.\mathtt{wrec}(f(b), g)) \qquad\qquad \lceil \boldsymbol{s} \rfloor\ \underline{n} \ {}_{w}{\mapsto}_{w} \ \underline{\boldsymbol{s}(n)}$$

In addition, the application operator is modified so that it evaluates its argument whenever the function is of the form $\lceil \boldsymbol{s} \rfloor$, i.e., $\lceil \boldsymbol{s} \rfloor\ a$ reduces to $\lceil \boldsymbol{s} \rfloor\ b$ when a reduces to b. Hence, for any metatheoretical function $\boldsymbol{s}$ in $\mathbb{N} \to \mathbb{N}$, $\lceil \boldsymbol{s} \rfloor$ inhabits $\mathfrak{B}$. These sequences are used in Sec. 5.5 to prove that the computation of Brouwer trees provided in Sec. 5.1 terminates. They are similar to the sequences of the form $\lambda\!\!\lambda x.M_x$ in [5], where the infinite sequence of terms $M_1, M_2, \ldots$ does not have a computational purpose, but is used to prove termination in their proof that some bar recursion operator realizes the negative translation of the axiom of choice. Similar sequences have been used in [31] to validate versions of the axiom of choice, and in [34] to validate variants of Brouwer's Bar Induction principle [24].

W and M types are interpreted in a standard way:

W types:
- $w \vDash W(A_1, B_1) \equiv W(A_2, B_2) \iff \mathsf{Fam}_w(A_1, A_2, B_1, B_2)$
- $w \vDash s_1 \equiv s_2 \in W(A, B) \iff \Box_w(w'.\mu(R.\exists(a_1, a_2, f_1, f_2 : \mathsf{Term}).w' \vDash a_1 \equiv a_2 \in A \wedge (\forall(b_1, b_2 : \mathsf{Term}).w' \vDash b_1 \equiv b_2 \in B(a_1) \to R\ f_1(b_1)\ f_2(b_2)) \wedge s_1 \Rightarrow_{w'} \mathsf{sup}(a_1, f_1) \wedge s_2 \Rightarrow_{w'} \mathsf{sup}(a_2, f_2))\ s_1\ s_2)$

M types:
- $w \vDash M(A_1, B_1) \equiv M(A_2, B_2) \iff \mathsf{Fam}_w(A_1, A_2, B_1, B_2)$
- $w \vDash s_1 \equiv s_2 \in M(A, B) \iff \Box_w(w'.\nu(R.\exists(a_1, a_2, f_1, f_2 : \mathsf{Term}).w' \vDash a_1 \equiv a_2 \in A \wedge (\forall(b_1, b_2 : \mathsf{Term}).w' \vDash b_1 \equiv b_2 \in B(a_1) \to R\ f_1(b_1)\ f_2(b_2)) \wedge s_1 \Rightarrow_{w'} \mathsf{sup}(a_1, f_1) \wedge s_2 \Rightarrow_{w'} \mathsf{sup}(a_2, f_2))\ s_1\ s_2)$

Therefore, $W(A,B)$ and $M(A,B)$ are types in $\mathbb{U}_i$ whenever $A \in \mathbb{U}_i$ and $B \in A \to \mathbb{U}_i$. Given $a \in A$ and $f \in B[a] \to W(A,B)$, $\mathsf{sup}(a,f) \in W(A,B)$ is a W type constructor, and if $f \in B[a] \to M(A,B)$ then $\mathsf{sup}(a,f) \in M(A,B)$ is an M type constructor. Given $t \in W(A,B)$ and $g \in \mathbf{\Pi} a{:}A.(B(a) \to W(A,B)) \to (B(a) \to C) \to C$, $\mathtt{wrec}(t,g) \in C$ is a W type recursor.

▶ **Example 2.** Given $A \in \mathbb{U}_i$ and $B \in A \to \mathbb{U}_i$, $W(A,B)$ denotes the type of inductive definitions with inhabitants of A representing the constructors (as well as their non-inductive parameters), and $B(a)$ representing the indices of inductive parameters at a given constructor a. For example, the natural numbers have two constructors: zero and succ, the latter having one inductive parameter. Therefore, natural numbers are encoded as:

$$W(\mathsf{Bool}, \lambda x.\mathtt{case}\ x\ \mathtt{of}\ \mathtt{inl}(\_) \Rightarrow \mathsf{Void} \mid \mathtt{inr}(\_) \Rightarrow \mathsf{Unit}),$$

where Void captures the lack of inductive parameters for zero and Unit captures succ's single inductive parameter. The constructors zero and succ are then be encoded as:

$$\mathsf{zero} \equiv \mathsf{sup}(\mathtt{inl}(\star), \lambda x.\star) \quad \text{and} \quad \mathsf{succ} \equiv \lambda n.\mathsf{sup}(\mathtt{inr}(\star), \lambda x.n)$$

3.2 Brouwer Tree-Based Inductive Continuity Principle

We can now state the inductive continuity principle that captures the moduli of continuity of functions in $\mathfrak{B}_{\mathsf{SNat}} \to \mathsf{Nat}$ using Brouwer trees, where $\mathfrak{B}_{\mathsf{SNat}} \equiv \mathsf{Nat} \to \mathsf{SNat}$ for SNat a subtype of Nat (this principle is therefore a family of principles for all such SNats). This continuity result, as well as the ones recalled in Sec. 4, are stated for pure functions only using the following quantification: $\mathbf{\Pi_p} a{:}A.B \equiv \mathbf{\Pi} a{:}(A \cap \mathtt{pure}).B$, which quantifies over pure members of A. We also write A_{p} for $A \cap \mathtt{pure}$ and $A +_{\mathsf{p}} B$ for $(A{+}B) \cap \mathtt{pure}$. It remains to be determined whether some effectful computations can be proved to be continuous.

We first define Brouwer trees (a class of dialogue trees where internal nodes are not labeled) using W types as follows.

▶ **Definition 3** (Brouwer Trees). *A* Brouwer tree *is a member of* $\mathsf{Bt} \equiv W(\mathsf{BtA}, \mathsf{BtB})$, *where* $\mathsf{BtA} \equiv \mathsf{Nat} +_{\mathsf{p}} \mathtt{Unit}$ *and* $\mathsf{BtB} \equiv \lambda a.\mathtt{if}\ a\ \mathtt{then}\ \mathtt{Void}\ \mathtt{else}\ \mathsf{SNat}_{\mathsf{p}}$. *Such trees have two constructors:* $\eta(i) \equiv \mathsf{sup}(\mathtt{inl}(i), \lambda x.\star)$, *which builds a leaf node with value* $i \in \mathsf{Nat}$; *and* $\digamma(f) \equiv \mathsf{sup}(\mathtt{inr}(\star), f)$, *which builds an internal node from a function* $f \in \mathsf{SNat}_{\mathsf{p}} \to \mathsf{Bt}$.

Using this definition, the Brouwer tree depicted in Fig. 3 is $\digamma(\lambda i.\digamma(\lambda j.\digamma(\lambda k.\eta(k))))$.

▶ **Theorem 4** (Inductive Continuity Principle). *The following continuity principle, referred to as* $\mathsf{ICP_p}$, *is valid in* $TT_{\mathcal{C}}^{\square}$[5] *(see* `contDiagVal` *in* barContP10.lagda *for details):*

$$\mathbf{\Pi_p} F{:}\mathfrak{B}_{\mathsf{SNat}} \to \mathsf{Nat}.\|\mathbf{\Sigma} d{:}\mathsf{Bt}.\mathbf{\Pi_p}\alpha{:}\mathfrak{B}_{\mathsf{SNat}}.\mathtt{follow}(d,\alpha){=}F(\alpha){\in}\mathsf{Nat}\| \qquad (\mathsf{ICP_p})$$

where $\mathtt{follow}(d,\alpha)$ *extracts the value of the leaf encountered when following* α *in* d *as follows:*

$$\mathtt{follow}(d,\alpha) \equiv \mathtt{wrec}(d, \lambda a.\lambda f.\lambda r.\lambda k.\mathtt{case}\ a\ \mathtt{of}\ \mathtt{inl}(i) \Rightarrow i \mid \mathtt{inr}(\_) \Rightarrow r\ (\alpha\ k)\ (k+\underline{1}))\ \underline{0}$$

At a high-level, the proof goes as follows (the full proof is carried out in Sec. 5).

Step 1: Given a function in $\mathfrak{B}_{\mathsf{SNat}} \to \mathsf{Nat}$, we first build by coinduction a possibly infinite co-Brouwer tree as an M type. This co-Brouwer tree contains the result of F applied to the finite sequence s at the leaf ending the path following s whenever s contains enough information to compute the result of F.

[5] "Valid in $\mathrm{TT}_{\mathcal{C}}^{\square}$" here means that the principle is realizable in $\mathrm{TT}_{\mathcal{C}}^{\square}$, thus it is consistent with the theory.

Step 2: Classically, this co-Brouwer tree is either finite or contains an infinite branch.
Step 3: If the co-Brouwer tree is finite, it is a Brouwere tree.
Step 4: If the co-Brouwer tree contains an infinite branch, then the branch gives rise to an infinite sequence α, and since F is continuous, the path must be finite. As discussed in Sec. 5.5, this step relies on a continuity argument similar to the one used to validate the weak continuity principle $\mathsf{WCP_p}$ recalled in Sec. 4.1.
Step 5: Finally, the obtained Brouwer tree is shown to contain the values of F at its leaves.

4 Relation with Other Continuity Principles

This section demonstrates that inductive continuity implies both Brouwer's continuity principle for numbers (referred to as weak continuity here) and uniform continuity.

4.1 Weak Continuity

$\mathrm{TT}^{\square}_{\mathcal{C}}$ was shown to satisfy the following version of Brouwer's continuity principle for numbers, also called the weak continuity principle, which therefore can be added as an axiom [11].

$$\boldsymbol{\Pi_p} F{:}\mathfrak{B} \to \mathsf{Nat}.\boldsymbol{\Pi_p}\alpha{:}\mathfrak{B}.\|\boldsymbol{\Sigma} n{:}\mathsf{Nat}.\boldsymbol{\Pi_p}\beta{:}\mathfrak{B}.(\alpha{=}\beta{\in}\mathfrak{B}_n) \to (F(\alpha){=}F(\beta){\in}\mathsf{Nat})\| \qquad (\mathsf{WCP_p})$$

$\mathsf{WCP_p}$ is realized in every world by the term $\lambda F.\lambda\alpha.\langle \mathtt{mod}(F,\alpha), \lambda\beta.\lambda e.\star\rangle$, where $\mathtt{mod}(F,\alpha)$ computes the modulus of continuity of the function $F \in \mathfrak{B} \to \mathsf{Nat}$ at $\alpha \in \mathfrak{B}$. Roughly speaking, $\mathtt{mod}(F,\alpha)$ generates a reference cell δ initialized with 0, applies F to a modified version of α (namely $\mathtt{upd}(\delta,\alpha)$) that keeps track using δ of the highest number α gets applied to, and then returns the value held by δ (plus one). Formally:

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

Note that the truncation in $\mathsf{WCP_p}$ is necessary. It has been shown that a non-truncated version of $\mathtt{WCP}$ is inconsistent with MLTT [18; 40], and the same applies to $\mathsf{WCP_p}$ and $\mathrm{TT}^{\square}_{\mathcal{C}}$. The main reason for this is the semantics of dependent functions given by $\mathrm{TT}^{\square}_{\mathcal{C}}$'s realizability model (see Fig. 2). Under this semantics, $f \in \boldsymbol{\Pi} x{:}A.B$ if f maps equal terms $a_1{=}a_2{\in}A$ to equal terms $f(a_1){=}f(a_2){\in}B[x\backslash a_1]$. As continuity is a non-extensional property [25], extensionally equal functions in $\mathfrak{B}$ might have different moduli of continuity, so $\mathsf{WCP_p}$'s realizer cannot inhabit a non-truncated version of $\mathsf{WCP_p}$. However, when B is of the form $\|C\|$, it suffices that $f(a_1)$ and $f(a_2)$ are both members of $C[x\backslash a_1]$, allowing $\mathsf{WCP_p}$'s validation.

▶ **Theorem 5.** $\mathsf{WCP_p}$ *is derivable from* $\mathsf{ICP_p}$ *in* $\mathrm{TT}^{\square}_{\mathcal{C}}$ *when* $\mathsf{SNat} \equiv \mathsf{Nat}$.

Proof outline. Let $F \in \mathfrak{B} \to \mathsf{Nat}$ a pure function and let $\alpha \in \mathfrak{B}$. It follows from $\mathsf{ICP_p}$ that: $\|\boldsymbol{\Sigma} d{:}\mathsf{Bt}.\boldsymbol{\Pi_p}\alpha{:}\mathfrak{B}.\mathtt{follow}(d,\alpha){=}F(\alpha){\in}\mathsf{Nat}\|$. Because both principles are truncated, we can assume the existence of a tree $d \in \mathsf{Bt}$ such that: $\boldsymbol{\Pi_p}\alpha{:}\mathfrak{B}.\mathtt{follow}(d,\alpha){=}F(\alpha){\in}\mathsf{Nat}$. Because d encodes the modulus of continuity of each sequence $\alpha \in \mathfrak{B}$, as the length of the branch in d that "follows" α, we instantiate the conclusion with: $n \equiv \mathtt{lenBranch}(d,\alpha) \in \mathsf{Nat}$, where:

$$\mathtt{lenBranch}(d,\alpha) \equiv \mathtt{wrec}(d, \lambda a.\lambda f.\lambda r.\lambda k.\mathtt{case}\ a\ \mathtt{of}\ \mathtt{inl}(i) \Rightarrow k\ \mid\ \mathtt{inr}(\_) \Rightarrow r\ (\alpha\ k)\ (k{+}\underline{1}))\ \underline{0}$$

It now remains to prove that $F(\alpha){=}F(\beta){\in}\mathsf{Nat}$, for any pure function $\beta \in \mathfrak{B}$ such that $\alpha{=}\beta{\in}\mathfrak{B}_n$. From $\mathsf{ICP_p}$, we know that $\mathtt{follow}(d,\alpha){=}F(\alpha){\in}\mathsf{Nat}$ and $\mathtt{follow}(d,\beta){=}F(\beta){\in}\mathsf{Nat}$. Therefore, it is enough to prove $\mathtt{follow}(d,\alpha){=}\mathtt{follow}(d,\beta){\in}\mathsf{Nat}$, which follows from the following fact: $\boldsymbol{\Pi}\alpha,\beta{:}\mathfrak{B}.\alpha{=}\beta{\in}\mathfrak{B}_{\mathtt{lenBranch}(d,\alpha)} \to \mathtt{follow}(d,\alpha){=}\mathtt{follow}(d,\beta){\in}\mathsf{Nat}$. ◀

4.2 Uniform Continuity

The uniform continuity principle states that all functions on the Cantor space ($\mathfrak{C} \equiv \mathsf{Nat} \to \mathsf{Bool}$) are uniformly continuous, meaning that all points $\alpha \in \mathfrak{C}$ have the same modulus of continuity. We consider here the following version:

$$\mathbf{\Pi}_{\boldsymbol{p}} F{:}\mathfrak{C} \to \mathsf{Nat}.\|\mathbf{\Sigma} n{:}\mathsf{Nat}.\mathbf{\Pi}_{\boldsymbol{p}} \alpha, \beta{:}\mathfrak{C}.(\alpha{=}\beta{\in}\mathfrak{C}_n) \to (F(\alpha){=}F(\beta){\in}\mathsf{Nat})\| \qquad (\mathsf{UCP_p})$$

Brouwer proved that all real-valued functions on the unit interval are uniformly continuous [8, Thm.3] using `WCP` and the Fan Theorem [38, Ch.7, Sec.7; 15, Sec.3.2], which he derived from Bar Induction. While it was shown that in the case of uniform continuity the truncation can be removed [18; 40], we leave formalizing this in $\mathrm{TT}_{\mathcal{C}}^{\square}$ for future work.

▶ **Theorem 6.** $\mathsf{UCP_p}$ *is derivable from* $\mathsf{ICP_p}$ *in* $TT_{\mathcal{C}}^{\square}$ *when* $\mathsf{SNat} \equiv \{x : \mathsf{Nat} \mid x < 2\}$ *or equivalently* $\mathsf{SNat} \equiv \mathsf{Bool}$ *(and therefore* $\mathfrak{B}_{\mathsf{SNat}}$ *is* $\mathfrak{C}$*).*

Proof outline. Let $F \in \mathfrak{C} \to \mathsf{Nat}$ be a pure function. Because both principles are truncated, we can assume the existence of a tree $d \in \mathsf{Bt}$ such that: $\mathbf{\Pi}_{\boldsymbol{p}}\alpha{:}\mathfrak{C}.\mathtt{follow}(d, \alpha){=}F(\alpha){\in}\mathsf{Nat}$. As d is finitely branching and encodes the modulus of continuity of each $\alpha \in \mathfrak{C}$ as the length of the branch in d that "follows" α, we compute the uniform modulus of continuity of F as d's depth as follows, where $\mathtt{max}(i, j)$ returns the maximum among the numbers i and j:

$$\mathtt{depth}(d) \equiv \mathtt{wrec}(d, \lambda a.\lambda f.\lambda r.\mathtt{case}\ a\ \mathtt{of}\ \mathtt{inl}(i) \Rightarrow \underline{1} \mid \mathtt{inr}(\_) \Rightarrow \mathtt{max}(r(0), r(1)) + \underline{1})$$

We then instantiate our conclusion with $n \equiv \mathtt{depth}(d) \in \mathsf{Nat}$, and have to prove that $F(\alpha){=}F(\beta){\in}\mathsf{Nat}$, for all pure functions $\alpha, \beta \in \mathfrak{C}$ such that $\alpha{=}\beta{\in}\mathfrak{C}_n$. From $\mathsf{ICP_p}$, we know that $\mathtt{follow}(d, \alpha){=}F(\alpha){\in}\mathsf{Nat}$ and $\mathtt{follow}(d, \beta){=}F(\beta){\in}\mathsf{Nat}$. Therefore, it is enough to prove $\mathtt{follow}(d, \alpha){=}\mathtt{follow}(d, \beta){\in}\mathsf{Nat}$, which follows from the following fact, which can be proved by induction on d: $\mathbf{\Pi}\alpha, \beta{:}\mathfrak{C}.\alpha{=}\beta{\in}\mathfrak{C}_{\mathtt{depth}(d)} \to \mathtt{follow}(d, \alpha){=}\mathtt{follow}(d, \beta){\in}\mathsf{Nat}$. ◀

5 Validity of the Inductive Continuity Principle

This section sketches the proof of Thm. 4, which has been formalized in Agda. For simplicity we focus here on functions in $\mathfrak{B} \to \mathsf{Nat}$, but as mentioned in Sec. 3, the principle holds for all functions in $\mathfrak{B}_{\mathsf{SNat}} \to \mathsf{Nat}$ where SNat is a subtype of Nat.

To validate $\mathsf{ICP_p}$ we assume that $\mathrm{TT}_{\mathcal{C}}^{\square}$'s $\square$ modality is a Kripke-like modality, i.e., $\forall(w : \mathcal{W}). \square_w f \to \forall_w^{\sqsubseteq}(f)$. This is used to derive a co-Brouwer tree from an $F \in \mathfrak{B} \to \mathsf{Nat}$. In short, when building a co-Brouwer tree in Step 1 by extending a node with branches for all $n \in \mathsf{Nat}$, if n does not compute to a number in the current world w (which a Kripke modality enforces), it is unclear how this can result in a co-tree in w. It was proved in [10] that $\mathrm{TT}_{\mathcal{C}}^{\square}$ is inconsistent with classical logic when $\square$ is a Kripke modality and $\mathcal{C}$ is instantiated using references, which is expected because continuity contradicts classical logic [38; 39].

5.1 Computing Brouwer Trees

To show that $\mathsf{ICP_p}$ is valid, we must exhibit a $\mathrm{TT}_{\mathcal{C}}^{\square}$ computation that can compute a Brouwer tree from a pure function in $\mathfrak{B} \to \mathsf{Nat}$. This computation is similar to the one provided in [35, Sec.1.3], and proceeds as follows: given $F \in \mathfrak{B} \to \mathsf{Nat}$, $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$ builds a tree in Bt satisfying the condition in Thm. 4, where $\alpha_0 \equiv \lambda\_.\underline{0}$, and loop is defined as follows:

$$\begin{aligned}
\mathsf{loop}(F) &\equiv \mathtt{fix}(\lambda R.\lambda k.\lambda \alpha.\boldsymbol{\nu} x.(x \coloneqq \underline{0});\mathtt{let}\ i = F(\mathsf{upd}(x, \alpha))\ \mathtt{in}\ \mathsf{cases}(x, R, k, \alpha, i))\\
\mathsf{cases}(\delta, R, k, \alpha, i) &\equiv \mathtt{if}\ !\delta < k\ \mathtt{then}\ \eta(i)\ \mathtt{else}\ \digamma(\lambda x.R\ (k + \underline{1})\ \mathtt{append}(k, \alpha, x))
\end{aligned}$$

The goal of this computation is to recursively build a Brouwer tree from the root, by applying F to a finite sequence (essentially, the pair $\langle k, \alpha \rangle$), which corresponds to a path in the tree, and which is extended as long as it does not contain enough information for F to compute a value, i.e., as long as F makes use of more than k values from α.

Note that a finite sequence, or a list, of elements of type A is encoded here as a pair of its length k and a function in $\mathsf{Nat} \to A$ where only its initial segment of length k is relevant. Given a list l given by the pair k and f, the operator $\mathsf{append}(k, f, a) \equiv \lambda x.\mathtt{if}\ x = k\ \mathtt{then}\ a\ \mathtt{else}\ f(x)$ returns a list of length $k+1$ that appends a to l. Lists are defined like this instead of using a W type because $\mathsf{loop}(F)$ applies F to a function with initial segment the list given as argument. Therefore, instead of using an additional operator to turn an element of such a W type into a function, with this encoding lists directly provide such functions.

The computation in [35] uses exceptions to test whether F requires more values than the ones provided in the current finite sequence, while we use here references as in [11]. Exceptions are well-suited to test whether the modulus of continuity is reached, but not to directly compute moduli of continuity. For example, the computation in [32] relies on exceptions and a loop, while the computation in [11] makes use of references and does not require an additional loop because a reference cell can be used to store the moduli of continuity. Instead of using a reference to a Boolean, which would be similar to using an exception, we use here a reference δ that points to a number, and apply F to $\mathtt{upd}(\delta, \alpha)$, as in $\mathsf{WCP_p}$'s realizer, as it allows us to reuse some of the results used in [11] to validate $\mathsf{WCP_p}$.

5.2 Step 1: Building a co-W

First, we prove that from a function $F \in \mathfrak{B} \to \mathsf{Nat}$, we get $\mathsf{loop}(F)\ \underline{0}\ \alpha_0 \in \mathsf{CoDiag}$, where $\mathsf{CoDiag} \equiv M(\mathsf{Nat} +_{\mathsf{p}} \mathtt{Unit}, \lambda a.\mathtt{if}\ a\ \mathtt{then}\ \mathtt{Void}\ \mathtt{else}\ \mathsf{Nat_p})$. We prove this by coinduction, and by inspecting the computation of $\mathsf{loop}(F)$ (see `coSem` in barContP2.lagda). Given $k \in \mathsf{Nat_p}$ and $\alpha \in \mathfrak{B}$, $(\mathsf{loop}(F)\ k\ \alpha)$ first evaluates $F(\mathtt{upd}(\delta, \alpha))$ to i for some "fresh" δ, and then returns $\eta(i)$ if $!\delta < k$, and otherwise returns $\digamma(\lambda x.\mathsf{loop}(F)\ (k + \underline{1})\ \mathsf{append}(k, \alpha, x))$. We now prove $\mathsf{loop}(F)\ k\ \alpha \in \mathsf{CoDiag}$ by cases. If $!\delta < k$ then it remains to prove that $\eta(i) \in \mathsf{CoDiag}$, which is straightforward because $F(\mathtt{upd}(\delta, \alpha)) \in \mathsf{Nat}$, and therefore i too. If $!\delta \nless k$ then it remains to prove $\digamma(\lambda x.\mathsf{loop}(F)\ (k + \underline{1})\ \mathsf{append}(k, \alpha, x)) \in \mathsf{CoDiag}$, which follows from the fact that $\lambda x.\mathsf{loop}(F)\ (k + \underline{1})\ \mathsf{append}(k, \alpha, x) \in \mathsf{Nat_p} \to \mathsf{CoDiag}$, which follows by coinduction.

5.3 Step 2: Case analysis

Using classical logic we analyze two cases: given $t \in M(A, B)$, either t's branches are all finite or there exists an infinite branch, where the type of branches w.r.t. the world w, type A, and family B is defined as follows, a right injection capturing the termination of a branch:

$$\mathsf{Branch} \equiv \forall (n : \mathbb{N}).(\exists (a, b : \mathsf{Term}).w \vDash a{\equiv}a{\in}A \wedge w \vDash b{\equiv}b{\in}B(a)) \vee \top$$

Note that a branch can either be finite if it returns an element of the right disjunct (i.e., $\top$) for some $n \in \mathbb{N}$, or infinite if it always returns an element of the left disjunct for all $n \in \mathbb{N}$. Branches are defined w.r.t. a term t in $W(A, B)$ or in $M(A, B)$, and we say that *a branch* $p \in \mathsf{Branch}$ *is a branch of a term* t if: $\forall (n : \mathbb{N}).p \in_n t$, where $p \in_n t$ is defined recursively as follows (for $\mathsf{shift}(p) \equiv \lambda k.p(k+1)$):

$$p \in_0 t \equiv \top \qquad p \in_{n+1} t \equiv \begin{cases} \exists (f : \mathsf{Term}).t \Mapsto_w \mathsf{sup}(a, f) \wedge \mathsf{shift}(p) \in_n f\ b, \\ \quad \text{when } p(0) \text{ is a left injection of } (a, b, \_, \_) \\ \top, \text{ otherwise} \end{cases}$$

The tree $t \in M(A,B)$ is $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$. In case t's branches are all finite, we show that $t \in W(A,B)$ (Sec. 5.4). In case t has an infinite branch, we derive a contradiction using an argument similar to one used to validate weak continuity in [11] (Sec. 5.5).

5.4 Step 3: Building a W type

In case t's branches are all finite, we prove that if $t \in M(A,B)$ then $t \in W(A,B)$. Again, we use classical logic: assuming $t \notin W(A,B)$ and deriving a contradiction. Given that $t \in M(A,B)$ and $t \notin W(A,B)$, we extract, by coinduction, an infinite co-branch u from t, where the type of co-branches u w.r.t. the world w, type A, and family B, is coinductively defined as follows (see `m2mb` in barContP.lagda):

$$\nu(R.\exists(a,f,b:\mathsf{Term}).u \Rrightarrow_w \mathsf{sup}(a,f) \wedge w \vDash b{\equiv}b{\in}B(a) \wedge R\ f(b))$$

In particular, such a co-branch provides a sequence of Bs. From this co-branch u, we build an infinite branch $p \in \mathsf{Branch}$ (see `mb2path` in barContP.lagda), which is a function from $n \in \mathbb{N}$ to (left injections of) Bs along with their corresponding As, derived by induction on n. From the assumption that t's branches are all finite we obtain that p must also be finite, from which we derive a contradiction (see `m2w` in barContP.lagda).

5.5 Step 4: Termination

In case t, which is here $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$, contains an infinite branch p, we derive a contradiction from F's continuity. Because p is infinite, i.e., only returns left injections, we obtain a metatheoretical function of the following type, which follows the branch p of $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$:

$$\mathbb{N} \to \exists(a,b:\mathsf{Term}).w \vDash a{\equiv}a{\in}\mathsf{BtA} \wedge w \vDash b{\equiv}b{\in}\mathsf{BtB}(a)$$

Therefore, for each $n \in \mathbb{N}$, there are two cases: either ($w \vDash a{\equiv}a{\in}\mathsf{Nat}$ and $w \vDash b{\equiv}b{\in}\mathtt{Void}$) or ($w \vDash a{\equiv}a{\in}\mathtt{Unit}$ and $w \vDash b{\equiv}b{\in}\mathsf{Nat_p}$). Since `Void` is not inhabited, it must be that $w \vDash a{\equiv}a{\in}\mathtt{Unit}$ and $w \vDash b{\equiv}b{\in}\mathsf{Nat_p}$. Hence, from this function, we obtain a metatheoretical function of the following type, which follows the branch p of $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$:

$$\mathbb{N} \to \exists(b:\mathsf{Term}).w \vDash b{\equiv}b{\in}\mathsf{Nat_p}$$

From this function, since $\square$ is a Kripke-like modality, we obtain a metatheoretical function $\boldsymbol{s} \in \mathbb{N} \to \mathbb{N}$, which given $n \in \mathbb{N}$ returns the path taken in the n^{th} $\digamma$ along the branch p following the computation $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$. As explained in Sec. 3.1, $\mathrm{TT}_{\mathcal{C}}^{\square}$'s calculus includes all metatheoretical functions from $\mathbb{N}$ to $\mathbb{N}$, which inhabit $\mathfrak{B}$. These sequences do not have any computational purpose here, and are only used to prove termination. We have $\lceil \boldsymbol{s} \rfloor \in \mathfrak{B}$, so by continuity of F we know that there is a $k \in \mathbb{N}$ such that the k^{th} iteration of $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$ runs $F(\mathtt{upd}(\delta, \lceil \boldsymbol{s} \rfloor))$ for some "fresh" δ such that δ's value stays under k during the computation of $F(\mathtt{upd}(\delta, \lceil \boldsymbol{s} \rfloor))$. This result makes use of `steps-sat-isHighestℕ` in continuity3.lagda, which was used to prove $\mathsf{WCP_p}$ in [11], and in particular to prove that $F(\mathtt{upd}(\delta, \lceil \boldsymbol{s} \rfloor))$ keeps track in δ of the highest number that $\boldsymbol{s}$ is applied to in the computation it performs. The modulus of continuity k of F at $\mathtt{upd}(\delta, \lceil \boldsymbol{s} \rfloor)$ is then the value stored by δ at the end of this computation.

Therefore, because the k^{th} iteration of $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$ runs $F(\mathtt{upd}(\delta, \lceil \boldsymbol{s} \rfloor))$ such that δ's value stays under k, it returns $\eta(i)$ for some i, which contradicts the assumption that the branch is infinite, i.e., contains only $\digamma$s (see `noInfPath` in barContP6.lagda for details).

Note that the k^{th} iteration of $\mathsf{loop}(F)\ \underline{0}\ \alpha_0$ does not quite run $F(\mathtt{upd}(\delta, \lceil \boldsymbol{s} \rfloor))$, but instead $F(\mathtt{upd}(\delta, \alpha))$, where as indicated in Sec. 5.1, α is built starting from α_0 using the append

function, and therefore is equal to $\lceil \boldsymbol{s} \rfloor$ up to k. We can interchangeably use $F(\mathtt{upd}(\delta, \lceil \boldsymbol{s} \rfloor))$ or $F(\mathtt{upd}(\delta, \alpha))$ thanks to Lem. 8 below (see `updSeq-steps-NUM` in barContP6.lagda).

▶ **Definition 7.** *The simulation relation* $t_1 \approx_{\delta, \boldsymbol{s}, n} t_2$ *holds iff*

$$\begin{aligned} &(t_1 = \mathtt{upd}(\delta, \boldsymbol{s}) \wedge t_2 = \mathtt{upd}(\delta, s2l(\boldsymbol{s}, n))) \vee (t_1 = \mathtt{upd}(\delta, s2l(\boldsymbol{s}, n)) \wedge t_2 = \mathtt{upd}(\delta, \boldsymbol{s})) \\ \vee\ &(t_1 = x \wedge t_2 = x) \vee (t_1 = \underline{n} \wedge t_2 = \underline{n}) \vee (t_1 = \lambda x.a \wedge t_2 = \lambda x.b \wedge a \approx_{\delta, \boldsymbol{s}, n} b) \\ \vee\ &(t_1 = (a_1\ b_1) \wedge t_2 = (a_2\ b_2) \wedge a_1 \approx_{\delta, \boldsymbol{s}, n} b_1 \wedge a_2 \approx_{\delta, \boldsymbol{s}, n} b_2) \ \vee\ \ldots \end{aligned}$$

where $s2l(\boldsymbol{s}, 0) \equiv \alpha_0$ *and* $s2l(\boldsymbol{s}, n+1) \equiv \mathsf{append}(n, s2l(\boldsymbol{s}, n), \underline{\boldsymbol{s}(n+1)})$.

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

▶ **Lemma 8.** *If* $a \approx_{\delta, \boldsymbol{s}, n} b$ *and* $a\ {}_{w_1}{\mapsto}^*_{w_2}\ \underline{k}$ *such that* n *is higher than any value held by* δ *throughout this computation, then* $b\ {}_{w_1}{\mapsto}^*_{w_2}\ \underline{k}$.

5.6 Step 5: The Continuity Property

It now remains to prove that given $F \in \mathfrak{B} \to \mathsf{Nat}$, the tree $d \equiv (\mathsf{loop}(F)\ \underline{0}\ \alpha_0) \in \mathsf{Bt}$ satisfies the property $\boldsymbol{\Pi_p} \alpha{:}\mathfrak{B}.\mathtt{follow}(d, \alpha){=}F(\alpha){\in}\mathsf{Nat}$ (see `semCond` in barContP9.lagda). For this we need to prove that $\mathtt{follow}(d, \alpha)$ computes to the same number that $F(\alpha)$ computes to, and this for any pure sequence $\alpha \in \mathfrak{B}$ and tree $d \equiv \mathsf{loop}(F)\ \underline{k}\ \alpha_k$, where α_k agrees with α up to k (see `follow-NUM` in barContP9.lagda). We prove this by induction on d. Either d is an $\eta(i)$, which we discuss below, or a $\digamma(f)$, in which case we conclude by induction. In case d is $\eta(i)$, we must prove that $F(\alpha)$ computes to i. In that case, d runs $F(\mathtt{upd}(\delta, \alpha_k))$ for some "fresh" δ, which computes to i for some α_k that agrees with α up to k. Here α_k is $s2l(\boldsymbol{s}, k)$, for some $\boldsymbol{s}$ equal to α in $\mathfrak{B}$. We use again here a metatheoretical sequence $\boldsymbol{s}$, which does not have any computational purpose. We can then prove that $F(\alpha)$ and $F(\boldsymbol{s})$ compute to the same number, and appealing to Lem. 8, we prove that $F(\boldsymbol{s})$ and $F(\mathtt{upd}(\delta, \alpha_k))$ compute to the same number, and therefore that $F(\alpha)$ computes to i, which concludes our proof.

6 Conclusion and Related Works

The paper presents the first internalization of the inductive dialogue-based continuity principle in a dependent type theory, namely $\mathrm{TT}_{\mathcal{C}}^{\square}$, which has been formalized in Agda. For this, we construct Brouwer trees via effectful computations that use references. Proving the inductive continuity principle internally entails new challenges, such as the termination proof which requires maintaining a strict connection between a meta-theoretical generic element and an internal computation. More generally, the class of effectful intuitionistic theories $\mathrm{TT}_{\mathcal{C}}^{\square}$, which now internalizes several continuity principles, provides a computational framework for further studying the relationship between these principles. WCP and ICP have been shown to coincide in the presence of Bar Induction (under certain restrictions), or assuming classical reasoning [6; 22; 9]. Bar Induction was shown to be consistent with a subsystem of $\mathrm{TT}_{\mathcal{C}}^{\square}$ [34]. Thus, it seems that $\mathrm{TT}_{\mathcal{C}}^{\square}$ provides an ideal framework in which one can formally verify this implication internally, as well as produce a corresponding computation. An immediate related question we leave for further study is then to establish the relation between the two principles in a general setting, without assuming Bar Induction or resorting to classical reasoning.

The technique of using dialogue trees to compute moduli of continuity originated in [20; 22; 21; 19], while the idea of recording the interaction of a function with an oracle to compute continuity goes back to Longley [28], where exceptions and references were used as a probing mechanism to compute moduli of continuity. In [19], Escardó defined a model of System T where $\mathbb{N}$ is interpreted as the type of dialogue trees and function types as functions between the interprations of the source and target types. This model contains a *generic element* of type $\mathbb{N} \to \mathbb{N}$, a function from dialogue trees to dialogue trees, that records queries to it in the structure of the resulting dialogue tree. Then, a dialogue tree is built using this generic element, from which the modulus of continuity can be calculated. Sterling [35] extended the effectful forcing technique to prove that System T validates the realizable bar thesis, which is equivalent to the inductive continuity principle considered here. System T was given a call-by-name interpretation, where types are interpreted as algebras over a dialogue tree monad. Although the carrier sets of this interpretation agree with those of Escardó, the actions of the algebras allow for a compositional interpretation of the recursor on numbers.

In [3], the authors prove that all BTT [29] functions are continuous by generalizing the method of [19]. However, their method does not allow *internalizing* the continuity principle, which is the goal of the present work. As they work in the metatheory, they can induct on the syntax of the $F \in \mathfrak{B} \to \mathsf{Nat}$ when constructing the dialogue trees, allowing for a constructive proof of continuity. In this work, we construct a program computing such trees in the theory itself, where recursion on syntax of terms is not available. As a result we resort to classical logic to prove finiteness of the computed trees and termination of this program. It remains to be seen if this can also be done internally, without resorting to classical logic.

References

[1] Michael Gordon Abbott. "Categories of containers". PhD thesis. University of Leicester, England, UK, 2003.

[2] *Agda Wiki*. http://wiki.portal.chalmers.se/agda/pmwiki.php.

[3] Martin Baillon, Assia Mahboubi and Pierre-Marie Pédrot. "Gardening with the Pythia A Model of Continuity in a Dependent Setting". In: *CSL*. Vol. 216. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 5:1–5:18. DOI: 10.4230/LIPIcs.CSL.2022.5.

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

[5] Stefano Berardi, Marc Bezem and Thierry Coquand. "On the Computational Content of the Axiom of Choice". In: *J. Symb. Log.* 63.2 (1998), pp. 600–622. DOI: 10.2307/2586854.

[6] Nuria Brede and Hugo Herbelin. "On the logical structure of choice and bar induction principles". In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470523.

[7] Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. London Mathematical Society Lecture Notes Series. Cambridge University Press, 1987.

[8] L.E.J. Brouwer. "From frege to Gödel: A Source Book in Mathematical Logic, 1879–1931". In: Harvard University Press, 1927. Chap. On the Domains of Definition of Functions.

[9] Venanzio Capretta and Tarmo Uustalu. "A Coalgebraic View of Bar Recursion and Bar Induction". In: *FOSSACS*. Vol. 9634. LNCS. Springer, 2016, pp. 91–106. DOI: 10.1007/978-3-662-49630-5_6.

[10] Liron Cohen and Vincent Rahli. "Constructing Unprejudiced Extensional Type Theories with Choices via Modalities". In: *FSCD*. Vol. 228. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 10:1–10:23. DOI: 10.4230/LIPIcs.FSCD.2022.10.

[11] Liron Cohen and Vincent Rahli. "Realizing Continuity Using Stateful Computations". In: *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*. Vol. 252. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 15:1–15:18. DOI: 10.4230/LIPIcs.CSL.2023.15.

[12] Thierry Coquand and Guilhem Jaber. "A Computational Interpretation of Forcing in Type Theory". In: *Epistemology versus Ontology*. Vol. 27. Logic, Epistemology, and the Unity of Science. Springer, 2012, pp. 203–213. DOI: 10.1007/978-94-007-4435-6_10.

[13] Thierry Coquand and Guilhem Jaber. "A Note on Forcing and Type Theory". In: *Fundam. Inform.* 100.1-4 (2010), pp. 43–52. DOI: 10.3233/FI-2010-262.

[14] M. J. Cresswell and G. E. Hughes. *A New Introduction to Modal Logic*. Routledge, 1996.

[15] Michael A. E. Dummett. *Elements of Intuitionism*. Second. Clarendon Press, 2000.

[16] Peter Dybjer and Anton Setzer. "A Finite Axiomatization of Inductive-Recursive Definitions". In: *TLCA'99*. Vol. 1581. LNCS. Springer, 1999, pp. 129–146. DOI: 10.1007/3-540-48959-2_11.

[17] Martín Escardó and Paulo Oliva. *Dialogue to Brouwer*. 2017. URL: https://www.cs.bham.ac.uk/~mhe/dialogue/dialogue-to-brouwer.html.

[18] Martín H. Escardó and Chuangjie Xu. “The Inconsistency of a Brouwerian Continuity Principle with the Curry-Howard Interpretation”. In: *TLCA 2015*. Vol. 38. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 153–164. DOI: 10.4230/LIPIcs.TLCA.2015.153.

[19] Martín Hötzel Escardó. “Continuity of Gödel’s System T Definable Functionals via Effectful Forcing”. In: *Electr. Notes Theor. Comput. Sci.* 298 (2013), pp. 119–141. DOI: 10.1016/j.entcs.2013.09.010.

[20] Neil Ghani, Peter G. Hancock and Dirk Pattinson. “Continuous Functions on Final Coalgebras”. In: *CMCS*. Vol. 164. Electronic Notes in Theoretical Computer Science 1. Elsevier, 2006, pp. 141–155. DOI: 10.1016/j.entcs.2006.06.009.

[21] Neil Ghani, Peter G. Hancock and Dirk Pattinson. “Continuous Functions on Final Coalgebras”. In: *MFPS*. Vol. 249. Electronic Notes in Theoretical Computer Science. Elsevier, 2009, pp. 3–18. DOI: 10.1016/j.entcs.2009.07.081.

[22] Neil Ghani, Peter G. Hancock and Dirk Pattinson. “Representations of Stream Processors Using Nested Fixed Points”. In: *Log. Methods Comput. Sci.* 5.3 (2009).

[23] S.C. Kleene. “Recursive Functionals and Quantifiers of Finite Types Revisited I”. In: *Generalized Recursion Theory II*. Vol. 94. Studies in Logic and the Foundations of Mathematics. Elsevier, 1978, pp. 185–222. DOI: https://doi.org/10.1016/S0049-237X(08)70933-9.

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

[25] Georg Kreisel. “On weak completeness of intuitionistic predicate logic”. In: *J. Symb. Log.* 27.2 (1962), pp. 139–158. DOI: http://dx.doi.org/10.2307/2964110.

[26] Saul A. Kripke. “Semantical Analysis of Intuitionistic Logic I”. In: *Formal Systems and Recursive Functions*. Vol. 40. Studies in Logic and the Foundations of Mathematics. Elsevier, 1965, pp. 92–130. DOI: https://doi.org/10.1016/S0049-237X(08)71685-9.

[27] Saul A. Kripke. “Semantical Analysis of Modal Logic I. Normal Propositional Calculi”. In: *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik* 9.5-6 (1963), pp. 67–96. DOI: 10.1002/malq.19630090502.

[28] John Longley. “When is a Functional Program Not a Functional Program?” In: *ICFP’99*. ACM, 1999, pp. 1–7. DOI: 10.1145/317636.317775.

[29] Pierre-Marie Pédrot and Nicolas Tabareau. “An effectful way to eliminate addiction to dependence”. In: *LICS 2017*. IEEE Computer Society, 2017, pp. 1–12. DOI: 10.1109/LICS.2017.8005113.

[30] Andrew M Pitts. *Nominal Sets: Names and Symmetry in Computer Science, volume 57 of Cambridge Tracts in Theoretical Computer Science*. 2013.

[31] Vincent Rahli. “Exercising Nuprl’s Open-Endedness”. In: *ICMS 2016*. Vol. 9725. LNCS. Springer, 2016, pp. 18–27. DOI: 10.1007/978-3-319-42432-3_3.

[32] Vincent Rahli and Mark Bickford. “A nominal exploration of intuitionism”. In: *CPP 2016*. Extended version: http://www.nuprl.org/html/Nuprl2Coq/continuity-long.pdf. ACM, 2016, pp. 130–141. DOI: 10.1145/2854065.2854077.

[33] Vincent Rahli and Mark Bickford. “Validating Brouwer’s continuity principle for numbers using named exceptions”. In: *Mathematical Structures in Computer Science* (2017), pp. 1–49. DOI: 10.1017/S0960129517000172.

[34] Vincent Rahli, Mark Bickford, Liron Cohen and Robert L. Constable. “Bar Induction is Compatible with Constructive Type Theory”. In: *J. ACM* 66.2 (2019), 13:1–13:35. DOI: 10.1145/3305261.

[35] Jonathan Sterling. “Higher order functions and Brouwer’s thesis”. In: *J. Funct. Program.* 31 (2021), e11. DOI: 10.1017/S0956796821000095.

[36] A.S. Troelstra. “A Note on Non-Extensional Operations in Connection With Continuity and Recursiveness”. In: *Indagationes Mathematicae* 39.5 (1977), pp. 455–462. DOI: 10.1016/1385-7258(77)90060-9.

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

[38] Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics An Introduction*. Vol. 121. Studies in Logic and the Foundations of Mathematics. Elsevier, 1988.

[39] Wim Veldman. “Understanding and Using Brouwer’s Continuity Principle”. In: *Reuniting the Antipodes — Constructive and Nonstandard Views of the Continuum*. Vol. 306. Synthese Library. Springer Netherlands, 2001, pp. 285–302. DOI: 10.1007/978-94-015-9757-9_24.

[40] Chuangjie Xu. “A continuous computational interpretation of type theories”. PhD thesis. University of Birmingham, UK, 2015.

[41] Chuangjie Xu and Martín Hötzel Escardó. “A Constructive Model of Uniform Continuity”. In: *TLCA 2013*. Vol. 7941. LNCS. Springer, 2013, pp. 236–249. DOI: 10.1007/978-3-642-38946-7_18.