JFP **31**, e11, 33 pages, 2021.

doi:10.1017/S0956796821000095

Higher order functions and Brouwer's thesis

JONATHAN STERLING

Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
(*e-mail:* jmsterli@cs.cmu.edu)

Abstract

Extending Martín Escardó's *effectful forcing* technique, we give a new proof of a well-known result: Brouwer's monotone bar theorem holds for any bar that can be realized by a functional of type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ in Gödel's **System T**. Effectful forcing is an elementary alternative to standard sheaf-theoretic forcing arguments, using ideas from programming languages, including computational effects, monads, the algebra interpretation of call-by-name λ-calculus, and logical relations.

Our argument proceeds by interpreting **System T** programs as well-founded dialogue trees whose nodes branch on a query to an oracle of type $\mathbb{N} \to \mathbb{N}$, lifted to higher type along a call-by-name translation. To connect this interpretation to the bar theorem, we then show that Brouwer's famous "mental constructions" of barhood constitute an invariant form of these dialogue trees in which queries to the oracle are made maximally and in order.

"The force of ***Church's Law*** *is that it postulates that all future notions of computation will be equivalent in expressive power (measured by definability of functions on the natural numbers) to the λ-calculus. Church's Law is therefore a* scientific law *in the same sense as, say, Newton's Law of Universal Gravitation, which makes a prediction about all future measurements of the acceleration in a gravitational field." (Harper,* 2016*)*

1 Introduction

It is right to rebel! A major theme of Bob Harper's thinking has been the critique of universal computation as a scientific explanation of functional programming, which is fundamentally about *higher order functions* rather than functions between the natural numbers. Indeed, many sensible models of computation do not coincide with Turing computability at higher type, a critical observation for the era of the *socialization of computation*, in which programs increasingly act as interacting nodes in complex systems.

An early exponent of this perspective was the Dutch topologist L.E.J. Brouwer, who based his *intuitionistic* program of mathematics on a profound reconstruction of the notion of higher order function in terms of infinitary dialogues, rejecting both the classical emphasis on static relations, and the (proto-)constructivist fixation on finitude and formal definability.

1.1 Point-set and constructive topology

Classical topology is based on point sets: a topological space is a set of points X together with a *frame* of open subsets $\mathscr{O}_X \subseteq \mathscr{P}(X)$, i.e., a collection of subsets of X closed under

unions and finite intersections. For instance, the real line $\mathbb{R}$ has the set of real numbers for points, and the opens are the unions of open intervals $\cup_\alpha (a_\alpha, b_\alpha)$. A continuous function between topological spaces $f : X \to Y$ is a function of the underlying point sets whose inverse image $f^* : \mathscr{P}(Y) \to \mathscr{P}(X)$ sends opens to opens.

Unfortunately, the theory of topological spaces based on point sets quickly becomes nonconstructive—meaning that most important theorems of topology rely crucially on Zermelo's axiom of choice, or at least the law of the excluded middle. Mathematical theorems proved *without* these axioms are of interest because they correspond to *algorithms* that can, in principle, be executed by a computer: to put it bluntly, a proof that there exists a number satisfying some property carries within it an algorithm to compute such a number.

Brouwer was a second-generation pioneer of this idea that mathematical proof is always a *programming act*, so he rejected both the general law of the excluded middle as well as the axiom of choice; as a professional topologist, Brouwer therefore needed to develop a theory of space that did not place point sets at its center. The purpose of this section is to explain how such a concept of space can exist, and to motivate Brouwer's controversial bar thesis.

In the interest of exposition, we take some historical liberties in Section 1.1.1 with our explanation of how to think of *point-free space*, using more modern language than was available to Brouwer in his lifetime, before turning to Brouwer's account in Section 1.1.2. There are by now many references that introduce modern perspectives on point-free topology (Johnstone, 1982, 1983; Mac Lane & Moerdijk, 1992; Vickers, 1989, 2007; Sambin, 2012; Anel & Joyal, 2019).

1.1.1 The algebraic structure of opens and constructive spaces

Many aspects of a topological space can be understood in terms of its open sets; for instance, if X is a Hausdorff space, the point set of X can be reconstructed from the set of *completely prime filters* on $\mathscr{O}_X$—the definition of a prime filter here is not as important as the fact that the point set can be reconstructed from purely algebraic data on open sets.

This suggests that one might define a *new* kind of "point-free space" generalizing Hausdorff spaces by taking open sets and inverse image functions as primitive, and deriving point sets and continuous functions between point sets from these. It turns out that most aspects of point-set topology that require classical axioms will become fully constructive when restated for these new point-free spaces.

The algebraic structure of the collection of open sets of a topological space is concentrated in the concept of a *frame*.

Definition 1.1 (Frame)**.** *A* frame *is a partially ordered set F closed under finite meets $U \wedge V$ and set-indexed joins $\bigvee_{\alpha \in S} U_\alpha$, such that meets distribute across joins. A morphism of frames, written $f^* : F \to G$, is a morphism of the underlying poset that preserves finite meets and set-indexed joins.*

The morphisms of frames abstract the behavior of the *inverse images* $f^* : \mathscr{O}_Y \to \mathscr{O}_X$ of continuous maps of topological spaces $f : X \to Y$. We obtain, therefore, a possible notion of *constructive space* by considering the formal dual of frames—by "dual" we mean that

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

a constructive space is just a frame, but a morphism between constructive spaces goes in the opposite direction.[1]

Definition 1.2 (Constructive space)**.** *A* constructive space X *is defined by a frame, conventionally called* $\mathcal{O}_X$*; a* continuous map *of constructive spaces* $f : X \to Y$ *is given by a morphism of frames* $f^* : \mathcal{O}_Y \to \mathcal{O}_X$*, called the* inverse image*.*

Then, geometrical figures like points are defined synthetically in terms of continuous maps. The constructive space corresponding to a single point $*$ is given by the frame $\mathcal{O}_* = \{\bot < \top\}$; then, one may define a point of an arbitrary constructive space X to just be a continuous map $x : * \to X$. Unraveling definitions, this turns out to be the same as a completely prime filter in $\mathcal{O}_X$, but it is not important for our purposes to know this definition.

1.1.2 Brouwer's presentation of constructive spaces as spreads

Intuitionists (like Brouwer and Weyl) and their forbears (like Poincaré) were not only concerned with the law of the excluded middle and the axiom of choice; they also did not accept the use of impredicative definitions, where the object being defined may implicitly refer to itself. In particular, the existence of the *powerset* $\mathcal{P}(X)$, the set of *all* subsets of X, was considered doubtful. Unfortunately, many aspects of the theory of constructive spaces are impredicative in just this sense, but it is possible to present constructive spaces by generators and relations in a predicative way—just like the infinite set of natural numbers can be presented by finitely many generators in a programming language:

```
datatype nat = Z | S of nat
```

For his purposes in developing intuitionistic analysis, i.e., the study of the continuum from the intuitionistic perspective, Brouwer developed such a presentation of constructive spaces called *spreads*, in which the algebraic structure of opens is made concrete in terms of finite paths in an infinitely broad and infinitely deep rooted tree.

Notation 1.3 (Lists)**.** *If* S *is a set, then* $\mathbf{list}(S)$ *is the set of lists of elements of* S*; we will use the following notations:*

1. $\langle\rangle$ *is the empty list.*
2. $\langle u \rangle$ *is the singleton list given* $u : S$*.*
3. $\vec{u} +\!\!+ \vec{v}$ *is the list obtained by appending* $\vec{v}$ *to the end of* $\vec{u}$*.*
4. *We write* $\mathrm{len}(\vec{u})$ *for the length of the list* $\vec{u}$*.*
5. *We write* $v :: \vec{u}$ *for* $\langle v \rangle +\!\!+ \vec{u}$*, and* $\vec{u} \,\hat{}\, v$ *for* $\vec{u} +\!\!+ \langle v \rangle$*.*
6. *We write* $\vec{u} \preceq \vec{v}$ *to assert that* $\vec{u}$ *is a prefix of* $\vec{v}$*.*

Notation 1.4 (Infinite sequences)**.** *If* S *is a set, then* $\mathbb{N} \to S$ *is the set of infinite sequences of elements of* S*, which we manipulate using the following notations:*

1. *We write* $\mathrm{take}(k, \alpha)$ *to mean the list comprised of the first* k *elements of* α*.*
2. *We write* $\alpha \succ \vec{u}$ *or* $\vec{u} \prec \alpha$ *to assert that* $\vec{u} : \mathbf{list}(S)$ *is a finite prefix of* $\alpha : \mathbb{N} \to S$*.*

[1] We use *constructive space* to refer to what is called a *locale* in the literature.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

Definition 1.5 (Spread law)**.** *A* spread law *is a decidable set* $\mathscr{B}$ *of lists* $\vec{u} : \mathbf{list}(\mathbb{N})$ *called "basic observations",*[2] *satisfying the following axioms that make* $\mathscr{B}$ *encode an infinitely wide and deep rooted tree:*

1. *The empty sequence* $\langle\rangle$ *is in* $\mathscr{B}$*; this ensures that the resulting tree is rooted.*
2. *If* $\vec{v} \in \mathscr{B}$*, then for any prefix* $\vec{u} \preceq \vec{v}$*, we have* $\vec{u} \in \mathscr{B}$*.*
3. *If* $\vec{u} \in \mathscr{B}$*, then there exists some* $k : \mathbb{N}$ *such that the extension* $\vec{u}\,\hat{}\,k$ *is in* $\mathscr{B}$*; this ensures that the tree is infinitely deep.*

To view the spread law as an encoding of a tree, think of each $\vec{u} \in \mathscr{B}$ as a sequence of directions to move from the root of the tree to one of its nodes. A *spread*, defined below, is the geometrical object corresponding to the algebraic data of a spread law.

Definition 1.6 (Spread)**.** *A* spread $\boldsymbol{S}$ *is defined by the data of a spread law* $\mathscr{B}_{\mathbf{S}}$*; one thinks of* $\boldsymbol{S}$ *as the "space of infinite paths" of the tree encoded by* $\mathscr{B}_{\mathbf{S}}$*.*

It is useful to switch freely between the intuition of $\vec{u}$ as a node in a tree and of $\vec{u}$ as an observation about an infinite path in a tree, substantiated by the fact that each list $\vec{u}$ determines the set of infinite sequences that it prefixes. While the data of a spread is the same as the data of a spread law, we define them separately because a function between spread laws, suitably defined, should encode the *inverse image* part of a continuous function between spreads.

Definition 1.7 (Points)**.** *A point of a spread* $\boldsymbol{S}$ *is an infinite path in the tree encoded by* $\mathscr{B}_{\mathbf{S}}$*, i.e., an infinite sequence* $\alpha : \mathbb{N} \to \mathbb{N}$ *such that each finite prefix* $\vec{u} \prec \alpha$ *is in* $\mathscr{B}_{\mathbf{S}}$*, i.e.,* $\mathrm{take}(k, \alpha) \in \mathscr{B}_{\mathbf{S}}$ *for all* $k : \mathbb{N}$*.*

A basic observation in a spread is then conceptually similar to a basic open in a basis for a topology, e.g., an open interval on the real line. From this perspective, the empty list $\langle\rangle$ represents the entire spread $\mathbf{S}$, and a general prefix $\vec{u}$ represents the subspace of $\mathbf{S}$ spanned by points that begin with $\vec{u}$.

Example 1.8 (Real numbers)**.** Of central importance is the *continuum* $\mathbf{R}$, the spread of real numbers. One construction of this spread involves an encoding of rational intervals as natural numbers; then a list of natural numbers $\vec{u}$ is admitted in $\mathscr{B}_{\mathbf{R}}$ just when its elements encode increasingly smaller nested rational intervals each of the form $\left[\frac{a}{2^{n-1}}, \frac{a+1}{2^{n-1}}\right]$. A point α in the resulting spread $\mathbf{R}$ is an encoding of a real number, and one may identify the collection $\mathbb{R}$ of real numbers with the equivalence classes of $\mathbf{R}$-points under the suitable relation (van Atten *et al.*, 2002).

Definition 1.9 (Fans, or finitary spreads)**.** *A spread whose nodes are all finitarily branching is called a* fan*; an example of such a finitary spread is the fan* $\mathbf{I}$ *representing the closed interval* $[0, 1] \subset \mathbb{R}$*. As above, we may write* $\mathbb{I}$ *for the appropriate collection of equivalence classes of* $\mathbf{I}$*-points.*

[2] The decidability of $\vec{u} \in \mathscr{B}$ of course is only meaningful in a setting where we have not assumed the law of the excluded middle.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

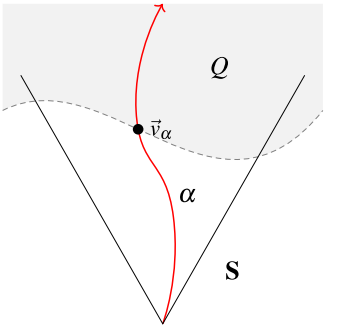


Fig. 1. A visualization of when a monotone subset $Q \subseteq \mathscr{B}_{\mathbf{S}}$ is a bar, i.e., $\langle\rangle \triangleleft Q$. The dotted line indicates the beginning of the bar, and α is an arbitrary point in the spread; $\vec{v}_\alpha$ is a prefix of α lying in Q.

1.2 Brouwer's bar thesis: higher order functions as trees

In classical topology, an *open cover* over an open $U \in \mathscr{O}_X$ is a set of opens $\{V_i \in \mathscr{O}_X \mid i : I\}$ such that every point $x \in U$ is contained in one of the V_i. Brouwer defined an analogous notion of cover for spreads, which he called a *bar*, also in terms of points.

Definition 1.10 (Bars in terms of points)**.** *Let* ***S*** *be a spread and let* $\vec{u} \in \mathscr{B}_{\mathbf{S}}$ *be a basic observation; then a* bar *over* $\vec{u}$ *is given by a subset* $Q \subseteq \mathscr{B}_{\mathbf{S}}$ *satisfying the following axioms:*[3]

1. Monotonicity. *If* $\vec{v}$ *is in* Q *and* $\vec{v} \preceq \vec{w}$*, then* $\vec{w}$ *is in* Q*.*
2. Covering. *For every point* $\alpha \succ \vec{u}$*, there exists some basic observation* $\vec{v}_\alpha \in Q$ *such that* $\alpha \succ \vec{v}_\alpha$*.*

We write $\vec{u} \triangleleft Q$ ***to assert that*** Q ***is a bar over*** $\vec{u}$*; colloquially, the* covering *axiom states that every infinite path* α *eventually "hits the bar" Q, as depicted in Figure 1. It is useful to observe that the covering axiom is equivalent to the assertion that for every* $\alpha \succ \vec{u}$*, there exists* $k : \mathbb{N}$ *such that* $\mathsf{take}(k, \alpha) \in Q$*.*

Given that the empty observation $\langle\rangle$ represents the entire spread **S** (because every point α is prefixed by $\langle\rangle$), one refers to a bar of **S** as a subset $Q \subseteq \mathscr{B}_{\mathbf{S}}$ such that $\langle\rangle \triangleleft Q$.

1.2.1 Intuitionistically, bars are higher order functions

From the point of view of the Brouwer–Heyting–Kolmogorov interpretation of the logical connectives (Heyting, 1956), to say that Q is a bar over $\vec{u}$ is the same as to require a *higher order function* $\phi : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ such that for all $\alpha \succ \vec{u}$, we have $\mathsf{take}(\phi(\alpha), \alpha) \in Q$. Intuitively, this higher order function receives a point and responds with how far one must proceed along the point's approximations in order to reach the bar.

[3] We use the term *bar* to mean what much of the literature refers to as a *monotone bar*. We do not require that membership in the subset $Q \subseteq \mathscr{B}_{\mathbf{S}}$ be decidable.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

1.2.2 Bars represented by well-founded trees

It is an empirical reality that one cannot define a discontinuous function on the reals by computational means—this is because such functions always rely on the ability to branch on whether two real numbers are equal, or something analogous, which cannot be achieved in finite time. Consequently, it was important for Brouwer's intuitionistic version of analysis to reflect this limitation in a *continuity theorem*: all functions defined on the reals are continuous.

In fact, the crown jewel of Brouwer's intuitionistic analysis is the verification of the even stronger *uniform continuity* theorem for all total functions defined on the closed interval, which exhibits not only a modulus of continuity for such functions, but one that is independent of the input:

$$\forall F : \mathbb{I} \to \mathbb{R}.\ \forall \epsilon : \mathbb{R}^+.\ \exists \delta : \mathbb{R}^+.\ \forall x_0, x_1 : \mathbb{I}.\ |x_0 - x_1| < \delta \Rightarrow |f(x_0) - f(x_1)| < \varepsilon \qquad \text{(UC)}$$

For intuitionistic mathematicians, uniform continuity is important because it reflects the limitations of human facilities for manipulating infinitary data: a human can only make finitely many observations of infinite data. The uniformity condition expresses that, in the case of the closed interval, one may bound the number of observations *regardless* of the point being observed. From a computer science point of view, the uniform continuity theorem suggests a realistic invariant on nodes that consume and emit bits of data: there is a bound on how many messages it takes to engender a response.

Because Brouwer represented the closed interval $\mathbb{I}$ in terms of the points of a fan (Definition 1.9), he could reduce the statement of uniform continuity to a combination of *ordinary* continuity and his famous fan theorem, a statement about *bars* Q on the fan $\mathbf{F}$ whose basic observations are all lists of bits 0, 1:

$$\langle\rangle \lhd Q \Rightarrow \exists k : \mathbb{N}.\ \forall \alpha : \mathbb{N} \to \mathbf{bool}.\ \mathrm{take}(k, \alpha) \in Q \qquad \text{(FT)}$$

Observe that the fan theorem is nothing more than a quantifier rotation: the assumption that Q is a bar associates to each sequence α the length k of a basic observation of α that lands in the bar; the conclusion of the fan theorem establishes a uniform bound on the length of this finite prefix for all α.

Brouwer argued for his fan theorem by reflecting on the possible ways to construct a proof of the antecedent $\vec{u} \lhd Q$; he hypothesized that such a proof could always be presented as a well-founded tree governed by the following primitive inferences:

1. η-*inference.* If $\vec{u} \in Q$, then Q bars $\vec{u}$.
2. $\digamma$-*inference.*[4] If Q bars every immediate subnode $\vec{u}\,{}^\frown x$ of $\vec{u}$, then Q bars $\vec{u}$.

In other words, Brouwer believed that a proof of $\vec{u} \lhd Q$ could always be tabulated into a proof of an alternative, *inductive* characterization of barhood $\vec{u} \blacktriangleleft Q$, defined relative to a spread $\mathbf{S}$:

$$\frac{\vec{u} \in Q}{\vec{u} \blacktriangleleft Q}\ \eta \qquad\qquad \frac{\vec{u} \in \mathscr{B}_{\mathbf{S}} \quad \forall x : \mathbb{N}.\ \vec{u}\,{}^\frown x \in \mathscr{B}_{\mathbf{S}} \Rightarrow \vec{u}\,{}^\frown x \blacktriangleleft Q}{\vec{u} \blacktriangleleft Q}\ \digamma$$

[4] Brouwer used $\digamma$, an archaic Greek letter pronounced "digamma".

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

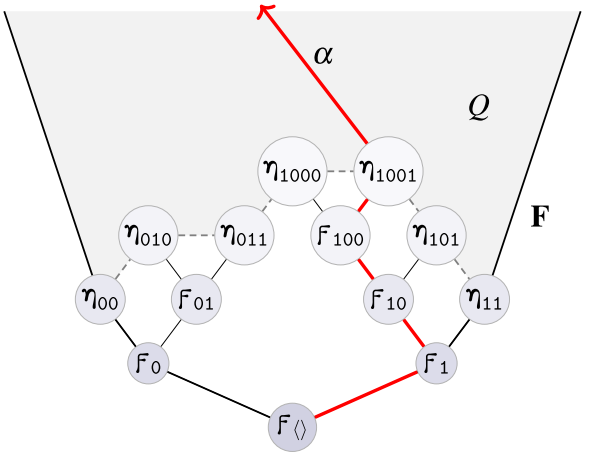


Fig. 2. A visualization of the *inductive* characterization of barhood $\langle\rangle \blacktriangleleft Q$ in a fan $\mathbf{F}$ consisting of sequences of bits. (Contrast with Figure 1.) In the depicted tree, each node is labeled with the list of bits that it codes; to be precise, a node labeled $\eta_{\vec{u}}$ represents a proof of $\vec{u} \blacktriangleleft Q$ that is a leaf node, and a node labeled $\mathsf{F}_{\vec{u}}$ represents a proof of $\vec{u} \blacktriangleleft Q$ that is a branching node.

One way to think of these trees is as a *dialogue* between an actor and a choice sequence α: at each step, the actor has either decided upon an answer (the η-inference), or asks the choice sequence what its next element is (the F-inference), and continues to interact using that information.

If every proof of $\vec{u} \triangleleft Q$ could in fact be tabulated into such a dialogue $\vec{u} \blacktriangleleft Q$ (presented visually in Figure 2), then it would be a simple matter to establish the uniform bound k by induction:

1. η-*inference.* We have $\vec{u} \in Q$, so assign $k = \mathrm{len}(\vec{u})$.
2. F-*inference.* By induction, we have for all possible extensions $\vec{u}\,{}^\frown x$, a bound k_x; because $\mathbf{F}$ is a fan, there are only finitely many such extensions $\vec{u}\,{}^\frown x$, so we simply choose $k = \max\{k_x \mid x\}$.

Theorem 1.11 (Soundness of inductive barhood)**.** *The inductive characterization of barhood is sound: if $\vec{u} \blacktriangleleft Q$ is proved, then $\vec{u} \triangleleft Q$ holds.*

Proof. We proceed by induction on the proof of $\vec{u} \blacktriangleleft Q$ to prove that prove that $\vec{u} \triangleleft Q$.

1. η-*inference.* Then $\vec{u} \in Q$, so we choose $k = \mathrm{len}(\vec{u})$.
2. F-*inference.* By induction, we have $\vec{u}\,{}^\frown x \triangleleft Q$ for all possible extensions x with a bound k; but this implies $\vec{u} \triangleleft Q$ immediately, with bound $k + 1$. ■

In order to prove his fan theorem (and thence establish uniform continuity for real functions on the closed interval), Brouwer wished to prove a converse to Theorem 1.11:

Proposition 1.12 (Brouwer's bar thesis)**.** *The inductive characterization of barhood is complete: if $\vec{u} \triangleleft Q$ then $\vec{u} \blacktriangleleft Q$.*

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

1.3 Functional programming interpretation: exceptions as shared secrets

It is possible to give an interpretation of Brouwer's bar thesis in terms of effectful functional programming in Standard ML, in the spirit of the "seemingly impossible functional programs" of Escardó (2007); Bauer (2006); Rahli *et al.* (2017). As we noted, a bar $\vec{u} \triangleleft Q$ can be thought of as an operation that assigns to each infinite sequence $\alpha \succ \vec{u}$ a number $k : \mathbb{N}$ such that take$(k, \alpha) \in Q$. Hence, we may represent such "functional bars" by a type of higher order functions:

```
type fun_bar = (nat → nat) → nat
```

On the other hand, an inductive bar can be represented in Standard ML by the following datatype of infinitely branching trees, in which the `SPIT` constructor encodes the η-inference and the `BITE` constructor encodes the $\digamma$-inference:

```
datatype ind_bar =
   SPIT of nat
 | BITE of (nat → ind_bar)
```

The argument of `SPIT` contains the number of steps that were necessary to reach the bar. Then, we may define an (effectful) algorithm to tabulate a functional bar `Q : fun_bar` as a tree `tabulate Q : ind_bar`, keeping track of the list of numbers `xs : nat list` we have received from `BITE` so far. The tabulation loop works in the following way:

1. If `Q` α `= Q` β gives the same answer for all α, β `: nat → nat` that start with `xs`, then we have reached the bar; therefore return `SPIT`.
2. Otherwise, we require at least another digit of information; call `BITE`, abstracting `x:nat` and recursing with `xs := xs @ [x]`.

The first step is nontrivial, and we may accomplish it using a computational effect. Given a fresh exception `E : exn`, we may define the "generic infinite sequence" extending `xs` by indexing into `xs` and raising `E` when out of bounds:

```
fun generic (E : exn) (xs : nat list) : nat → nat =
  fn i ⇒
    List.nth (xs, i) handle
      Subscript ⇒ raise E
```

The idea is that we may use the exception `E` as a *shared secret* in our tabulation loop[5], to discover that `Q` has not yet converged and requires a further digit.

[5] The view of Standard ML-style exceptions as shared secrets is explored by Harper in his blog and in his book (Harper, 2012, 2016).

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

```
fun tabulate (Q : fun_bar) : ind_bar =
  let
    fun loop xs =
      let
        exception Secret
      in
        SPIT (Q (generic Secret xs)) handle
          Secret ⇒ BITE (fn x ⇒ loop (xs @ [x]))
      end
  in
    loop []
  end
```

The use of a computational effect in the Standard ML implementation of the tabulation is the essence of what is meant by Escardó's term *effectful forcing* (Escardó, 2013); indeed, the proof of our main result follows the moral arc of the code presented above, likewise hinging on the existence of a generic sequence that has the ability to "spy" on the internals of any function that calls it in the manner of a debug harness.

1.3.1 Relation to the bar thesis

Does the `tabulate` function terminate? The answer to this question depends entirely on what arguments `Q : fun_bar` actually can be defined, which is in essence what Brouwer's bar thesis aims to control. Constructive mathematics is compatible with interpretations in which certain bizarre higher order functions `Q` exist for which `tabulate Q` does not terminate; the most famous such case arises from the *Kleene tree*, an object that exists in the Church–Turing interpretation of second-order functions as first-order functions on codes (Troelstra & van Dalen, 1988).

1.4 Perspective and contribution

Returning to the more modern perspective on constructive topology presented in Section 1.1, the bar thesis enunciated in Proposition 1.12 states that a certain constructive space **B** is completely determined by the topological space obtained from its points; but the spatiality of **B**, a classical triviality (Dummett, 2000), is independent of constructive mathematics.

Brouwer's "proof" of Proposition 1.12 was therefore not successful. Considering the Brouwer–Heyting–Kolmogorov interpretation of a proof of $\vec{u} \triangleleft Q$ as a higher order function, to accept Brouwer's bar thesis is to require that higher order functions of a certain type can always be tabulated into well-founded trees or dialogues. Under the standard Church–Turing interpretation of higher order functions, this tabulation is refuted (Troelstra & van Dalen, 1988); far from a no-go theorem, this is in fact an invitation to consider alternative notions of computability that justify Brouwer's identification of higher order functions $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ with inductive dialogues.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

In this paper, we explain one possible notion of computability under which Brouwer's bar thesis is justified: definability in Gödel's **System T**. Our work is an extension of Martín Escardó's *effectful forcing* technique (Escardó, 2013), including a novel normalization theorem for certain interaction trees into Brouwer's $(\eta, \digamma)$-trees.

1.5 Formalization in Agda

The main results in this paper have been formalized in the Agda proof assistant (Norell, 2009). We carried out our proof in Martin-Löf 's Intensional Type Theory extended by the function extensionality axiom, which was needed in order to prove the monad laws for Escardó's dialogue monad; we do not assume the unicity of identity proofs, and therefore our proof could in principle be converted to Cubical Agda (Vezzosi *et al.*, 2019), in which function extensionality computes. Our formalization can be accessed as follows:

1. Browsable source: http://jonsterling.github.io/agda-effectful-forcing
2. Repository: http://www.github.com/jonsterling/agda-effectful-forcing

2 Gödel's System T

In his famous *Dialectica* paper, Kurt Gödel introduced **System T** to serve as a formal theory of constructions for Heyting arithmetic (Gödel, 1958); in modern terms, **T** is a minimal, total functional programming language with function types $\sigma \Rightarrow \tau$ and a base type of natural numbers $\mathtt{nat}$.

$$
\begin{array}{llll}
\textit{(contexts)} & \Gamma, \Delta & ::= & \cdot \mid \Gamma, x : \sigma \\
\textit{(types)} & \sigma, \tau & ::= & \mathtt{nat} \mid \sigma \Rightarrow \tau \\
\textit{(terms)} & M, N & ::= & x \mid \mathtt{fn}\, x \Rightarrow M \mid M \mathbin{@_\sigma} N \mid \mathtt{z} \mid \mathtt{s}(M) \mid \\
 & & & \mathtt{rec}\, N\, [\mathtt{z} \Rightarrow M_z \mid \mathtt{s}(x)\ \mathtt{with}\ h \Rightarrow M_s]
\end{array}
$$

In the term for functional abstraction $\mathtt{fn}\, x \Rightarrow M$, the variable x is bound; likewise, in the term for the recursor $\mathtt{rec}\ N\ [\mathtt{z} \Rightarrow M_z \mid \mathtt{s}(x)\ \mathtt{with}\ h \Rightarrow M_s]$, the variables x, h are both bound.

Convention 2.1 (Variable binding)**.** *In everything that follows, terms are considered modulo α-equivalence (i.e., up to permutations of bound variables), and we assume the ability to freely obtain a fresh variable that is not free in a term.*

2.1 Static semantics of System T

The theory of **System T** is given by two forms of judgment:

1. $\boxed{\Gamma\ \mathit{ctx}}$ asserts that Γ is a well-formed context. Contexts assign types to the free variables that are in scope.
2. $\boxed{\Gamma \vdash M : \sigma}$ asserts that M is a well-formed element of type σ in context Γ; this judgment is only defined when $\Gamma\ \mathit{ctx}$.

We omit definitional equality entirely, as our dialogue model will be too intensional to validate the usual equations of **System T**.[6] We do not need a separate judgment $\sigma\ \mathit{type}$,

[6] We thank Pierre-Marie Pédrot for pointing this out to us.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

because all raw types σ generated by the grammar above are trivially well-formed. The rules for forming contexts are quite simple, and have the main role of preventing variable shadowing (a simplifying measure that will make semantic interpretation easier):

$$\frac{}{\cdot\ ctx} \qquad \frac{\Gamma\ ctx \qquad x \notin |\Gamma|}{\Gamma, x:\sigma\ ctx}$$

(where $|\Gamma|$ is the set of variables declared in Γ)

The well-typed terms are generated by standard rules of inference; for instance, a variable x is well-formed if it appears in the context:

$$\text{VARIABLE}\ \frac{}{\Gamma, x:\sigma, \Delta \vdash x:\sigma}$$

2.1.1 The function type

The elements of the function type $\sigma \Rightarrow \tau$ are characterized by an abstraction and an application rule. In our version of **T**, we have included just enough typing annotations in the syntax to guarantee the existence of a coherent interpretation of *judgments* $\Gamma \vdash M:\sigma$ into models; for this reason, we include a typing annotation on function application $M @_\sigma N$, but have no need for a typing annotation on abstraction $\mathtt{fn}\, x \Rightarrow M$:

$$\text{FUNCTION ABSTRACTION}\ \frac{\Gamma, x:\sigma \vdash M:\tau}{\Gamma \vdash \mathtt{fn}\, x \Rightarrow M : \sigma \Rightarrow \tau} \qquad \text{FUNCTION APPLICATION}\ \frac{\Gamma \vdash M:\sigma \Rightarrow \tau \qquad \Gamma \vdash N:\sigma}{\Gamma \vdash M @_\sigma N : \tau}$$

2.1.2 Natural numbers and recursion

The natural numbers are given a unary encoding in our presentation of **T**, where $\mathtt{z}$ codes 0 and $\mathtt{s}(N)$ codes $n+1$ if N codes n; a numeral $n:\mathbb{N}$ is therefore coded as the iterated application $\mathtt{s}^n(\mathtt{z})$.

$$\text{ZERO}\ \frac{}{\Gamma \vdash \mathtt{z}:\mathtt{nat}} \qquad \text{SUCCESSOR}\ \frac{\Gamma \vdash M:\mathtt{nat}}{\Gamma \vdash \mathtt{s}(M):\mathtt{nat}}$$

Programs on natural numbers in **T** are written by *primitive recursion*, at possibly higher type.[7] The recursor is typed as follows:

$$\text{RECURSION}\ \frac{\Gamma \vdash N:\mathtt{nat} \qquad \Gamma \vdash M_z:\sigma \qquad \Gamma, x:\mathtt{nat}, h:\sigma \vdash M_s:\sigma}{\Gamma \vdash \mathtt{rec}\ N\ [\mathtt{z} \Rightarrow M_z \mid \mathtt{s}(x)\ \mathtt{with}\ h \Rightarrow M_s] : \sigma}$$

The recursor should be thought of as a construct for defining a function $F:\mathtt{nat} \Rightarrow \sigma$ and then immediately applying it to the target N; in the case for the successor, one binds not only $x:\mathtt{nat}$ but also $h:\sigma$, which intuitively stands for $F(x)$ (the induction hypothesis).

[7] Sometimes the term *primitive recursion* is restricted to mean recursion at base type.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

2.2 Programming in System T

To gain some intuition for higher order primitive recursion, we work through some examples.

Convention 2.2 (Informal notation for **T**). *In order to make our examples more clear, we impose some informal notation for terms.*

1. *Nested function abstractions are written all at once: for instance* $\mathtt{fn}\, x, y \Rightarrow M$ *means* $\mathtt{fn}\, x \Rightarrow \mathtt{fn}\, y \Rightarrow M$.
2. *Numeric constants* $\bar{n}$ *can be written instead of their encodings; for instance* $\bar{2}$ *means* $\mathtt{s}(\mathtt{s}(\mathtt{z}))$.
3. *Function applications can be written* $M(N)$ *rather than* $M \mathbin{@}_\sigma N$, *when the* σ *can be easily inferred from context.*
4. *A function defined by primitive recursion can be written as* $\mathtt{rec}\, [\mathtt{z} \Rightarrow M_z \mid \mathtt{s}(x)\ \mathtt{with}\ h \Rightarrow M_s]$ *rather than the abstracted* $\mathtt{fn}\, y \Rightarrow \mathtt{rec}\, y\, [\mathtt{z} \Rightarrow M_z \mid \mathtt{s}(x)\ \mathtt{with}\ h \Rightarrow M_s]$.

The simplest example of programming using primitive recursion is given by the addition function:

$$\mathtt{plus} : \mathtt{nat} \Rightarrow \mathtt{nat} \Rightarrow \mathtt{nat}$$
$$\mathtt{plus} \triangleq \mathtt{fn}\, x, y \Rightarrow \mathtt{rec}\, x\, [\mathtt{z} \Rightarrow y \mid \mathtt{s}(\_)\ \mathtt{with}\ h \Rightarrow \mathtt{s}(h)]$$

Another version of the addition function can be given that uses primitive recursion at higher type:

$$\mathtt{plus}' : \mathtt{nat} \Rightarrow \mathtt{nat} \Rightarrow \mathtt{nat}$$
$$\mathtt{plus}' \triangleq \mathtt{rec}\, [\mathtt{z} \Rightarrow (\mathtt{fn}\, y \Rightarrow y) \mid \mathtt{s}(\_)\ \mathtt{with}\ h \Rightarrow (\mathtt{fn}\, y \Rightarrow \mathtt{s}(h(y)))]$$

Higher order primitive recursion is stronger than ordinary primitive recursion; for instance, Ackermann's function $\mathscr{A} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is not primitive recursive, but we will see that it can be defined in **T**.[8]

$$\mathscr{A}(0, n) = n + 1$$
$$\mathscr{A}(m + 1, 0) = \mathscr{A}(m, 1)$$
$$\mathscr{A}(m + 1, n + 1) = \mathscr{A}(m, \mathscr{A}(m + 1, n))$$

In **T**, Ackermann's function $\mathscr{A}$ is encoded by a program $\mathtt{ack} : \mathtt{nat} \Rightarrow \mathtt{nat} \Rightarrow \mathtt{nat}$ that computes a *function* by recursion, as in Harper (Harper, 2016):

$$\mathtt{ack} : \mathtt{nat} \Rightarrow \mathtt{nat} \Rightarrow \mathtt{nat}$$
$$\mathtt{ack} \triangleq \mathtt{rec}\, \left[\mathtt{z} \Rightarrow (\mathtt{fn}\, n \Rightarrow \mathtt{s}(n)) \mid \mathtt{s}(\_)\ \mathtt{with}\ g \Rightarrow \left(\mathtt{fn}\, n \Rightarrow \mathtt{iter}(g)(n)(g(\bar{1}))\right)\right]$$

where we define the n-fold iteration of a function as follows:

$$\mathtt{iter} : (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat} \Rightarrow (\mathtt{nat} \Rightarrow \mathtt{nat})$$
$$\mathtt{iter} = \mathtt{fn}\, f \Rightarrow \mathtt{rec}\, [\mathtt{z} \Rightarrow (\mathtt{fn}\, x \Rightarrow x) \mid \mathtt{s}(\_)\ \mathtt{with}\ g \Rightarrow (\mathtt{fn}\, x \Rightarrow f(g(x)))]$$

[8] A three-parameter variant of Ackermann's function was discussed by Hilbert in his 1925 lecture *Über das Unendliche* (Hilbert, 1926), later proved by Ackermann to not be primitive recursive (Ackermann, 1928).

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

The ability to use recursion to define functions of arbitrary order is the hallmark of **System T**.

2.3 Standard semantics of System T

In order to state what it means for a bar Q to be "realizable in **System T**", we must first explain the standard/Tarski-style semantics of **System T**, in which $\mathtt{nat}$ is interpreted by the ordinary set of natural numbers. We begin by defining the basic domains of interpretation, starting with contexts and types:

$$\begin{aligned}[\![\Gamma]\!] &= \textstyle\prod_{x\in|\Gamma|}[\![\Gamma(x)]\!]\\ [\![\mathtt{nat}]\!] &= \mathbb{N}\\ [\![\sigma \Rightarrow \tau]\!] &= [\![\sigma]\!] \to [\![\tau]\!]\end{aligned}$$

A term $\Gamma \vdash M : \sigma$ is interpreted as a *function* from $[\![\Gamma]\!]$ to $[\![\sigma]\!]$ in the obvious way:

$$\begin{aligned}[\![M]\!] &: [\![\Gamma]\!] \to [\![\sigma]\!]\\ [\![x]\!]g &= g(x)\\ [\![\mathtt{fn}\, x \Rightarrow M]\!]g &= \lambda a.\ [\![M]\!](g, x \mapsto a)\\ [\![M\ @_\sigma\ N]\!]g &= \big([\![M]\!]g\big)\big([\![N]\!]g\big)\\ [\![\mathtt{z}]\!]g &= 0\\ [\![\mathtt{s}(M)]\!]g &= 1 + [\![M]\!]g\\ [\![\mathtt{rec}\ N\ [\mathtt{z} \Rightarrow M_z \mid \mathtt{s}(x)\ \mathtt{with}\ h \Rightarrow M_s]]\!]g &= \mathrm{Rec}_\sigma \begin{pmatrix} [\![N]\!]g,\\ [\![M_z]\!]g,\\ \lambda(a,b).\ [\![M_s]\!](g, x \mapsto a, h \mapsto b)\end{pmatrix}\end{aligned}$$

where we define a meta-level recursor as follows:

$$\begin{aligned}\mathrm{Rec}_\sigma &: \mathbb{N} \times [\![\sigma]\!] \times (\mathbb{N} \times [\![\sigma]\!] \to [\![\sigma]\!]) \to [\![\sigma]\!]\\ \mathrm{Rec}_\sigma\,(0, z, s) &= z\\ \mathrm{Rec}_\sigma\,(n+1, z, s) &= s(n, \mathrm{Rec}_\sigma\,(n, z, s))\end{aligned}$$

*2.3.1 **System T**-realizable bars*

Every closed term $\cdot \vdash F : (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}$ determines, via the standard semantics above, an assignment of natural numbers $F \diamond_{\mathbf{T}} \alpha : \mathbb{N}$ to each sequence $\alpha : \mathbb{N} \to \mathbb{N}$:

$$F \diamond_{\mathbf{T}} \alpha \triangleq [\![F]\!](\alpha)$$

Using this notation, we are equipped to define a variation on $\vec{u} \lhd Q$ that expresses realizability in **System T**, which we will write $\vec{u} \lhd_{\mathbf{T}} Q$:

$$(\vec{u} \lhd_{\mathbf{T}} Q) \triangleq \exists F.\ (\cdot \vdash F : (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}) \wedge \forall \alpha : \mathbb{N} \to \mathbb{N}.\ \vec{u} +\!\!+ \mathrm{take}(F \diamond_{\mathbf{T}} \alpha, \alpha) \in Q$$

2.4 Interactive semantics of System T

Martín Escardó pioneered a technique called "effectful forcing" for demonstrating nonconstructive (Brouwerian) principles for the definable functionals of Gödel's

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

System T (Escardó, 2013), including the continuity of functionals $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ and uniform continuity of functionals $(\mathbb{N} \to 2) \to \mathbb{N}$.

The idea of effectful forcing is to give a nonstandard *interactive* model of **System T** in which an element of type `nat` is interpreted not as a single natural number, but as a kind of infinitely branching well-founded tree with natural numbers at the leaves; the branch nodes of the tree represent queries to an oracle (a choice sequence). Escardó used this model to show that definable functionals in **System T** are always continuous; our contribution is to extend Escardó's method to prove that the realizable bars can be tabulated into Brouwer's $(\eta, \digamma)$-trees, establishing a restricted version of the bar theorem.

Intuitively, we would to arrange our interactive model in such a way that the naturals are interpreted quite literally as Brouwer's $(\eta, \digamma)$-trees, but it will be more clear to *first* interpret **System T** into a more flexible shape of tree (which we will call "Escardó dialogues"), in which queries to the choice sequence can be made out of order and repeated. Then, we prove that Brouwer's $(\eta, \digamma)$-trees can be construed as a *normal* form for the Escardó dialogues.

2.4.1 *Escardó dialogues: ideal codes for functionals*

Escardó's dialogue trees provide, for any sets $I, J, A \in \mathbf{Set}$, a representation $\mathbf{ETree}(I, J, A) \in \mathbf{Set}$ of computations of an element of A relative to an *oracle* of type $I \to J$ (i.e., an oracle that receives queries of type I and responds with answers of type J). The Escardó dialogues are generated by the following rules of formation:

$$\boxed{\frac{I, J, A : \mathbf{Set}}{\mathbf{ETree}(I, J, A) : \mathbf{Set}}} \qquad \text{RETURN}\ \frac{a : A}{\eta(a) : \mathbf{ETree}(I, J, A)} \qquad \text{QUERY}\ \frac{i : I \qquad \mathfrak{a} : (J \to \mathbf{ETree}(I, J, A))}{\beta(i; \mathfrak{a}) \in \mathbf{ETree}(I, J, A)}$$

The Escardó dialogues exhibit an endofunctor $\mathbf{Set} \to \mathbf{Set}$ in the following way:

$$\begin{aligned} (\langle\$\rangle) &: (A \to B) \to \mathbf{ETree}(I, J, A) \to \mathbf{ETree}(I, J, B) \\ f \mathbin{\langle\$\rangle} \eta(a) &= \eta(f(a)) \\ f \mathbin{\langle\$\rangle} \beta(i; \mathfrak{a}) &= \beta(i; \lambda j.\, f \mathbin{\langle\$\rangle} \mathfrak{a}(j)) \end{aligned}$$

In fact, the Escardó dialogues also have the structure of a monad, taking $\eta(-)$ as the unit:

$$\begin{aligned} (\gg\!=) &: \mathbf{ETree}(I, J, A) \to (A \to \mathbf{ETree}(I, J, B)) \to \mathbf{ETree}(I, J, B) \\ \eta(a) \gg\!= f &= f(a) \\ \beta(i; \mathfrak{a}) \gg\!= f &= \beta(i; \lambda j.\, \mathfrak{a}(j) \gg\!= f) \end{aligned}$$

The functor and monad laws for $\mathbf{ETree}(I, J, -)$ hold immediately by induction on dialogue trees; in the inductive step, however, we do need function extensionality.

2.4.2 *Running Escardó dialogues*

An Escardó dialogue $\mathfrak{a} : \mathbf{ETree}(I, J, A)$ may be executed on a function $\alpha : I \to J$ to return a result $\mathfrak{a} \diamond_{\mathbf{E}} \alpha : A$ as follows:

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

$$(\diamond_{\mathbf{E}}) : \mathbf{ETree}(I, J, A) \to (I \to J) \to A$$
$$\eta(a) \diamond_{\mathbf{E}} \alpha = a$$
$$\beta(i; \mathfrak{a}) \diamond_{\mathbf{E}} \alpha = \mathfrak{a}(\alpha(i)) \diamond_{\mathbf{E}} \alpha$$

The following two lemmas are due to Escardó (2013).

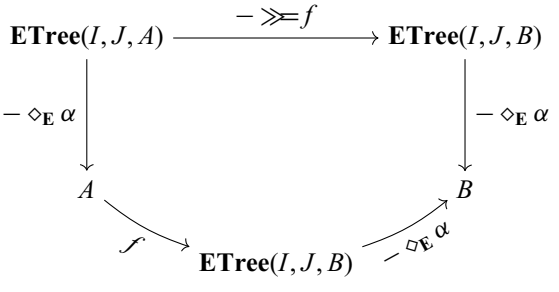
Lemma 2.3 (Naturality of execution)**.** *For any $\alpha : I \to J$, the execution map $- \diamond_{\mathbf{E}} \alpha$ is a natural transformation $\mathbf{ETree}(I, J, -) \to \mathbf{id}_{\mathbf{Set}}$. In other words, for any $f : A \to B$, the following square commutes:*

$$\begin{array}{ccc} \mathbf{ETree}(I,J,A) & \xrightarrow{f\ \langle\$\rangle\ -} & \mathbf{ETree}(I,J,B) \\ \Big\downarrow{\scriptstyle - \diamond_{\mathbf{E}} \alpha} & & \Big\downarrow{\scriptstyle - \diamond_{\mathbf{E}} \alpha} \\ A & \xrightarrow[f]{} & B \end{array} \tag{2.1}$$

Proof. Immediate by induction on the dialogue tree. ∎

The following lemma is needed in Section 2.5 to establish the compatibility of the standard model of **System T** with the dialogue model defined below in Section 2.4.3.

Lemma 2.4 (Execution of Kleisli extension)**.** *Kleisli extension commutes with execution, in the sense that the following diagram commutes for every $f : A \to \mathbf{ETree}(I, J, B)$ and $\alpha : I \to J$:*



(2.2)

Proof. Immediate by induction on the dialogue tree. ∎

*2.4.3 The dialogue model of **System T***

We give an effectful *call-by-name* interpretation of **System T**, in which each type is interpreted as an *algebra* for the dialogue monad $\mathbf{ETree}(\mathbb{N}, \mathbb{N}, -)$, as suggested by one of the anonymous referees. The role of algebras here is the usual for effectful call-by-name semantics: a type is a set equipped with the capability to make queries to an oracle.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

Definition 2.5. *An algebra for the dialogue monad is a set A together with a function* $\mathrm{alg}_A : \mathbf{ETree}(\mathbb{N}, \mathbb{N}, A) \to A$ *satisfying the following conditions:*[9]

1. *For all* $a : A$, *we have* $\mathrm{alg}_A(\boldsymbol{\eta}(a)) = a$.
2. *For all* $m : \mathbf{ETree}(\mathbb{N}, \mathbb{N}, \mathbf{ETree}(\mathbb{N}, \mathbb{N}, A))$, *we have* $i(\mathrm{alg}_A \mathbin{\langle \$ \rangle} m) = \mathrm{alg}_A(\mu(m))$ *where* $\mu(m) = m \gg\!= \lambda x.x$.

Remark 2.6. The purpose of the algebra laws is to ensure that the action is compatible with the monadic structure of Escardó trees. For instance, the first law expresses that a "constant interaction" with the oracle that always returns a is taken to a; the second law ensures that interpreting a nested interaction is the same as interpreting each level of the interaction successively.

There is a forgetful functor U from algebras to sets taking an algebra $\underline{A} = (A, \mathrm{alg}_A)$ to its carrier set $\mathrm{U}(\underline{A}) = A$.

Construction 2.7 (Free algebras)**.** *The forgetful functor has a left adjoint* F *that takes a set A to the* free *algebra* $\mathrm{F}(A)$ *whose carrier is the set of Escardó trees on A.*

$$\mathrm{U}(\mathrm{F}(A)) = \mathbf{ETree}(\mathbb{N}, \mathbb{N}, A) \qquad \begin{aligned} \mathrm{alg}_{\mathrm{F}(A)} &: \mathbf{ETree}(\mathbb{N}, \mathbb{N}, \mathrm{U}(\mathrm{F}(A))) \to \mathrm{U}(\mathrm{F}(A)) \\ \mathrm{alg}_{\mathrm{F}(A)}(m) &= m \gg\!= \lambda x.x \end{aligned}$$

Construction 2.8 (Function algebras)**.** *Given a set A and an algebra $\underline{B}$, we may define the* function algebra $A \to \underline{B}$ *whose carrier set is the function set* $A \to \mathrm{U}(\underline{B})$*; the algebra structure is obtained by applying* alg_B *pointwise.*

$$\mathrm{U}(A \to \underline{B}) = A \to \mathrm{U}(\underline{B}) \qquad \begin{aligned} \mathrm{alg}_{A \to \underline{B}} &: \mathbf{ETree}(\mathbb{N}, \mathbb{N}, \mathrm{U}(A \to \underline{B})) \to \mathrm{U}(A \to \underline{B}) \\ \mathrm{alg}_{A \to \underline{B}}(m) &= \lambda x.\mathrm{alg}_{\underline{B}}((\lambda f.f(x)) \mathbin{\langle \$ \rangle} m) \end{aligned}$$

Construction 2.9 (Dependent product algebras)**.** *Likewise, given a set I and a family of algebras $\underline{B}_{(i \in I)}$, we may form the dependent product algebra $\prod_{i \in I} \underline{B}_i$ in the same way.*

$$\mathrm{U}\left(\prod_{i \in I} \underline{B}_i\right) = \prod_{i \in I} \mathrm{U}(\underline{B}_i) \qquad \begin{aligned} \mathrm{alg}_{\prod_{i \in I} \underline{B}_i} &: \mathbf{ETree}\left(\mathbb{N}, \mathbb{N}, \mathrm{U}\left(\prod_{i \in I} \underline{B}_i\right)\right) \to \mathrm{U}\left(\prod_{i \in I} \underline{B}_i\right) \\ \mathrm{alg}_{\prod_{i \in I} \underline{B}_i}(m) &= \lambda i.\mathrm{alg}_{\underline{B}_i}((\lambda f.f(i)) \mathbin{\langle \$ \rangle} m) \end{aligned}$$

Remark 2.10. In both Constructions 2.8 and 2.9, the verification of the algebra laws requires function extensionality—which we already needed to prove the monad laws for the dialogue monad.

[9] In fact, only the first condition is needed for our main result; in spite of this, it is more natural to speak of algebras for a monad than to speak of algebras for its underlying "pointed endofunctor". Our intention is to identify the most intuitive proof, not the one that makes the minimal assumptions.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

The above is enough to define the interpretation of **System T** contexts and types Γ, σ into algebras $\langle\!\langle \Gamma \rangle\!\rangle, \langle\!\langle \sigma \rangle\!\rangle$ as follows:[10]

$$\begin{aligned}
\langle\!\langle \Gamma \rangle\!\rangle &= \textstyle\prod_{x \in |\Gamma|} \langle\!\langle \Gamma(x) \rangle\!\rangle \\
\langle\!\langle \mathtt{nat} \rangle\!\rangle &= \mathrm{F}(\mathbb{N}) \\
\langle\!\langle \sigma \Rightarrow \tau \rangle\!\rangle &= \mathrm{U}\langle\!\langle \sigma \rangle\!\rangle \to \langle\!\langle \tau \rangle\!\rangle
\end{aligned}$$

We may now define the interpretation of terms into the dialogue model:

$$\begin{aligned}
\langle\!\langle M \rangle\!\rangle &: \mathrm{U}\langle\!\langle \Gamma \rangle\!\rangle \to \mathrm{U}\langle\!\langle \sigma \rangle\!\rangle \\
\langle\!\langle x \rangle\!\rangle \mathfrak{g} &= \mathfrak{g}(x) \\
\langle\!\langle \mathtt{fn}\, x \Rightarrow M \rangle\!\rangle \mathfrak{g} &= \lambda \mathfrak{a}. \langle\!\langle M \rangle\!\rangle (\mathfrak{g}, x \mapsto \mathfrak{a}) \\
\langle\!\langle M \mathbin{@_\sigma} N \rangle\!\rangle \mathfrak{g} &= \big(\langle\!\langle M \rangle\!\rangle \mathfrak{g}\big)\big(\langle\!\langle N \rangle\!\rangle \mathfrak{g}\big) \\
\langle\!\langle \mathtt{z} \rangle\!\rangle \mathfrak{g} &= \boldsymbol{\eta}(0) \\
\langle\!\langle \mathtt{s}(M) \rangle\!\rangle \mathfrak{g} &= (1 + -) \mathbin{\langle \$ \rangle} \langle\!\langle M \rangle\!\rangle \mathfrak{g} \\
\langle\!\langle \mathtt{rec}\ N\ [\mathtt{z} \Rightarrow M_z \mid \mathtt{s}(x)\ \mathtt{with}\ h \Rightarrow M_s] \rangle\!\rangle \mathfrak{g} &= \mathrm{alg}_{\langle\!\langle \sigma \rangle\!\rangle}(\langle\!\langle N \rangle\!\rangle \gg\!= R) \\
\text{where } R(n) = \mathrm{Rec}_{\mathrm{U}\langle\!\langle \sigma \rangle\!\rangle} &\begin{pmatrix} n, \\ \boldsymbol{\eta}(\langle\!\langle M_z \rangle\!\rangle \mathfrak{g}), \\ \lambda(\mathfrak{a}, \mathfrak{b}).\ \boldsymbol{\eta}(\langle\!\langle M_s \rangle\!\rangle (\mathfrak{g}, x \mapsto \boldsymbol{\eta}(a), h \mapsto \mathrm{alg}_{\langle\!\langle \sigma \rangle\!\rangle} \mathfrak{b})) \end{pmatrix}
\end{aligned}$$

2.4.3.1 Comparison with Escardó's interpretation. Our dialogue interpretation of **System T** differs from that of Escardó, which fails to treat the recursor compositionally; Escardó interprets every type σ as a set $\langle\!\langle \sigma \rangle\!\rangle$, and then to interpret the recursor he must define a *lifting* of the Kleisli extension to higher type by recursion on type structure:

$$\begin{aligned}
\langle \gg\!= \rangle_\sigma &: \mathbf{ETree}(\mathbb{N}, \mathbb{N}, X) \to (X \to \langle\!\langle \sigma \rangle\!\rangle) \to \langle\!\langle \sigma \rangle\!\rangle \\
m \mathbin{\langle \gg\!= \rangle_{\mathtt{nat}}} f &= m \gg\!= f \\
m \mathbin{\langle \gg\!= \rangle_{\sigma \Rightarrow \tau}} f &= \lambda s : \langle\!\langle \sigma \rangle\!\rangle . m \mathbin{\langle \gg\!= \rangle_\tau} \lambda x : X . f(x, s)
\end{aligned}$$

Then recursion at σ is interpreted by taking the lifted Kleisli extension $\langle \gg\!= \rangle_\sigma$ of the recursor $R(-)$ defined above. While this style of interpretation works fine, it is not compositional: the meaning of an expression should be determined in directly from the meaning of its subexpressions, but here we resorted to assigning a different meaning for the recursor at each type. Levy (2006) already suggests to resolve this problem by means of the algebra translation in Section 2.1 of his monograph on call-by-push-value, which leads to the more elegant account of recursion presented in this paper.

2.4.4 The generic point

In addition to the standard natural numbers (which are interpreted as "constant" trees, i.e., trees whose leaves are all labeled by the same numeral), the dialogue model of **System T** contains *nonstandard* natural numbers that do not lie in the image of the interpretation function).

[10] It is worth noting that our interpretation of the natural numbers as the free algebra on the set $\mathbb{N}$ is somewhat bizarre from the standpoint of call-by-name semantics, where one expects the successor nodes to encapsulate an effect. However, as pointed out by one of the anonymous referees, one seems to disrupt the main result by employing the call-by-name natural numbers.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

As a consequence of having nonstandard numbers, the dialogue model also has nonstandard sequences; the most important of these sequences is called the *generic point* $\mathbf{generic} : \mathrm{U}\langle\!\langle \mathtt{nat} \Rightarrow \mathtt{nat} \rangle\!\rangle$, a special nonstandard sequence that can be used to probe a higher order function for its intensional structure, providing the main ingredient for the tabulation of higher order functions as trees.

$$\mathbf{generic} : \mathrm{U}\langle\!\langle \mathtt{nat} \Rightarrow \mathtt{nat} \rangle\!\rangle$$
$$\mathbf{generic} = \lambda \mathfrak{n}.\ \mathfrak{n} \gg\!= \lambda x.\ \beta(x; \boldsymbol{\eta}(-))$$

Intuitively, by applying the dialogue interpretation of a functional $\cdot \vdash F : (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}$ to this generic point, we get a dialogue tree $\langle\!\langle F \rangle\!\rangle(\mathbf{generic}) : \mathbf{ETree}(\mathbb{N}, \mathbb{N}, \mathbb{N})$ that is precisely the trace of F's calls to its argument. Then, assuming that F witnesses $\vec{u} \lhd_{\mathbf{T}} Q$, we can compute the derivation of $\vec{u} \blacktriangleleft Q$ by induction on this trace.

Lemma 2.11. *The generic point commutes with dialogue execution in the sense that the following diagram commutes for all* $\alpha : \mathbb{N} \to \mathbb{N}$*:*

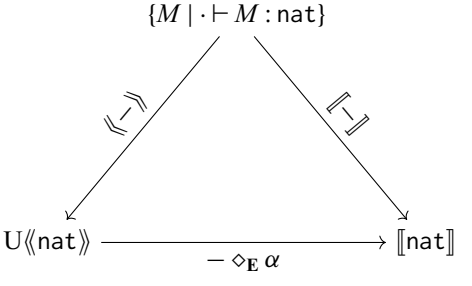
$$\begin{array}{ccc} \mathrm{U}\langle\!\langle \mathtt{nat} \rangle\!\rangle & \xrightarrow{\mathbf{generic}} & \mathrm{U}\langle\!\langle \mathtt{nat} \rangle\!\rangle \\ {\scriptstyle - \diamond_{\mathbf{E}} \alpha}\downarrow & & \downarrow{\scriptstyle - \diamond_{\mathbf{E}} \alpha} \\ \mathbb{N} & \xrightarrow[\alpha]{} & \mathbb{N} \end{array} \tag{2.3}$$

Proof. Fixing $\mathfrak{n} : \mathrm{U}\langle\!\langle \mathtt{nat} \rangle\!\rangle$, one obtains $\alpha(\mathfrak{n} \diamond_{\mathbf{E}} \alpha) = \mathbf{generic}(\mathfrak{n}) \diamond_{\mathbf{E}} \alpha$ immediately by induction on the dialogue tree $\mathfrak{n}$. ■

2.5 Compatibility of standard and interactive semantics

In order to use the generic point to tabulate a higher order function $\cdot \vdash F : (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}$ into a dialogue tree, we will need to establish that the interactive semantics $\langle\!\langle \ldots \rangle\!\rangle$ and the standard semantics $[\![\ldots]\!]$ are compatible at all types, in a sense that we must define. In essence, we will be using the fact that the dialogues in the image of the interpretation of **System T** encode pure functions, even though they coexist with "impure" functions.

For intuition, it is worth considering what it would mean for the two interpretations to be compatible at the base type $\mathtt{nat}$. In this case, we would want to ensure for any term $\cdot \vdash M : \mathtt{nat}$, that $[\![M]\!]$ is the same as $\langle\!\langle M \rangle\!\rangle \diamond_{\mathbf{E}} \alpha$ for all sequences $\alpha : \mathbb{N} \to \mathbb{N}$:



(2.4)

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

In order to prove the above, however, we cannot simply proceed by induction on the syntax of **System T**—the induction hypothesis will not be strong enough when we pass underneath a binder; the compatibility condition we have stated in in fact a closure property of the entire language that must be established at all higher types simultaneously, a situation that calls for *logical relations* (Tait, 1967).

Tait's method of logical relations is a tool to endow a property of terms at base type with a hereditary action on all higher types. In our case, we will extend the property of "having compatible standard and dialogue interpretations" from $\mathtt{nat}$ to all types σ, and then show that the resulting structure is closed under all the constructors of **System T**.

2.5.1 A logical relation between the two models

We begin by defining the main constituents of a logical relation between $[\![-]\!]$ and $\mathrm{U}\langle\!\langle - \rangle\!\rangle$. For each point $\alpha : \mathbb{N} \to \mathbb{N}$, we will define an interpretation of each type σ as a relation between $[\![\sigma]\!]$ and $\mathrm{U}\langle\!\langle \sigma \rangle\!\rangle$ that expresses the *hereditary* compatibility of the standard interpretation with the execution of the dialogue interpretation at α.

.iven $s : [\![\sigma]\!]$ and $\mathfrak{s} : \mathrm{U}\langle\!\langle \sigma \rangle\!\rangle$, we will define the following predicate by recursion on the type σ:

$$\boxed{\alpha \vDash \sigma \ni s \sim \mathfrak{s}}$$

$$\frac{n = \mathfrak{n} \diamond_{\mathbf{E}} \alpha}{\alpha \vDash \mathtt{nat} \ni n \sim \mathfrak{n}} \qquad \frac{\forall s : [\![\sigma]\!], \mathfrak{s} : \mathrm{U}\langle\!\langle \sigma \rangle\!\rangle.\ \alpha \vDash \sigma \ni s \sim \mathfrak{s} \implies \alpha \vDash \tau \ni f(s) \sim \mathfrak{f}(\mathfrak{s})}{\alpha \vDash \sigma \Rightarrow \tau \ni f \sim \mathfrak{f}}$$

Likewise, we may define an analogous relation on contexts and environments as follows, given $g : [\![\Gamma]\!]$ and $\mathfrak{g} : \mathrm{U}\langle\!\langle \Gamma \rangle\!\rangle$:

$$\frac{\forall x \in |\Gamma|.\ \alpha \vDash \Gamma(x) \ni g(x) \sim \mathfrak{g}(x)}{\alpha \vDash \Gamma \ni g \sim \mathfrak{g}}$$

Theorem 2.12 (Fundamental theorem)**.** *Fix a point $\alpha : \mathbb{N} \to \mathbb{N}$ and let $\Gamma \vdash M : \sigma$ be a term of **System T**; then for all $g : [\![\Gamma]\!]$ and $\mathfrak{g} : \mathrm{U}\langle\!\langle \Gamma \rangle\!\rangle$ such that $\alpha \vDash \Gamma \ni g \sim \mathfrak{g}$, we have $\alpha \vDash \sigma \ni [\![M]\!]g \sim \langle\!\langle M \rangle\!\rangle \mathfrak{g}$.*

Proof. By induction on $\Gamma \vdash M : \sigma$, using the monad axioms, the algebra axioms, and Lemmas 2.3 and 2.4. ■

2.6 Tabulating realizable bars into Escardó dialogues

Fixing a monotone subset $Q \subseteq \mathbf{list}(\mathbb{N})$, our goal has been to show that $\vec{u} \blacktriangleleft Q$ follows from $\vec{u} \lhd_{\mathbf{T}} Q$, recalling the definition of the latter:

$$(\vec{u} \lhd_{\mathbf{T}} Q) \triangleq \exists(\cdot \vdash f : (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}).\ \forall \alpha : \mathbb{N} \to \mathbb{N}.\ \vec{u} +\!\!+ \mathrm{take}(f \diamond_{\mathbf{T}} \alpha, \alpha) \in Q$$

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

Using the results of the previous section, we are now equipped to prove an intermediate theorem that tabulates any **System T**-realizable bar into an *Escardó* dialogue, in the sense that $\vec{u} \lhd_{\mathbf{T}} Q$ implies $\vec{u} \blacktriangleleft_{\mathbf{E}} Q$, defined below:

$$(\vec{u} \blacktriangleleft_{\mathbf{E}} Q) \triangleq \exists \mathfrak{n} : \mathbf{ETree}(\mathbb{N}, \mathbb{N}, \mathbb{N}).\ \forall \alpha : \mathbb{N} \to \mathbb{N}.\ \vec{u} +\!\!+ \mathrm{take}(\mathfrak{n} \diamond_{\mathbf{E}} \alpha, \alpha) \in Q$$

It is here that we make use of the generic point $\mathbf{generic} : \mathrm{U}\langle\!\langle \mathtt{nat} \Rightarrow \mathtt{nat} \rangle\!\rangle$ in a critical way.

Theorem 2.13 (Tabulation of realizable bars)**.** *If* $\vec{u} \lhd_{\mathbf{T}} Q$ *then* $\vec{u} \blacktriangleleft_{\mathbf{E}} Q$.

Proof. We abbreviate $\sigma \triangleq (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}$. Suppose $\vec{u} \lhd_{\mathbf{T}} Q$, i.e., we have some term $\cdot \vdash F : \sigma$ such that $\vec{u} +\!\!+ \mathrm{take}(F \diamond_{\mathbf{T}} \alpha, \alpha) \in Q$ for all $\alpha : \mathbb{N} \to \mathbb{N}$. Interpreting F into the dialogue model, we have $\mathfrak{F} \triangleq \langle\!\langle F \rangle\!\rangle : \mathrm{U}\langle\!\langle \sigma \rangle\!\rangle$; instantiating with the generic point, we therefore obtain a single Escardó dialogue $\mathfrak{F}(\mathbf{generic}) : \mathrm{U}\langle\!\langle \mathtt{nat} \rangle\!\rangle$.

Using $\mathfrak{F}(\mathbf{generic})$ as our witness to $\vec{u} \blacktriangleleft_{\mathbf{E}} Q$, we fix $\alpha : \mathbb{N} \to \mathbb{N}$ to verify that Q contains the list $\vec{u} +\!\!+ \mathrm{take}(\mathfrak{F}(\mathbf{generic}) \diamond_{\mathbf{E}} \alpha, \alpha)$. Recalling our assumption, it would suffice to verify that $F \diamond_{\mathbf{T}} \alpha = \mathfrak{F}(\mathbf{generic}) \diamond_{\mathbf{E}} \alpha$.

By the fundamental theorem (Theorem 2.12), we already have $\alpha \vDash \sigma \ni [\![F]\!] \sim \mathfrak{F}$. Unfolding the logical relation explicitly this amounts to the fact that $[\![F]\!](\beta) = \mathfrak{F}(\mathfrak{b}) \diamond_{\mathbf{E}} \alpha$ for all $\alpha \vDash \mathtt{nat} \Rightarrow \mathtt{nat} \ni \beta \sim \mathfrak{b}$; we note that $[\![F]\!](\beta) = F \diamond_{\mathbf{T}} \beta$. Choosing $\beta = \alpha$ and $\mathfrak{b} = \mathbf{generic}$, it therefore remains only to show that $\alpha \vDash \mathtt{nat} \Rightarrow \mathtt{nat} \ni \alpha \sim \mathbf{generic}$. We fix $\alpha \vDash \mathtt{nat} \ni n \sim \mathfrak{n}$ to show that $\alpha \vDash \mathtt{nat} \ni \alpha(n) \sim \mathbf{generic}(\mathfrak{n})$, which is the same as to say $\alpha(n) = \mathbf{generic}(\mathfrak{n}) \diamond_{\mathbf{E}} \alpha$. But this is immediate by Lemma 2.11. ∎

3 Brouwerian dialogues: a normal form for Escardó Dialogues

In the previous section, we showed that a **System T**-realizable bar can be tabulated into an Escardó dialogue: in other words, $\vec{u} \blacktriangleleft_{\mathbf{E}} Q$ follows from $\vec{u} \lhd_{\mathbf{T}} Q$. In this section, we bridge the gap between the Escardó dialogues and Brouwer's $(\boldsymbol{\eta}, \digamma)$-trees, by proving that the latter express a *normal form* for the former with respect to permutation, omission, and repetition of queries to the choice sequence.

3.1 Ephemerality, the essence of Brouwerian dialogues

The content of Brouwer's purported (but failed) proof of his bar thesis was to assert that one can tabulate the evidence for barhood into a well-founded mental construction (van Atten, 2004; Dummett, 2000); Escardó's translation of **System T** terms into dialogue trees is essentially a formalization of Brouwer's insight.

However, Escardó's dialogues differ from Brouwer's mental constructions of barhood (which are captured precisely by the judgment $\vec{u} \blacktriangleleft Q$) in one crucial respect: whereas Escardó's trees branch on an arbitrary query to the ambient choice sequence, queries in Brouwer's mental constructions must be made in order, i.e., with respect to the current moment in ideal time; as such, the Brouwerian dialogues are *ephemeral*—with each query, the head of the ambient choice sequence is consumed and the remainder of the dialogue is interpreted with respect to the tail of the choice sequence.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

Our task, then, will be to normalize Escardó's dialogues into Brouwer's ephemeral mental constructions, and then show how to massage these into a derivation of $\vec{u} \blacktriangleleft Q$. Below we define the set of Brouwerian dialogues $\mathbf{BTree}(J, A)$ (coding functionals $(\mathbb{N} \to J) \to A$) as the least set closed under the rules below:

$$\boxed{\frac{J, A : \mathbf{Set}}{\mathbf{BTree}(J, A) : \mathbf{Set}}} \qquad \text{SPIT}\ \frac{a : A}{\eta(a) : \mathbf{BTree}(J, A)} \qquad \text{BITE}\ \frac{\mathbf{a} : (J \to \mathbf{BTree}(J, A))}{\digamma(\mathbf{a}) : \mathbf{BTree}(J, A)}$$

Just as we showed how to execute an Escardó dialogue against a choice sequence, we can do the same for the Brouwerian (ephemeral) version. For $\mathbf{a} : \mathbf{BTree}(J, A)$ and $\alpha : \mathbb{N} \to J$, we define $\mathbf{a} \diamond_{\mathbf{B}} \alpha : A$ by recursion on $\mathbf{a}$ as follows:

$$(\diamond_{\mathbf{B}}) : \mathbf{BTree}(J, A) \to (\mathbb{N} \to J) \to A$$
$$\eta(a) \diamond_{\mathbf{B}} \alpha = a$$
$$\digamma(\mathbf{a}) \diamond_{\mathbf{B}} \alpha = \mathbf{a}(\mathrm{head}(\alpha)) \diamond_{\mathbf{B}} \mathrm{tail}(\alpha)$$

3.1.1 Barhood relative to Brouwerian dialogues

Just as we have defined $\vec{u} \blacktriangleleft_{\mathbf{E}} Q$ as a notion of barhood witnessed by an Escardó dialogue, we now do the same for Brouwerian dialogues, recalling the definitions of barhood we have given so far:

$$(\vec{u} \blacktriangleleft_{\mathbf{T}} Q) \triangleq \exists(\cdot \vdash f : (\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}).\ \forall \alpha : \mathbb{N} \to \mathbb{N}.\ \vec{u} +\!\!+ \mathrm{take}(f \diamond_{\mathbf{T}} \alpha, \alpha) \in Q$$
$$(\vec{u} \blacktriangleleft_{\mathbf{E}} Q) \triangleq \exists \mathfrak{n} : \mathbf{ETree}(\mathbb{N}, \mathbb{N}, \mathbb{N}).\ \forall \alpha : \mathbb{N} \to \mathbb{N}.\ \vec{u} +\!\!+ \mathrm{take}(\mathfrak{n} \diamond_{\mathbf{E}} \alpha, \alpha) \in Q$$
$$(\vec{u} \blacktriangleleft_{\mathbf{B}} Q) \triangleq \exists \mathbf{n} : \mathbf{BTree}(\mathbb{N}, \mathbb{N}).\ \forall \alpha : \mathbb{N} \to \mathbb{N}.\ \vec{u} +\!\!+ \mathrm{take}(\mathbf{n} \diamond_{\mathbf{B}} \alpha, \alpha) \in Q$$

The Brouwerian dialogues that we have defined are nothing but representations of proofs of $\vec{u} \blacktriangleleft Q$, as the following lemma shows.

Lemma 3.1. *For any monotone subset $Q \subseteq \mathbf{list}(\mathbb{N})$ and basic observation $\vec{u} : \mathbf{list}(\mathbb{N})$, if $\vec{u} \blacktriangleleft_{\mathbf{B}} Q$ then $\vec{u} \blacktriangleleft Q$.*

Proof. Fixing Q and $\vec{u}$ such that $\vec{u} \blacktriangleleft_{\mathbf{B}} Q$, we therefore have some Brouwerian dialogue $\mathbf{n} : \mathbf{BTree}(\mathbb{N}, \mathbb{N})$ such that for all sequences α, we have $\vec{u} +\!\!+ \mathrm{take}(\mathbf{n} \diamond_{\mathbf{B}} \alpha, \alpha) \in Q$. We proceed by induction on $\mathbf{n}$ to prove $\vec{u} \blacktriangleleft Q$, using the fact that Q is monotone. ■

3.2 Normalizing Escardó dialogues into Brouwerian dialogues

We wish to normalize elements of $\mathbf{ETree}(\mathbb{N}, J, A)$ (Escardó dialogues) into elements of $\mathbf{BTree}(J, A)$ (Brouwerian dialogues) in a way that commutes with execution. To do so, we design an inductive/proof-theoretic characterization of the normalizable Escardó dialogues, and then show that all Escardó dialogues can be coded as such. This yields a constructive and structurally recursive normalization algorithm. To this end, we define below two mutually inductive forms of judgment, whose rules are given in Figure 3:

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

$$\boxed{\vec{u} \Vdash \mathfrak{a} \rightsquigarrow \mathbf{a} \text{ presupposing } \vec{u} : \mathbf{list}(J),\ \mathfrak{a} : \mathbf{ETree}(\mathbb{N}, J, A),\ \mathbf{a} : \mathbf{BTree}(J, A)}$$

$$\boxed{\vec{u} \mid i \Vdash \mathfrak{a} \ll \vec{v} \rightsquigarrow \mathbf{a} \text{ presupposing } \vec{u}, \vec{v} : \mathbf{list}(J),\ i : \mathbb{N},\ \mathfrak{a} : J \to \mathbf{ETree}(\mathbb{N}, J, A),\ \mathbf{a} : \mathbf{BTree}(J, A)}$$

$$\frac{}{\vec{u} \Vdash \eta(a) \rightsquigarrow \eta(a)} \qquad \frac{\vec{u} \mid i \Vdash \mathfrak{a} \ll \vec{u} \rightsquigarrow \mathbf{a}}{\vec{u} \Vdash \beta(i; \mathfrak{a}) \rightsquigarrow \mathbf{a}} \qquad \frac{\vec{u} \Vdash \mathfrak{a}(j) \rightsquigarrow \mathbf{a}}{\vec{u} \mid 0 \Vdash \mathfrak{a} \ll j :: \vec{v} \rightsquigarrow \mathbf{a}}$$

$$\frac{\vec{u} \mid i \Vdash \mathfrak{a} \ll \vec{v} \rightsquigarrow \mathbf{a}}{\vec{u} \mid i+1 \Vdash \mathfrak{a} \ll j :: \vec{v} \rightsquigarrow \mathbf{a}} \qquad \frac{\forall j : J.\ \vec{u} \mathbin{\hat{}} j \Vdash \mathfrak{a}(j) \rightsquigarrow \mathbf{a}(j)}{\vec{u} \mid 0 \Vdash \mathfrak{a} \ll \langle\rangle \rightsquigarrow \digamma(\mathbf{a})} \qquad \frac{\forall j : J.\ \vec{u} \mathbin{\hat{}} j \mid i \Vdash \mathfrak{a} \ll \langle\rangle \rightsquigarrow \mathbf{a}(j)}{\vec{u} \mid i+1 \Vdash \mathfrak{a} \ll \langle\rangle \rightsquigarrow \digamma(\mathbf{a})}$$

Fig. 3. Rules for dialogue normalization.

```
datatype ('i, 'j, 'a) etree =
   RET of 'a
 | QUERY of 'i * ('j → ('i, 'j, 'a) etree)

datatype ('j, 'a) btree =
   SPIT of 'a
 | BITE of ('j → ('j, 'a) btree)

fun norm us (RET a) = SPIT a
  | norm us (QUERY (i, k)) = normQ us i k us

and normQ us Z k [] = BITE (fn u ⇒ norm (us @ [u]) (k u))
  | normQ us Z k (v :: _) = norm us k v
  | normQ us (S i) (_ :: vs) = normQ us i k vs
  | normQ _ _ _ _ = raise Subscript
```

Fig. 4. A Standard ML implementation of the dialogue normalization algorithm whose graph is specified in Figure 3.

1. $\vec{u} \Vdash \mathfrak{a} \rightsquigarrow \mathbf{a}$, presupposing $\vec{u} : \mathbf{list}(J)$ and $\mathfrak{a} : \mathbf{ETree}(\mathbb{N}, J, A)$, and guaranteeing $\mathbf{a} : \mathbf{BTree}(J, A)$, means that the Escardó dialogue $\mathfrak{a}$ normalizes to the Brouwerian dialogue $\mathbf{a}$.
2. $\vec{u} \mid i \Vdash \mathfrak{a} \ll \vec{v} \rightsquigarrow \mathbf{a}$ presupposes $\vec{u}, \vec{v} : \mathbf{list}(J)$, $i : \mathbb{N}$ and $\mathfrak{a} : J \to \mathbf{ETree}(\mathbb{N}, J, A)$, and guarantees $\mathbf{a} : \mathbf{BTree}(J, A)$.

For the sake of intuition, we provide an implementation of the algorithm in Standard ML in Figure. 4; the mathematical version given in this section can be seen as a specification and termination proof for the Standard ML program.

3.2.1 Explanation of algorithm

Normalization occurs with respect to a node $\vec{u}$ that is extended every time a $\digamma$ node is inserted; $\vec{u}$ should be thought of as the "current known prefix of the choice sequence". To

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

normalize an Escardó dialogue $\mathfrak{a}$ over a node $\vec{u}$, one proceeds by case: if $\mathfrak{a} = \boldsymbol{\eta}(a)$ then it is already normal; on the other hand, if $\mathfrak{a} = \beta(i; \mathfrak{b})$, then it is necessary to find out what the ith element of the choice sequence is.

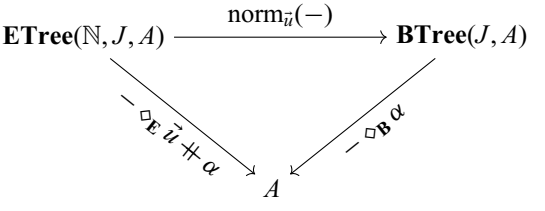
If $i < \mathrm{len}(\vec{u})$, then we have already "cached" this query in $\vec{u}_i$, and we therefore proceed by normalizing $\mathfrak{b}(\vec{u}_i)$; on the other hand, if $i \geq \mathrm{len}(\vec{u})$, we must continue to insert $\digamma$-queries until we have information about the ith element of the choice sequence. Locating "cached" information and inserting new queries is the role of the $\vec{u} \mid i \Vdash \mathfrak{a} \ll \vec{v} \rightsquigarrow \mathbf{a}$ judgment.

Lemma 3.2. *The inductive characterization of normalization $\vec{u} \Vdash \mathfrak{a} \rightsquigarrow \boldsymbol{a}$ in Figure 3 is functional, i.e., for any $\vec{u} : \mathbf{list}(J)$ and $\mathfrak{a} : \mathbf{ETree}(\mathbb{N}, J, A)$ there is a unique $\boldsymbol{a} : \mathbf{BTree}(J, A)$ such that $\vec{u} \Vdash \mathfrak{a} \rightsquigarrow \boldsymbol{a}$.*

Proof. In fact, we must simultaneously establish the functionality of $\vec{u} \Vdash \mathfrak{a} \rightsquigarrow \mathbf{a}$ and $\vec{u} \mid i \Vdash \mathfrak{b} \ll \vec{v} \rightsquigarrow \mathbf{b}$. The existence part of functionality follows by simultaneous induction on $\mathfrak{a}$, $\vec{v}$, and i; the uniqueness part follows immediately by inspection of the generating rules. ■

Corollary 3.3 (Normalization function)**.** *We have a structurally recursive function* $\mathrm{norm}_{\vec{u}}(\mathfrak{a})$ *such that for all* $\vec{u} : \mathbf{list}(J)$ *and* $\mathfrak{a} : \mathbf{ETree}(\mathbb{N}, Y, Z)$, $\vec{u} \Vdash \mathfrak{a} \rightsquigarrow \mathrm{norm}_{\vec{u}}(\mathfrak{a})$.

Lemma 3.4 (Coherence of normalization and execution)**.** *Execution of Brouwerian dialogues is compatible with normalization, as defined in Corollary 3.3; to be precise, the following diagram commutes for all* $\vec{u} : \mathbf{list}(J)$ *and* $\alpha : \mathbb{N} \to J$.



(3.1)

When $\vec{u} = \langle\rangle$, *this becomes the statement that* $\mathfrak{a} \diamond_{\mathbf{E}} \alpha = \mathrm{norm}_{\langle\rangle}(\mathfrak{a}) \diamond_{\mathbf{B}} \alpha$ *for any dialogue* $\mathfrak{a}$.

Proof. By induction on the graph of the normalization function. ■

4 Validity of the bar thesis for T-realizable bars

We obtain the main theorem of this paper as a corollary to the constructions of the past several sections.

The bar thesis for **System T** states that for any monotone set $Q \subseteq \mathbf{list}(\mathbb{N})$ of nodes, the inductive definition of barhood is complete in the sense that we can conclude $\langle\rangle \blacktriangleleft Q$ from $\langle\rangle \lhd_{\mathbf{T}} Q$.

Theorem 4.1 (The bar thesis for realizable bars)**.** *Brouwer's bar thesis holds for bars that are realizable in* ***System T****: for any monotone subset* $Q \subseteq \mathbf{list}(\mathbb{N})$, *if* $\langle\rangle \lhd_{\mathbf{T}} Q$ *then* $\langle\rangle \blacktriangleleft Q$.

Proof. Supposing $\langle\rangle \lhd_{\mathbf{T}} Q$, we need to construct $\langle\rangle \blacktriangleleft Q$; by Lemma 3.1, it is enough to show that $\langle\rangle \blacktriangleleft_{\mathbf{B}} Q$, i.e., exhibit some $\mathbf{n} : \mathbf{BTree}(\mathbb{N}, \mathbb{N})$ such that for all $\alpha : \mathbb{N} \to \mathbb{N}$, we

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

have take($\mathbf{n} \diamond_{\mathbf{B}} \alpha, \alpha) \in Q$. On the other hand, by Theorem 2.13 we already have $\langle\rangle \blacktriangleleft_{\mathbf{E}} Q$, from which we obtain some $\mathfrak{n} : \mathbf{ETree}(\mathbb{N}, \mathbb{N}, \mathbb{N})$ such that for all $\alpha : \mathbb{N} \to \mathbb{N}$, we have take($\mathfrak{n} \diamond_{\mathbf{E}} \alpha, \alpha) \in Q$. Therefore, choosing $\mathbf{n} = \mathrm{norm}_{\langle\rangle}(\mathfrak{n})$, all that remains is check that take($\mathrm{norm}_{\langle\rangle}(\mathfrak{n}) \diamond_{\mathbf{B}} \alpha, \alpha) = \mathrm{take}(\mathfrak{n} \diamond_{\mathbf{E}} \alpha, \alpha)$. But $\mathrm{norm}_{\langle\rangle}(\mathfrak{n}) \diamond_{\mathbf{B}} \alpha = \mathfrak{n} \diamond_{\mathbf{E}} \alpha$ by Lemma 3.4. ■

5 Discussion and related work

5.1 The bar induction principle

The essence of Brouwer's bar thesis lies in the tabulation of a proof of $\vec{u} \triangleleft Q$ into a proof of $\vec{u} \blacktriangleleft Q$. In constructive metamathematics, however, an equivalent presentation of the Bar Thesis as an induction principle is usually considered:

Proposition 5.1 (Monotone Bar Induction)**.** *For any subset $R \in \mathbf{list}(\mathbb{N})$ and a monotone bar $\langle\rangle \triangleleft Q$, we can conclude $\langle\rangle \in R$ from the following conditions:*

1. Base case. *R contains the bar, i.e., $Q \subseteq R$.*
2. Inductive step. *R is inductive, i.e., $\vec{u} \in R$ follows from $\forall x.\, \vec{u}\,{}^\frown x \in R$.*

To prove the equivalence of Proposition 5.1 with our formulation of Brouwer's bar thesis on monotone bars (Proposition 1.12), we begin by verifying that it suffices to consider unqualified barhood of the root node $\langle\rangle$. In this section we write $\vec{v} \preceq \vec{u}$ to mean that $\vec{u} = \vec{v} +\!\!+ \vec{w}$ for some $\vec{w}$; this notation is the reverse of Dummett's (2000), but we find it more concrete.

Lemma 5.2 (It suffices to consider the root)**.** *If $\langle\rangle \triangleleft Q \implies \langle\rangle \blacktriangleleft Q$ for all monotone Q, then $\vec{u} \triangleleft Q \implies \vec{u} \blacktriangleleft Q$ for all $\vec{u}$ and monotone Q.*

Proof. Suppose that $\langle\rangle \triangleleft Q \implies \langle\rangle \blacktriangleleft Q$ for all monotone Q. Fixing monotone Q' such that $\vec{u} \triangleleft Q'$, we need to show that $\vec{u} \blacktriangleleft Q'$. Letting $Q = \{\vec{v} \mid \vec{u} \preceq \vec{v} \implies \vec{v} \in Q'\}$, we observe that $\langle\rangle \triangleleft Q$ immediately, whence $\langle\rangle \blacktriangleleft Q$ follows by assumption. To see that $\vec{u} \blacktriangleleft Q'$ follows from $\langle\rangle \blacktriangleleft Q$, we observe that the latter can be obtained immediately as a subtree of the former. ■

Theorem 5.3. *Monotone Bar Induction (Proposition 5.1) is equivalent to Brouwer's Thesis on monotone bars (Proposition 1.12).*

Proof. ($\Rightarrow$) Suppose that Proposition 5.1 holds; by Lemma 5.2, it suffices to show that $\langle\rangle \blacktriangleleft Q$ follows from $\langle\rangle \triangleleft Q$. We choose a *motive of bar induction* based on our goal, namely $R \triangleq \{\vec{u} \mid \vec{u} \blacktriangleleft Q\}$.

1. *Base case.* Fixing $\vec{u} \in Q$, we need to see that $\vec{u} \blacktriangleleft Q$; this is just the $\boldsymbol{\eta}$-inference.
2. *Inductive step.* Suppose that $\vec{u}\,{}^\frown x \blacktriangleleft Q$ for all x; we conclude $\vec{u} \blacktriangleleft Q$ by the $\boldsymbol{\mu}$-inference.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

($\Leftarrow$) Suppose that Proposition 1.12 holds. Fix a monotone bar $\langle\rangle \triangleleft Q$ and a subset $R \supseteq Q$ such that $\vec{u} \in R$ follows from $\forall x.\ \vec{u}\,{}^\frown x \in R$ for all $\vec{u}$, to show that $\langle\rangle \in \vec{R}$. By Proposition 1.12, we have $\langle\rangle \blacktriangleleft Q$; we proceed by induction, generalizing to conclude $\vec{v} \in R$ from $\vec{v} \blacktriangleleft Q$ for all $\vec{v}$:

1. η-*inference.* If $\vec{v} \in Q$, then $\langle\rangle \in R$ because $Q \subseteq R$.
2. F-*inference.* Suppose that $\vec{v}\,{}^\frown x \blacktriangleleft Q$ for all x; by induction, we have $\vec{v}\,{}^\frown x \in R$ for all x, whence by assumption we have $\vec{v} \in R$. ∎

5.2 Monotonicity, decidability, and continuity

We have focused exclusively on *monotone* subsets $Q \subseteq \mathbf{list}(\mathbb{N})$, i.e., collections of basic observations that are closed under extension; intuitively, such a bar is one that cannot be "escaped". However, another variant of the bar thesis that concerns *decidable* subsets has received more attention in the literature. A subset $Q \subseteq \mathbf{list}(\mathbb{N})$ is decidable when $\forall \vec{u}.\ \vec{u} \in Q \vee \vec{u} \notin Q$ holds; of course, this is classically true for every subset Q, but in constructive logic, it doesn't necessarily hold for all Q.

The bar induction principle for decidable bars is weaker than monotone bar induction (Proposition 5.1); in essence, this is because any decidable bar can be completed into a monotone bar (Dummett, 2000).

Lemma 5.4. *Any decidable bar $\langle\rangle \triangleleft Q$ can be freely completed into a monotone decidable bar $\langle\rangle \triangleleft Q^\natural$ with $Q \subseteq Q^\natural$.*

Proof. The inclusion of the *discrete* poset of lists of natural numbers into the poset of lists with the approximation ordering $\vec{u} +\!\!+ \vec{v} \succeq \vec{u}$ induces a contravariant reindexing from monotone subsets into general subsets. This reindexing has both left and right adjoints, by Kan extension. In particular, the left adjoint takes a general subset Q and transforms it into a monotone subset $Q^\natural \triangleq \{\vec{u} \mid \exists \vec{v} \preceq \vec{u}.\ \vec{v} \in Q\}$.

First we observe that the transformation described above preserves the decidability of the bar: to decide $\vec{u} \in Q^\natural$, one simply decides $\vec{v} \in Q$ for as many prefixes $\vec{v} \preceq \vec{u}$ as necessary. Second, we check that $\langle\rangle \triangleleft Q^\natural$ assuming $\langle\rangle \triangleleft Q$. Fixing $\alpha : \mathbb{N} \to \mathbb{N}$, we need some $k : \mathbb{N}$ such that $\mathrm{take}(k, \alpha) \in Q^\natural$. But Q is a bar, so we already have k such that $\mathrm{take}(k, \alpha) \in Q$, from which we immediately obtain $\mathrm{take}(k, \alpha) \in Q^\natural$. ∎

Theorem 5.5. *Decidable bar induction follows from monotone bar induction.*

Proof. We fix a *decidable* bar $\langle\rangle \triangleleft Q$ and an inductive superset $R \supseteq Q$, to show that $\langle\rangle \in R$. By Lemma 5.4, we have a monotone decidable bar $Q^\natural$; following Dummett (2000), we apply monotone bar induction on $Q^\natural$, choosing the motive $S \triangleq R \cup Q^\natural$. We clearly have the base case $Q^\natural \subseteq S$; to see that S is inductive, we assume that $\forall x.\ \vec{u}\,{}^\frown x \in S$, to show that $\vec{u} \in S$. Because $Q^\natural$ is decidable, we can proceed by case:

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

1. If $\vec{u} \in Q^\natural$, then we immediately have $\vec{u} \in S$.
2. If $\vec{u} \notin Q^\natural$, then our only option is to check that $\vec{u} \in R$. Using the inductiveness of R, it suffices to verify that $\vec{u}\,\hat{}\,x \in R$ for all x. By assumption, we have $\vec{u}\,\hat{}\,x \in S$; we proceed by case on whether $\vec{u}\,\hat{}\,x \in Q^\natural$:
   a. Suppose $\vec{u}\,\hat{}\,x \in Q^\natural$; because $\vec{u} \notin Q^\natural$, we know that no prefix of $\vec{u}$ is in Q, so therefore we must have $\vec{u}\,\hat{}\,x \in Q$. Because Q is a subset of R, we are done.
   b. Suppose $\vec{u}\,\hat{}\,x \notin Q^\natural$; because we have assumed $\vec{u}\,\hat{}\,x \in S$, we therefore already have $\vec{u}\,\hat{}\,x \in R$. ■

Under a further continuity principle, which Escardó (2013) has shown to be validated for **T**-definable functionals, monotone and decidable bar induction are in fact equivalent (Troelstra & van Dalen, 1988; Dummett, 2000).

5.3 Schwichtenberg's closure theorem

In 1979, Schwichtenberg proved an even stronger result than what we have proved here, namely that for any closed **System T** term that codes a bar of type 0 and 1, the *bar recursor* can already be defined in **T** (Schwichtenberg, 1979). In more recent work, Oliva and Steila give an elegant and *direct* proof of Schwichtenberg's result (Oliva & Steila, 2018). It should be possible to replicate this result in our setting by using Church encodings of dialogues, which Escardó has used to exhibit the modulus of continuity of a **T**-definable functional as a program in **System T** (Escardó, 2013).

The results of Schwichtenberg, Oliva and Steila, Escardó, Xu, and ourselves are part of long tradition of comparing formal definability with continuity and bar induction principles, initiated by Kreisel & Troelstra (1970) in a pioneering tour de force that clarified the closure conditions enjoyed by neighborhood functions (functions that correspond to Brouwer's inductive bars).

5.4 Bar recursion and continuous moduli

Whereas bar induction is a principle of logic (a method to obtain proofs), bar *recursion* is an principle of mathematical construction (a method to define functions) proposed by Spector (1962). In results obtained subsequent to those described in our own paper, Fujiwara & Kawai (2019) show that bar induction for decidable bars is equivalent to the existence of a bar recursor for every functional of type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ that has a *continuous* modulus of continuity. This result is closely related to that of Schwichtenberg (1979), as well as those of Escardó and ourselves, considering Escardó's proof that every definable such functional in **System T** has a definable modulus of continuity—and hence a continuous modulus of continuity (Escardó, 2013).

In other recent work, also obtained subsequent to our results,[11] Xu has extended Escardó's methods to achieve a monadic translation of **System T** into itself parameterized in a monad; Xu's result includes an elegant algorithm to obtain both moduli of continuity and bar recursors of **System T** programs (Xu, 2020a,b).

[11] Our proof was completed and announced in Spring of 2016.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

5.5 Martin-Löf's analysis of Π^1_1 sentences

In his doctoral dissertation, Per Martin-Löf pursued an explanation of constructive meaning in which Π^1_1 sentences ending in a decidable predicate are *defined* to be canonically verified by $(\eta, \digamma)$-trees as envisioned by Brouwer (Martin-Löf, 1970). A Π^1_1 sentence is one of the form $\forall\overline{\alpha_i : S_i}.\ \exists\overline{\beta_j : T_j}.\ P(\overline{\alpha_i} \ldots \overline{\beta_i})$, in which S_i, T_i are types of order at most 1; the statement that some decidable subset Q is a bar is an example of such a sentence, but Martin-Löf's insight was that one might as well consider *all* such sentences as being about bars.

In contrast to the Brouwer–Heyting–Kolmogorov (BHK) interpretation of Intuitionistic logic, Martin-Löf did not at the time consider every well-formed sentence generated from $\forall, \exists, \vee, \wedge, \supset, \neg, P$ to be constructively meaningful; rather, he argued that different kinds of sentences needed to be analyzed individually and interpreted, in order to be endowed with constructive meaning. This hesitant and open-ended perspective on constructive meaning is in fact much closer to Brouwer's views than the perhaps infelicitously named BHK interpretation—an interpretation that, in generality, fails to verify Brouwer's thesis on bars (which might as well be called "Brouwer's thesis on Π^1_1 sentences").

Exhibiting a theory of meaning that is at once compatible with both the BHK interpretation *and* Brouwer's interpretation of Π^1_1 sentences is a difficult matter, and leads inevitably to *forcing*—both sheaf-theoretic and effectful. The subject of this paper is to explain one possible BHK-compatible theory of meaning that also validates Brouwer's thesis, but we must admit that realizability in **System T** is too formalistic a constraint to be realistic.

5.6 Operational forcing

A forerunner of Escardó's effectful forcing can be found in Coquand and Jaber's *A computational interpretation of forcing in Type Theory* (2012); unlike effectful forcing, which is a purely denotational construction, Coquand and Jaber give an *operational* interpretation of **System T** extended by an oracle $\mathtt{f} : \mathtt{nat} \Rightarrow \mathtt{bool}$, and use it to show that the definable functionals on sequences of booleans are *uniformly* continuous.

Using similar ideas, Coquand & Mannaa (2016) have constructed a countermodel to Markov's principle for dependent type theory from an operational perspective, adding a new constant $\mathtt{f} : \mathtt{nat} \Rightarrow \mathtt{bool}$ to the language. In order to specify the equational behavior of the generic point f, the judgments of the type theory are indexed in a *forcing condition* p that specifies a finite subgraph of a sequence of booleans. Then, two conditions are imposed:

1. The judgments of type theory must be *monotone* with respect to approximations of forcing conditions.
2. The judgments of type theory must be *local* with respect to coverings of forcing conditions: roughly, if the judgment $\mathscr{J}$ holds at each leaf of an (η, β) tree rooted at p, then $\mathscr{J}$ holds at p.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

The locality condition above has the result of rendering the semantic evidence of the *judgments* of the forcing extension into (η, β) trees; this is to be contrasted with the denotational style of effectful forcing, in which the terms themselves are interpreted as (η, β) trees (potentially at higher type). This is concordant with the usual differences between operational and denotational approaches to semantics.

5.6.1 Forcing in Nuprl

Nuprl is one of the oldest implementations of dependent type theory, inspired by the ideas of Bishop, De Bruijn, Martin-Löf, and Scott (Constable *et al.*, 1986). In contrast to the algebraic tradition of type theory, in which the sound and complete interpretation of a formal language into a suitable class of models is emphasized, Nuprl is based on the idea of considering a *single* intended model, whose properties are meant to approximate the limitations of human mental construction.

Over the years, the designers of Nuprl have increasingly emphasized the validity of nonconstructive but computationally justified principles in their evolving computational model, including type-theoretic variants of Markov's principle, bar induction, and Brouwer's continuity theorems (Constable, 2014; Rahli & Bickford, 2016; Rahli *et al.*, 2017). The validity of Markov's principle and bar induction in Nuprl followed roughly from the fact that Nuprl's computational semantics had been developed in a classical metatheory, in which these principles already hold. At a high level, the classical metatheory of the operational model allowed termination properties of programs to be established using classical logic.

Finding this state of affairs unsatisfactory, researchers working on Nuprl have recently developed a new constructive semantics based on forcing (Bickford *et al.*, 2018), in a manner that generalizes the forcing conditions of Coquand & Mannaa (2016) to include finite data about an unbounded collection of choice sequences.

The resulting Nuprl system can be seen as a synthesis of Brouwer's purely denotational understanding of constructive meaning with the operational approach to constructivity pioneered by Martin-Löf in his influential report, *Constructive Mathematics and Computer Programming* (1979).

5.6.2 Relation to sheaf models

The operational account of forcing extensions of type theory described above is tantalizing close to the notion of a sheaf model over a topological space (such as the space of sequences of naturals or booleans). While it was perhaps unclear at the time, due to the highly unfolded character of the constructions, this connection can be rationalized in the following way: the extended operational model of Coquand and Mannaa is, at a high level, just the *standard* operational model of dependent type theory carried out internally to the topos of sheaves on Cantor space. A similar statement should be true of the operational model of Nuprl's forcing extension (Bickford *et al.*, 2018).

More recently, Sterling & Harper (2018) have developed an analogous construction of a different forcing extension of type theory for *guarded recursion*, utilizing the internal language of another sheaf topos to express the operational semantics and relational interpretation of the judgments of type theory.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

5.7 Denotational forcing

The main ideas underlying our proof of Brouwer's bar thesis for **T**-realizable bars were invented by Escardó in his paper *Continuity of Gödel's System T definable functionals via effectful forcing* (Escardó, 2013). Escardó used the effectful forcing technique to prove a related property of **T**-definable functionals $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, namely continuity:

$$\forall \alpha.\ \exists k.\ \forall \beta.\ \mathrm{take}(k, \alpha) = \mathrm{take}(k, \beta) \implies F(\alpha) = F(\beta)$$

The proof of this fact, being constructive, contains an *algorithm* to compute this modulus of continuity k from each sequence α.

Relative to Escardó, we have shown that the same method can be used to validate a restricted version of Brouwer's bar thesis, contributing a novel normalization theorem for Escardó's noncanonical (η, β)-trees into Brouwer's canonical $(\eta, \digamma)$ trees. Unlike Escardó, who developed effectful forcing for a version of **System T** extended by an oracle, we have avoided adding the oracle to the syntax, following a suggestion from Vincent Rahli; finally, following a suggestion from one of the anonymous referees, we have achieved a more compositional dialogue interpretation.

5.7.1 Syntactic forcing translations

Recently a number of researchers have investigated forcing extensions of the Coq proof assistant and its calculus of inductive constructions, further clarifying the connection between forcing, computational effects, and parametricity (Jaber *et al.*, 2012; Jaber *et al.*, 2016; Pédrot & Tabareau, 2017; Pédrot & Tabareau, 2019). Coq's forcing apparatus differs from that of Nuprl in a couple important ways.

First, forcing over an arbitrary preorder is supported, whereas Nuprl has been hard-coded for a specific forcing extension. On the other hand, Nuprl's forcing is *localized* in the sense that one may amalgamate objects defined on a bar, in essence a sheaf condition; the local character of the Nuprl semantics is very important, as it is the main ingredient to justifying the bar induction principle in the forcing extension.

Second, the consistency of Coq's forcing extension is justified by purely syntactical means through a simple translation into the base calculus. In contrast, Nuprl's forcing extension is justified by means of a completely new version of the partial equivalence relation semantics in which monotonicity and local character conditions are fully unfolded à la Coquand & Mannaa (2016) rather than treated abstractly à la Sterling & Harper (2018), an enormous undertaking that took multiple years.

5.7.2 Sheaf-theoretic forcing

An arguably more direct approach to obtain the compatiblity of type theories with Brouwerian principles such as bar induction and continuity is to consider the petit topos of sheaves on a suitable space (such as Baire space or Cantor space), or the gros topos of sheaves over a suitable category of test spaces (Fourman, 1984).

In his doctoral thesis, Xu (2015) considers sheaf models of dependent type theory validating uniform continuity and the fan theorem. There is an apparent difficulty modeling type-theoretic universes in sheaves, which some authors have erroneously located in the fact that the amalgamation of a family of types defined at the leaves of a bar is unique only

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

up to isomorphism (Xu & Escardó, 2016), or in the apparent impredicativity of certain classic constructions of sheafification (Xu, 2015).

In fact, universes of sheaves are known to be unproblematic (Streicher, 2005), and sheafification is entirely compatible with predicative foundations (Awodey *et al.*, 2009); the real location of the difficulty pointed out by Xu and Escardó is that known constructions of sheafification do not preserve the choice of codes for type connectives on the nose, so the strict reduction rules that identify $\mathrm{El}(\hat{A} \hat{\to} \hat{B})$ with $\mathrm{El}(\hat{A}) \to \mathrm{El}(\hat{B})$ must be weakened to canonical isomorphisms.

These difficulties motivated Coquand *et al.* (2017) to consider models of type theory in *stacks*, which express descent from a bar in a weak enough fashion that universes can be modeled without sheafification. The stack model of type theory contains a universe that classifies *discrete* types (types with no higher dimensional structure), and has been used to give a purely denotational account of the independence of Martin-Löf's type theory from Markov's principle.

Acknowledgments

Thanks to Carlo Angiuli, Mark van Atten, Mark Bickford, Bob Constable, Thierry Coquand, Martín Escardó, Bob Harper, and Per Martin-Löf for enlightening conversations about the bar theorem; I am especially thankful to Pierre-Marie Pédrot for pointing out an error in a previous draft of this paper, and to Vincent Rahli for suggesting a significant simplification to the proof, and to one of the anonymous referees for a suggestion to make the dialogue interpretation compositional. I am grateful to Bob Constable for scanning and sending to me a copy of Per Martin-Löf's doctoral dissertation. In my formalization, I benefited from Darin Morrison's alternative Agda prelude library. Finally, I thank the referees for their helpful and constructive criticisms, and I thank Tristan Nguyen at AFOSR for support.

This work was supported in part by AFOSR under grants MURI FA9550-15-1-0053 and FA9550-19-1-0216. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR.

Conflicts of interest

None.

References

Ackermann, W. (1928) Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen* **99**(1), 118–133.

Anel, M. & Joyal, A. (2019) *Topo-logie*. Preprint.

Awodey, S., Gambino, N., Lumsdaine, P. L. & Warren, M. A. (2009) Lawvere–Tierney sheaves in algebraic set theory. *J. Symb. Logic.* **74**(Sept.), 861–890.

Bauer, A. (2006) *Sometimes all Functions are Continuous*. Blog post.

Bickford, M., Cohen, L., Constable, R. L. & Rahli, V. (2018) Computability beyond church-turing via choice sequences. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. pp. 245–254. ACM.

https://doi.org/10.1017/S0956796821000095 Published online by Cambridge University Press

Brouwer, L. E. J. (1981) *Brouwer's Cambridge Lectures on Intuitionism*. Cambridge University Press.

Capretta, V. & Uustalu, T. (2016) A coalgebraic view of bar recursion and bar induction. In *Foundations of Software Science and Computation Structures: 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings,* Jacobs, B. & Löding, C. (eds). Springer Berlin Heidelberg, pp. 91–106.

Constable, R. L., Allen, S. F., Bromley, H. M., Cleaveland, W. R., Cremer, J. F., Harper, R. W., Howe, D. J., Knoblock, T. B., Mendler, N. P., Panangaden, P., Sasaki, J. T. & Smith, S. F. (1986) *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc.

Constable, R. (2014) *Virtual Evidence: A Constructive Semantics for Classical Logics*.

Constable, R. and Bickford, M. (2014) Intuitionistic completeness of first-order logic. *Ann. Pure Appl. Logic* **165**(1), 164–198. The Constructive in Logic and Applications.

Coquand, T., Mannaa, B. & Ruch, F. (2017) Stack semantics of type theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS),* pp. 1–11.

Coquand, T. & Jaber, G. (2012) A computational interpretation of forcing in type theory. In *Epistemology versus Ontology*, Dybjer, P., Lindström, S., Palmgren, E. and Sundholm, G. (eds), Logic, Epistemology, and the Unity of Science, vol. 27. Springer Netherlands, pp. 203–213.

Coquand, T. & Mannaa, B. (2016) The independence of Markov's principle in type theory. In *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, Kesner, D. and Pientka, B. (eds), Leibniz International Proceedings in Informatics (LIPIcs), vol. 52. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 17:1–17:18.

Coquand, T., Sambin, G., Smith, J. & Valentini, S. (2003) Inductively generated formal topologies. *Ann. Pure Appl. Logic* **124**(1), 71–106.

Dummett, M. (2000) *Elements of Intuitionism*. 2nd edn. Oxford Logic Guides, vol. 39. The Clarendon Press Oxford University Press.

Escardó, M. (2007) *Seemingly Impossible Functional Programs*. Guest post on Andrej Bauer's blog.

Escardó, M. (2013) Continuity of Gödel's System T definable functionals via effectful forcing. *Electron. Not. Theor. Comput. Sci.* **298**, 119–141. Agda development: http://www.cs.bham.ac.uk/~mhe/dialogue/dialogue-lambda.html.

Fourman, M. P. (1982) Notions of choice sequence. In *L.E.J. Brouwer Centenary Symposium,* van Dalen, D. and Troelstra, A. (eds). North-Holland, pp. 91–105.

Fourman, M. P. (1984) Continuous Truth I: Non-constructive objects. In *Logic Colloquium 1982*, G. Lolli, G. L. and Marcja, A. (eds), Studies in Logic and the Foundations of Mathematics, vol. 112. Elsevier, pp. 161–180.

Fourman, M. P. (2013) Continuous truth II: Reflections. In *Logic, Language, Information, and Computation: 20th International Workshop, WoLLIC 2013, Darmstadt, Germany, August 20-23, 2013. Proceedings*, Libkin, L., Kohlenbach, U. and de Queiroz, R. (eds). Springer Berlin Heidelberg, pp. 153–167.

Fujiwara, M. & Kawai, T. (2019) Equivalence of bar induction and bar recursion for continuous functions with continuous moduli. *Ann. Pure Appl. Logic* **170**(8), 867–890.

Gambino, N. & Schuster, P. (2007) Spatiality for formal topologies. *Mathematical. Structures in Comp. Sci.* **17**(1), 65–80.

Gödel, K. (1958) Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica* **12**(3–4), 280–287.

Harper, R. (2012) *Exceptions are Shared Secrets*. Blog post.

Harper, R. (2016) *Practical Foundations for Programming Languages*. 2nd edn. Cambridge University Press.

Heyting, A. (1956) *Intuitionism, an Introduction*. Studies in Logic and the Foundations of Mathematics. North-Holland. Revised edition, 1966.

Hilbert, D. (1926) Über das Unendliche. *Mathematische Annalen* **95**, 161–190.

Jaber, G., Tabareau, N. & Sozeau, M. (2012) Extending type theory with forcing. In *2012 27th Annual IEEE Symposium on Logic in Computer Science,* pp. 395–404.

Jaber, G., Lewertowski, G., Pédrot, P.-M., Sozeau, M. & Tabareau, N. (2016) The Definitional Side of the Forcing. *Logics in Computer Science*.

Johnstone, P. T. (1982) *Stone Spaces*. Cambridge Studies in Advanced Mathematics. Cambridge University Press.

Johnstone, P. T. (1983) The point of pointless topology. *Bull. (New Ser.) Amer. Math. Soc.* **8**(1), 41–53.

Kreisel, G. & Troelstra, A. S. (1970) Formal systems for some branches of intuitionistic analysis. *Ann. Math. Logic* **1**(3), 229–387.

Levy, P. B. (2006) Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order Symb. Comput.* **19**, 377–414.

Longley, J. (1999) When is a functional program not a functional program? In *Proceedings of the Fourth ACM SIGPLAN International Conference on Functional Programming*, *ICFP 1999*. ACM, pp. 1–7.

Longley, J. & Normann, D. (2015) *Higher-Order Computability*. Theory and Applications of Computability. Springer.

Mac Lane, S. & Moerdijk, I. (1992) *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. Springer.

Martin-Löf, P. (1970) *Notes on Constructive Mathematics*. Almqvist and Wiksell.

Martin-Löf, P. (1979) Constructive mathematics and computer programming. In *6th International Congress for Logic, Methodology and Philosophy of Science*, pp. 153–175. Published by North Holland, Amsterdam, 1982.

Norell, U. (2009) Dependently typed programming in Agda. In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation*, *TLDI 2009*. ACM, pp. 1–2.

Oliva, P. & Steila, S. (2018) A direct proof of Schwichtenberg's bar recursion closure theorem. *J. Symb. Logic* **83**(1), 70–83.

Pédrot, P. & Tabareau, N. (2017) An effectful way to eliminate addiction to dependence. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE Press, pp. 1–12.

Pédrot, P.-M. & Tabareau, N. (2019) The fire triangle: How to mix substitution, dependent elimination, and effects. *Proc. ACM Program. Lang.* **4**(POPL).

Petrakis, I. (2010) *Brouwer's Fan Theorem*. Master's Thesis.

Rahli, V. & Bickford, M. (2016) A nominal exploration of Intuitionism. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, *CPP 2016*. ACM, pp. 130–141.

Rahli, V., Bickford, M. & Constable, R. (2017) Bar induction: The good, the bad, and the ugly. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS),* pp. 1–12.

Sambin, G. (2012) Real and ideal in constructive mathematics. In *Epistemology versus Ontology*, Dybjer, P., Lindström, S., Palmgren, E. and Sundholm, G. (eds), Logic, Epistemology, and the Unity of Science, vol. 27. Springer Netherlands, pp. 69–85.

Schwichtenberg, H. (1979) On bar recursion of types 0 and 1. *J. Symb. Logic* **44**(3), 325–329.

Spector, C. (1962) Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In *Recursive Function Theory: Proc. Symposia in Pure Mathematics*, Dekker, F. D. E. (ed), vol. 5. American Mathematical Society, pp. 1–27.

Sterling, J. & Harper, R. (2018) Guarded Computational Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM.

Streicher, T. (2005) Universes in toposes. In *From Sets and Types to Topology and Analysis: Towards Practical Foundations for Constructive Mathematics*, Crosilla, L. & Schuster, P. (eds), Oxford Logical Guides, vol. 48. Oxford University Press, pp. 78–90.

Sundholm, G. & van Atten, M. (2008) The proper explanation of intuitionistic logic: on Brouwer's demonstration of the bar theorem. In *One Hundred Years of Intuitionism (1907–2007): The Cerisy Conference*, van Atten, M., Boldini, P., Bourdeau, M. & Heinzmann, G. (eds). Birkhäuser Basel, pp. 60–77.

Tait, W. W. (1967) Intensional Interpretations of Functionals of Finite Type I. *J. Symb. Logic* **32**(2), 198–212.

Troelstra, A. & van Dalen, D. (1988) *Constructivism in Mathematics: An Introduction*. Studies in Logic and the Foundations of Mathematics, vol. 1. North-Holland.

van Atten, M., van Dalen, D. & Tieszen, R. (2002) Brouwer and weyl: The phenomenology and mathematics of the intuitive continuumt. *Philos. Math.* **10**(2), 203–226.

van Atten, M. (2004) *On Brouwer*. Wadsworth Philosophers Series. Thompson/Wadsworth.

van Dalen, D. (2013) *L.E.J. Brouwer: Topologist, Intuitionist, Philosopher: How Mathematics is Rooted in Life*. Springer.

van der Hoeven, G. & Moerdijk, I. (1984) Sheaf models for choice sequences. *Ann. Pure Appl. Logic* **27**(1), 63–107.

van Heijenoort, J. (2002) *From Frege to Gödel : A Source Book in Mathematical Logic, 1879–1931 (Source Books in the History of the Sciences)*. Harvard University Press.

Vezzosi, A., Mörtberg, A. & Abel, A. (2019) Cubical Agda: A dependently typed programming language with univalence and higher inductive types. In *Proceedings of the 24th ACM SIGPLAN International Conference on Functional Programming*, *ICFP 2019*. ACM.

Vickers, S. (1989) *Topology via Logic*. Cambridge University Press.

Vickers, S. (2007) *Locales and Toposes as Spaces*. Springer Netherlands, pp. 429–496.

Xu, C. (2015) *A Continuous Computational Interpretation of Type Theories*. PhD thesis, University of Birmingham.

Xu, C. (2020a) A Gentzen-Style Monadic Translation of Gödel's System T. In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, Ariola, Z. M. (ed), Leibniz International Proceedings in Informatics (LIPIcs), vol. 167. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 25:1–25:17.

Xu, C. (2020b) A syntactic approach to continuity of T-definable functionals. *Log. Meth. Comput. Sci.* **16**(Feb.).

Xu, C. & Escardó, M. (2016) *Universes in Sheaf Models*. Unpublished note.