Open Bar — a Brouwerian Intuitionistic Logic with a Pinch of Excluded Middle

Mark Bickford
Cornell University, USA

Liron Cohen
Ben-Gurion University, Israel

Robert L. Constable
Cornell University, USA

Vincent Rahli
University of Birmingham, UK

Abstract

One of the differences between Brouwerian intuitionistic logic and classical logic is their treatment of time. In classical logic truth is atemporal, whereas in intuitionistic logic it is time-relative. Thus, in intuitionistic logic it is possible to acquire new knowledge as time progresses, whereas the classical Law of Excluded Middle (LEM) is essentially flattening the notion of time stating that it is possible to decide whether or not some knowledge will *ever* be acquired. This paper demonstrates that, nonetheless, the two approaches are not necessarily incompatible by introducing an intuitionistic type theory along with a Beth-like model for it that provide some middle ground. On one hand they incorporate a notion of progressing time and include evolving mathematical entities in the form of choice sequences, and on the other hand they are consistent with a variant of the classical LEM. Accordingly, this new type theory provides the basis for a more classically inclined Brouwerian intuitionistic type theory.

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

Keywords and phrases Intuitionism, Extensional type theory, Constructive Type Theory, Realizability, Choice sequences, Classical Logic, Law of Excluded Middle, Theorem proving, Coq

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.21

1 Introduction

Classical logic and intuitionistic logic are commonly viewed as distinct philosophies. Much of the difference between the two philosophies can be attributed to the way they handle the notion of *time*. In intuitionistic logic time plays a major role as the intuitionistic notions of knowledge and truth evolve over time. In particular, the seminal concept of intuitionistic mathematics as developed by Brouwer is that of *infinitely proceeding* sequences of choices (called choice sequences) from which the continuum is defined [7, Ch.3]. Choice sequences are a primitive concept of finite sequences of entities (e.g., natural numbers) that are never complete, and can always be further extended with new choices [28; 8; 47; 48; 33; 49; 38]. These sequences can be "free" in the sense that they are not necessarily procedurally generated. This manifestation of the evolving concept of time in intuitionistic logic entails a notion of computability that goes far beyond that of Church-Turing. In fact, the concept of evolving knowledge in intuitionistic logic is grounded in Krikpe's Schema, which in turn relies on the notion of choice sequences, and is inconsistent with Church's Thesis [19, Sec.5]. Classical logic, on the other hand, is time-invariant. That is, its notions of knowledge and truth are constant and so the aspect of time is, intuitively speaking, flattened. As mentioned by van Atten, "Many people believe, unlike Brouwer, that mathematical truths are not tensed but eternal—either because such truths are outside time altogether (atemporal) or because they hold in all time (omnitemporal)" [7, p.19].

This critical difference between the two philosophies has been used extensively to refute classical results in intuitionistic logic. Brouwer himself used his concept of choice sequences to provide weak counterexamples to classical results such as "any real number different from 0 is also apart from 0" [26, Ch.8]. Those counterexamples are called *weak* in the sense that they depend on the existence of formulas that have not been either proven or disproven yet (e.g., the Goldbach conjecture). By defining a choice sequence in which the value 1 can only be picked once such an undecided conjecture has been resolved (proved or disproved), one could resolve this undecided conjecture using the Law of Excluded Middle (LEM), leading to a weak counterexample of LEM [13, Ch.1, Sec.1]. Kripke [34, Sec.1.1] also used the unconstrained nature of choice sequences to refute other classical results, namely Kuroda's conjecture and Markov's principle in Kreisel's FC system [30].[1] A constructive version of LEM in which the operators are interpreted constructively is also false in realizability theories such as the CTT constructive type theory [15; 5] because it allows deciding the undecidable halting problem [42, Sec.6.3] (therefore not relying on undecided conjectures). However, a weaker version of LEM that does not require providing a realizer of either its left or right disjuncts, was proved to be consistent with CTT [18; 29; 42, Sec.6.3]. But using a similar technique to Brouwer's, even this weak version of LEM was shown to be inconsistent with BITT, an intuitionistic extension of CTT with a computable notion of choice sequences [10, Appx.A].

The use of the growing-over-time nature of choice sequences to refute classical axioms, and in particular LEM which is a key component of classical reasoning, seems to indicate an incompatibility between classical logic and intuitionistic logic. However, in this paper we show that this does not have to be the case. To this end, we present a relaxed model of time that mitigates the two approaches. Namely, on one hand it supports the evolving nature of choice sequences, and on the other hand it enables variants of the classical LEM.

Concretely, we present OpenTT, a novel intuitionistic extensional type theory that incorporates the Brouwerian notion of choice sequences, and is inspired by BITT [10]. OpenTT goes beyond and departs from BITT in several ways. First, it is validated w.r.t. a novel Beth-like model, which we call the *open bar* model, that is significantly simpler than the one presented in [10]. Beth models were originally developed to provide meaning to intuitionistic formulas [50; 9; 23, Sec.145; 21, Sec.5.4], and they have proven especially well-suited to interpret choice sequences [19]. In such models, formulas are interpreted w.r.t. infinite trees of elements (such as numbers). The models are typically formulated using a forcing interpretation where the forcing conditions are finite elements of those trees that provide meaning to choice sequences at a given point in time. Allowing access within the logic to the infinitely proceeding elements of the forcing layer, i.e., the branches of the Beth trees formulas are interpreted against, enables the use of the undecided nature of those elements to derive the negation of otherwise classically valid formulas such as LEM. The open bar model sufficiently weakens the "undecided" nature of those elements to enable validating a variant of LEM.

Another benefit of OpenTT over BITT is that the notion of time induced by the new model is flexible enough to capture an intuitionistic theory of computable choice sequences,

[1] This method to refute classical axioms was reused via forcing methods (see, e.g., [20, Sec.7.2.4] for the relation between forcing and choice sequences). E.g., the independence of Markov's Principle with Martin-Löf's type theory was proven using a forcing method where the "free" nature of forcing conditions replaces the "free" nature of free choice sequences in Kripke's proof [16].

and in particular the Axiom of Open Data (a continuity axiom) that was missing from BITT [10] and is a key axiom of choice sequence theories. Therefore, OpenTT provides a computational setting for exploring the implications of such entities, for example, it can enable the development of constructive Brouwerian real number theories. At the same time, it also enables validating variants of the classical LEM. In other words, OpenTT together with the open bar model presented in the paper enable a more relaxed notion of time, providing a basis for a more classically-inclined Brouwerian intuitionistic theory.

Contributions and roadmap. Sec. 2 describes the core *syntactic* components of the type theory OpenTT. Sec. 3 presents the novel open bar *semantic* model, which is used to validate OpenTT. Then, OpenTT is shown to capture both a theory of choice sequences (Sec. 4), as well as a variant of LEM (Sec. 5). Sec. 6 concludes by discussing related and future work. All the results in the paper are formalized in Coq, see `https://github.com/vrahli/NuprlInCoq/tree/ls3/`, and we provide clickable hyperlinks to the formalization throughout the paper.

2 OpenTT and Choice Sequences

OpenTT is an intuitionistic extensional dependent type theory. It is composed of an untyped programming language, and a dependent type system that associates types with programs. A type T, viewed as a proposition, is said to be true if it is inhabited, i.e., if some program t has type T—in which case t is said to realize T. This connection is made formal through a realizability model described in Sec. 3, where types are interpreted as partial equivalence relations on programs. In addition to standard program constructs, OpenTT contains computable choice sequences.

Choice sequences are the seminal component in Brouwer's intuitionistic theory, and the one manifesting notions of time and growth over time. Choice sequences are infinitely proceeding sequences of elements, which are chosen over time from a previously well-defined collection. There are two main classes of choice sequences, which are often referred to as *lawlike* and *lawless* [46]. The lawlike ones are "completed constructions" [46, Sec.1.2], where the choices must be chosen w.r.t. a pre-determined "law" (e.g., a general recursive program). The lawless ones, by contrast, are never fully completed and can always be extended over time with further choices that are not constrained by any law, that is, they can be chosen "freely" (hence the name *free choice sequences*). In this paper we focus on a theory with free choice sequences, which is a key distinguishing feature in Brouwer's intuitionistic logic, and a manifestation of the fact that time is an essential component of Brouwer's logic because unlike lawlike sequences that are time-invariant, lawless ones keep on evolving over time.

The notion of time in OpenTT is captured through the use of worlds. The worlds discussed in Sec. 2.2 constitute, as is standard practice, a poset, and are concretely defined as states that store definitions as well as choice sequences' choices. Thus, a world captures a state at a given point in time. The evolving nature of time is then captured via a notion of world extension, allowing to add new definitions, choice sequences, and choices.

OpenTT is inspired by BITT [10]. To make the paper self-contained we shall also review the components that are identical to those in BITT, noting the differences, which we summarize here. In addition to the standard inference rules for the standard types that are listed in Fig. 1 (which are discussed in Appx. B), OpenTT also contains inference rules that capture a theory of choice sequences, as described in Sec. 4. Among those, the Axiom of Open Data is new compared to BITT. Another key difference between OpenTT and BITT is that the former also contains a variant of the Law of Excluded Middle (the salient principle

Figure 1 Syntax of OpenTT

$\eta \in$ `CSName`	(C.S. name)	$\delta \in$ `Abstraction`	(abstraction)		
$v \in$ `Value` $::= vt$	(type)	$\mid \lambda x.t$	(lambda)	$\mid \langle t_1, t_2 \rangle$	(pair)
$\mid \star$	(axiom)	$\mid$ `inl`(t)	(left injection)	$\mid$ `inr`(t)	(right injection)
$\mid \underline{i}$	(integer)	$\mid \eta$	(choice sequence)		
$vt \in$ `Type` $::= \mathbf{\Pi} x{:}t_1.t_2$	(product)	$\mid \mathbf{\Sigma} x{:}t_1.t_2$	(sum)		
$\mid \mathbb{U}_i$	(universe)	$\mid t_1 = t_2 \in t$	(equality)		
$\mid t_1{+}t_2$	(disjoint union)	$\mid \{x : t_1 \mid t_2\}$	(set)		
$\mid \mathbb{N}$	(numbers)	$\mid t_1 < t_2$	(less than)		
$\mid \mathbb{N}_{\lightning}$	(T.S. numbers)	$\mid t_1 <_{\lightning} t_2$	(T.S. less than)		
$\mid t_1 \# t_2$	(free from definitions)	$\mid$ `Free`	(choice sequences)		
$\mid {\lightning} t$	(time squashing)				
$t \in$ `Term` $::= x$	(variable)	$\mid t_1\ t_2$	(application)		
$\mid v$	(value)	$\mid$ `let` $x, y = t_1$ `in` t_2	(spread)		
$\mid$ `fix`(t)	(fixpoint)	$\mid$ `case` t_1 `of` `inl`$(x) \Rightarrow t_2 \mid$ `inr`$(y) \Rightarrow t_3$	(decide)		
$\mid$ `wDepth`	(world depth)	$\mid$ `if` $t_1{=}t_2$ `then` t_3 `else` t_4	(equality test)		
$\mid \delta$	(abstraction)				

of classical logic), described in Sec. 5, which is not valid in the latter.[2]

2.1 Syntax

OpenTT's programming language is an untyped, call-by-name λ-calculus, whose syntax is given in Fig. 1, and operational semantics in Sec. 2.3. For simplicity, numbers are considered to be primitive, and we write $\underline{n}$ for an OpenTT number, where n is a metatheoretical number. A term is either (1) a variable; (2) a canonical term, i.e., a value; or (3) a non-canonical term. Non-canonical terms are evaluated according to the operational semantics presented in Sec. 2.3. As discussed below, abstractions of the form δ can be unfolded through definitions, and are otherwise left abstract for the purpose of this paper. In what follows, we use all letters as metavariables and their types can be inferred from the context.

Choice sequences A choice sequence is identified with its name, of the form η, which for the purpose of this paper is an abstract type equipped with a decidable equality. For simplicity we only discuss choice sequences of numbers, while our Coq formalization supports more kinds of choice sequences. OpenTT includes a comparison operator on choice sequences, **if** $t_1{=}t_2$ **then** t_3 **else** t_4, which as defined in Sec. 2.3 reduces to the *then* branch if t_1 and t_2 are two choice sequences with the same name, and otherwise reduces to the *else* branch.

Types Types are syntactic forms that are given semantics in Sec. 3 via a realizability interpretation. The type system contains standard types such as dependent products of the form $\mathbf{\Pi} x{:}t_1.t_2$ and dependent sums of the form $\mathbf{\Sigma} x{:}t_1.t_2$. For convenience we often write $a =_T b$ for the type $a = b \in T$; $t \in T$ for $t =_T t$; $\mathbf{\Pi} x_1, \dots, x_n{:}t_1.t_2$ for $\mathbf{\Pi} x_1{:}t_1.\dots\mathbf{\Pi} x_n{:}t_1.t$ (and similarly for the other operators with binders); $t_1 \to t_2$ for the non-dependent $\mathbf{\Pi}$ type; `True` for $(\underline{0} = \underline{0} \in \mathbb{N})$; `False` for $(\underline{0} = \underline{1} \in \mathbb{N})$; and $\neg T$ for $(T \to$ `False`$)$.

OpenTT also includes types that allow capturing specific aspects of choice sequences. In particular, OpenTT includes a type `Free` of free choice sequences. It also includes the type $t \# T$ that indicates that t is a *sealed* member of T in the sense that it is equivalent to a term

[2] Precisely establishing the relationship between the two systems is left for future work.

Figure 2 Operational semantics of OpenTT

$$
\begin{array}{lll}
(\lambda x.F)\ a \mapsto_w F[x\backslash a] & \eta(\underline{i}) \mapsto_w w[\eta][i], & \text{if } \eta \text{ has a } i\text{'s choice in } w \\
\texttt{fix}(v) \mapsto_w v\ \texttt{fix}(v) & \texttt{wDepth} \mapsto_w |w| & \\
\multicolumn{3}{l}{\texttt{let}\ x, y = \langle t_1, t_2 \rangle\ \texttt{in}\ F \mapsto_w F[x\backslash t_1; y\backslash t_2]} \\
\multicolumn{3}{l}{\texttt{case}\ \texttt{inl}(t)\ \texttt{of}\ \texttt{inl}(x) \Rightarrow F \mid \texttt{inr}(y) \Rightarrow G \mapsto_w F[x\backslash t]} \\
\multicolumn{3}{l}{\texttt{case}\ \texttt{inr}(t)\ \texttt{of}\ \texttt{inl}(x) \Rightarrow F \mid \texttt{inr}(y) \Rightarrow G \mapsto_w G[y\backslash t]} \\
\multicolumn{3}{l}{\texttt{if}\ \eta_1 {=} \eta_2\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \mapsto_w t_i, \quad \text{where } i = 1 \text{ if } \eta_1 = \eta_2, \text{ and } i = 2 \text{ otherwise}}
\end{array}
$$

u in T, which is syntactically free from abstractions and choice sequences, which we denote by $\texttt{synSealed}(u)$ here (see Sec. 3 for more details). Those types are used to state axioms of the theory of choice sequences in Sec. 4.1.

2.2 Worlds

OpenTT's computation system is equipped with a library of definitions in which we also store choice sequences. We here call the library a *world*. A definition entry is a pair of an abstraction δ and a term t, written δ `==` t, which stipulates that δ unfolds to t.[3] A choice sequence entry is a pair of a choice sequence name, and a list of choices (i.e. terms).[4] For example, the pair $\langle \eta, [4, 8, 15] \rangle$ is an entry for the choice sequence named η, where $[4, 8, 15]$ is its list of choices so far. A world is therefore a state that records, at a given point in time, all the current definitions together with all the choice sequences that have been started so far, along with the choices that have been made so far for those choice sequences.

▶ **Definition 1** (Worlds). *A world w is a list of entries, where an entry is either a definition entry or a choice sequence entry. We denote by `World` the type of worlds.*

Next we introduce some necessary operations and properties on worlds.

▶ **Definition 2** (World operations and properties). *Let $w \in$ `World`. (1) $|w|$ denotes w's* depth, *that is the number of choices of its longest choice sequence. (2) w is called* singular, *denoted* $\texttt{sing}(w)$, *if it does not have two entries with the same name.*

The depth of worlds is used in Sec. 4.1 to approximate the modulus of continuity of a predicate at a choice sequence; while `sing` is used in Lem. 14.

A world (or a particular snapshot of the library) can be seen as a the state of knowledge at a given point in time. It may grow over time by adding new definitions, new choice sequence entries, or more terms to an already existing choice sequence entry. Accordingly, a world w_2 is said to extend a world w_1 if it contains more entries and choices, without overriding the ones in w_1. Note that the extension relation on worlds defines a partial order on `World`.

▶ **Definition 3** (World extension). *A world w_2 is said to extend w_1, denoted $w_2 \succeq w_1$, if w_1 is a list of the form $[e_1, \ldots, e_n]$ and w_2 is a concatanation of some world w and $[e'_1, \ldots, e'_n]$, where for all $1 \leq i \leq n$, either $e_i = e'_i$ or e_i and e'_i are choice sequence entries with the same name such that the list in e_i is an initial segment of that in e'_i.*

[3] As the precise form of definitions is irrelevant here, we refer the interested reader to [43].

[4] Our formalization also includes mechanisms to impose further restrictions on choice sequences which are not discussed here. See computation/library.v for further details.

2.3 Operational Semantics

Fig. 2 presents OpenTT's small-step operational semantics. It defines the $t_1 \mapsto_w t_2$ ternary relation between two terms and a world, which expresses that t_1 reduces to t_2 in one step of computation *w.r.t. the world* w. We omit the congruence rules that allow computing within terms such as: if $t_1 \mapsto_w t_2$ then $t_1(u) \mapsto_w t_2(u)$.

The application $\eta(\underline{i})$ of a choice sequence η to a number $\underline{i}$ reduces to $w[\eta][i]$, i.e., η's i's choice recorded in w, if such a choice exists, and otherwise the computation gets stuck. Note that even though this is a call-by-name calculus, it includes the following congruence rule to access choices of choice sequences: if $t_1 \mapsto_w t_2$ then $\eta(t_1) \mapsto_w \eta(t_2)$.

In OpenTT we also allow computing the depth of a world w, that is, the number of choices recorded in its longest choice sequence entry (this is an addition to BITT). The nullary expression `wDepth` reduces to $|w|$ in one computation step. It is used to realize an axiom of the theory of choice sequences in Sec. 4.1.2. It is important to note that before introducing this new computation, all computations were *time-invariant computations* in the sense that if a term t computes to a value v in a world w_1, then it will compute to a value computationally equivalent[5] to v in any world $w_2 \succeq w_1$. For example, for numbers, if a term t computes to a number $\underline{n}$ in some world w, then it also computes to $\underline{n}$ in all extensions of w. Such terms are called *time-invariant terms*. It is straightforward to see that `wDepth` is not time-invariant, as it can compute to different numbers in different extensions of a world. For example, if w_1 contains only one choice sequence η for which 4 choices have been made, then the expression `wDepth` reduces to $\underline{4}$ in w_1. Now, adding another choice to η gives us a world $w_2 \succeq w_1$ in which `wDepth` reduces to $\underline{5}$. This operator is said to be *weakly monotonic* in the sense that if it returns $\underline{k}$ in w_1, and $w_2 \succeq w_1$, then it can only return a value $\underline{k'} \geq \underline{k}$ in w_2. We next introduce types capturing the concept of time-invariance.

2.4 Space Squashing and Time Squashing

OpenTT includes a *squashing* mechanism, which we use (among other things) to validate some of the axioms in Sec. 4 and 5. It erases the evidence that a type is inhabited by squashing it down to a single constant inhabitant using set types [15, pp.60]: $\downarrow T = \{x : \texttt{True} \mid T\}$. The only member of this type is the constant $\star$, which is `True`'s single inhabitant. The constant $\star$ inhabits $\downarrow T$ if T is true/inhabited, but we do not keep the proof that it is true. See Appx. C or [41] for more details on squashing.

In addition to the space squashing operator OpenTT also features another form of squashing called *time squashing*. As discussed in Sec. 2.3, some computations are *time-invariant*, while others, such as `wDepth`, are not. These two kinds of computations have different properties,[6] and this distinction should be captured at the level of types. To this end, OpenTT includes type constructors such as the time-squashing operator $\rotatebox[origin=c]{90}{$\leftrightsquigarrow$}$. Given a type T, one can build the type $\rotatebox[origin=c]{90}{$\leftrightsquigarrow$}T$, that in addition to T's members also contains terms that behave like members of T at a particular instant of time (in a particular world).

For the purpose of this paper, we only focus on a particular form of time-squashing for numbers, omitting the general construction.[7] Accordingly, OpenTT features a $\mathbb{N}_{\rotatebox[origin=c]{90}{$\leftrightsquigarrow$}}$ type of

[5] For a precise definition of computational equivalence, see [27].

[6] E.g., if t is a time-invariant term that computes to a number $\underline{m}$ less than $\underline{n}$ in a world w, then t will also be less than $\underline{n}$ in all $w' \succeq w$. However, if t is a non-time-invariant number, t might be less than $\underline{n}$ in some extensions of w, and larger in others.

[7] See `per_qtime` in per/per.v for further details on $\rotatebox[origin=c]{90}{$\leftrightsquigarrow$}$'s sematics.

non-time-invariant (or time-squashed) numbers. While $\mathbb{N}$ is required to only be inhabited by time-invariant terms, $\mathbb{N}_{\wr}$ is not, and allows for terms (such as `wDepth`) to compute to different numbers in different world extensions. For example, $\mathbb{N}_{\wr}$ is allowed to be inhabited by a term t that computes to $\underline{3}$ in some world w, and to $\underline{4}$ in some world $w' \succeq w$. This distinction between $\mathbb{N}$ and $\mathbb{N}_{\wr}$ will be critical in the validation of one of the choice sequence axioms in Sec. 4.1.2, where we make use of the depth of worlds which is not time-invariant.

In addition to the time-squashed $\mathbb{N}_{\wr}$ type, OpenTT features a less than relation $t_1 <_{\wr} t_2$ on time-squashed numbers, whose semantics is described in Sec. 3. Although similar to the $t_1 < t_2$ type, as for $\mathbb{N}_{\wr}$, $t_1 <_{\wr} t_2$ differs by not requiring t_1 and t_2 to be time-invariant.

3 Open Bar Realizability Model

This section presents a novel Beth-style model, called the open bar model, used below to validate OpenTT, which as mentioned above contains both a theory of choice sequences and a weak version of the classical LEM. As is standard in Beth models (or Kripke models [35; 34]), formulas are interpreted w.r.t. worlds. Using Beth models such as the one used in [10], a syntactic expression T is given meaning at a world w if there exists a collection B of worlds that covers all possible extensions of w, such that T corresponds to a legal type in all worlds in B. Such a collection is called *a bar* of w. In these models one has to construct such bars to prove that expressions are types or that types are inhabited. For example, to prove that choice sequences have type $\mathbb{N} \to \mathbb{N}$, given a choice sequence η and a number $\underline{n}$, one must exhibit a bar where $\eta(\underline{n})$ indeed computes to a number.

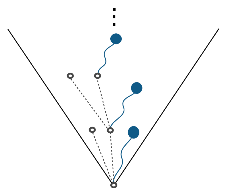
In this paper we take a different approach, one that avoids having to build bars altogether, and only requires building individual extensions of worlds. Intuitively, instead of requiring that a property P be true at a bar of a given world w, we require that for each extension w' of w, P holds for some extension of w'. Therefore, a major distinction between standard Beth models and our model is that in the former the semantics of a logical formula is computed based on the interpretation of that formula at a bar for the current world, while the latter only requires that in any possible extension of the current world there is always a further extension where the formula is given some meaning. Thus, our model only requires exhibiting *open bars* in the sense that not all infinite extensions of the current world necessarily have a finite prefix in the bar. Therefore, open bars are derivable from "standard" bars, but the converse does not hold. For the proof that choice sequences have type $\mathbb{N} \to \mathbb{N}$, this means that given an extension w' of the current world w, one must exhibit a further extension w'' where $\eta(\underline{n})$ computes to a number, which can be done by constructing w'' in which η contains at least $n+1$ choices.[8] As mentioned, in standard Beth models, in addition to this construction one has to also construct the bar. Thus, the notion of open bars seems to provide a more relaxed connection between truth and constructions than in the traditional Beth-like interpretation of intuitionistic logic, where one must *construct* bars to establish validity. By not having to make the full construction, the open bar model provides some middle ground between classical and intuitionistic logic. Furthermore, note that in a standard Beth model, depending on how the bar is defined, it is not always possible to constructively exhibit a point in the bar, whereas in the open bar model, the existence of the open bar directly gives a point at the open bar. This makes the construction of building bars from other bars generally simpler.

We start by introducing the concept of open bars, which is used below to interpret types.

[8] See rules/rules_choice1.v for a proof of this statement.

▶ **Definition 4** (Open Bars). *Let w be a world and f be a (metatheoretical) predicate on worlds. We say that f is true at an* open bar *of w if:*

$$\begin{aligned} \mathcal{O}(w,f) &= \forall_{\texttt{EXT}}(w, \lambda w'.\exists_{\texttt{EXT}}(w', \lambda w''.\forall_{\texttt{EXT}}(w'', f))) \\ \textit{where} \quad \forall_{\texttt{EXT}}(w,f) &= \forall w'.\ w' \geq w \Rightarrow f(w') \\ \exists_{\texttt{EXT}}(w,f) &= \exists w'.\ w' \geq w \wedge f(w') \end{aligned}$$



Informally, an open bar can be thought of as an object such as the one depicted on the right. There, the large solid blue nodes indicate worlds which we already know to be at the bar, while the small hollow nodes indicate worlds not yet at the bar from which the open bar provides a way to obtain worlds at the bar. For example, given the root of the tree, the open bar might give us the lowest solid blue world w. Given a world w', such as the one left to w, where different choices have been made from w, we can ask the bar to produce another world at the bar compatible with w' (i.e., that extends w'), and we might get the middle solid blue world.

The open bar semantics bears resemblance to the well known double negation translation [25] in standard Kripke models [35; 34]. Informally, in Kripke interpretations, $A \rightarrow B$ is interpreted as follows: $[\![A \rightarrow B]\!]_w = \forall_{\texttt{EXT}}(w, \lambda w'.[\![A]\!]_{w'} \Rightarrow [\![B]\!]_{w'})$. In such a semantics, the formula $\neg\neg A$ is then interpreted as $\forall_{\texttt{EXT}}(w, \lambda w'.\neg\forall_{\texttt{EXT}}(w', \lambda w''.\neg[\![A]\!]_{w''}))$, which is classically equivalent to $\forall_{\texttt{EXT}}(w, \lambda w'.\exists_{\texttt{EXT}}(w', \lambda w''.[\![A]\!]_{w''}))$. Nonetheless, our interpretation has two benefits over such a double negation translation: it is fully constructive, and it internalizes this double-negation/open-bar operator within the semantics, thereby avoiding having to use it explicitly in the theory. Note that this correspondence is unique to the *open* bar models, and does not hold in BITT's closed-bar model.

We now use open bars to provide meaning to OpenTT's types. As was done for similar theories [3; 4; 18; 6; 10], types are interpreted here by Partial Equivalence Relations (PERs) on closed terms. This PER semantics can be seen as an inductive-recursive definition of (see [22; 17] for similar construction methods):[9] (1) an inductive relation $T_1{\equiv_w}T_2$ that expresses type equality; (2) a recursive function $t_1{\equiv_w}t_2{\in}T$ that expresses equality in a type. The inductive definition $T_1{\equiv_w}T_2$ has one constructor per OpenTT type plus one additional constructor giving meaning to a type at a world w, based on its interpretation at an open bar of w (see Def. 6). Therefore, the recursive function $t_1{\equiv_w}t_2{\in}T$ has as many cases as there are constructors for $T{\equiv_w}T'$. The rest of this section presents some of these constructors and cases that illustrate key aspects of the new semantics. For simplicity we present them as equivalences, which are derivable from the formal definition. The others are defined similarly in Appx. A or in per/per.v. We first define some useful abstractions.

▶ **Definition 5.** *A term t is said to* inhabit *or* realize *a type T at w if $t{\equiv_w}t{\in}T$. We further use the following notations:* $\texttt{inh}(w,T)$ *for $\exists t.\ t{\equiv_w}t{\in}T$; $a \Downarrow_w b$ for 'a computes to b w.r.t. w', i.e., the reflexive and transitive closure of $\mapsto$; and $a \Downarrow\!\!\Downarrow_w b$ for $\forall_{\texttt{EXT}}(w, \lambda w'.a \Downarrow_{w'} b)$ which captures that a is time-invariant.*[10]

As mentioned above, a key aspect of our open bar model is that it is defined to be closed under open bars, allowing interpreting all types and their PERs in terms of open bars.

[9] Due to the limited support for induction-recursion in Coq, our formalization instead combines these two definitions into a single inductive definition following the method described in [4; 14], which results in the same theory, however defined in a slightly more convoluted way that the one defined here.

[10] We here omit some technical details; see `ccomputes_to_valc_ext` in per/per.v for the full definition.

▶ **Definition 6** (Open Bar Closure). *OpenTT's semantics is closed under open bars as follows:*

$$
\begin{aligned}
T_1 \equiv_w T_2 &\iff \mathcal{O}(w, \lambda w'.\exists T_1', T_2'.\ T_1 \Downarrow_{w'} T_1' \wedge T_2 \Downarrow_{w'} T_2' \wedge T_1' \equiv_{w'} T_2') \\
t_1 \equiv_w t_2 \in T &\iff \mathcal{O}(w, \lambda w'.\exists T'.\ T \Downarrow_{w'} T' \wedge t_1 \equiv_{w'} t_2 \in T')
\end{aligned}
$$

Let us now turn to the semantics of key types of OpenTT under the open bar semantics. We start with demonstrating the $\mathbb{N}$ type which is in the core types of CTT.

▶ **Definition 7** (Time-Invariant Numbers). *The $\mathbb{N}$ type is interpreted as follows:*

$$
\mathbb{N} \equiv_w \mathbb{N} \iff \texttt{True} \qquad t \equiv_w t' \in \mathbb{N} \iff \mathcal{O}(w, \lambda w'.\exists \underline{n}.\ t \Downarrow\!\!\downarrow_{w'} \underline{n} \wedge t' \Downarrow\!\!\downarrow_{w'} \underline{n})
$$

Note the use of $\Downarrow\!\!\downarrow$ above, since such numbers are required to be time-invariant (see Sec. 2.4).

In the next definition the time-invariant constraint is relaxed, allowing inhabitants of $\mathbb{N}_{\lightning}$ to compute to different numbers in different world extensions. For example, a term that computes to $\underline{3}$ in the current world w and to $\underline{4}$ in all (strict) extensions of w, inhabits $\mathbb{N}_{\lightning}$ but not $\mathbb{N}$. While $\mathbb{N}$ is a subtype of $\mathbb{N}_{\lightning}$, in the sense that all equal members of $\mathbb{N}$ are equal members of $\mathbb{N}_{\lightning}$, the converse does not hold. For example, `wDepth` is in $\mathbb{N}_{\lightning}$ but not in $\mathbb{N}$.

▶ **Definition 8** (Time-Squashed Numbers). *The $\mathbb{N}_{\lightning}$ type is interpreted as follows:*

$$
\begin{aligned}
&\mathbb{N}_{\lightning} \equiv_w \mathbb{N}_{\lightning} \iff \texttt{True} \qquad t \equiv_w t' \in \mathbb{N}_{\lightning} \iff \mathcal{O}(w, \lambda w'.\texttt{sameNats}(w', t, t')) \\
&\textit{where}\ \ \texttt{sameNats}(w, t, t') = \exists k.\ t \Downarrow_w \underline{k} \wedge t' \Downarrow_w \underline{k}
\end{aligned}
$$

As mentioned in Sec. 2.4, in addition to the $\mathbb{N}_{\lightning}$ type, OpenTT also provides a 'less-than' operator on such numbers, which we interpret as follows.

▶ **Definition 9** (Time-Squashed Less-Than). *The $t_1 <_{\lightning} t_2$ type is interpreted as follows:*

$$
\begin{aligned}
t_1 <_{\lightning} t_2 \equiv_w t_1' <_{\lightning} t_2' &\iff \mathcal{O}(w, \lambda w'.\texttt{sameNats}(w', t_1, t_1') \ \wedge\ \texttt{sameNats}(w', t_2, t_2')) \\
t \equiv_w t' \in t_1 <_{\lightning} t_2 &\iff \mathcal{O}(w, \lambda w'.\exists k_1, k_2.\ t_1 \Downarrow\!\!\downarrow_{w'} \underline{k_1} \wedge t_2 \Downarrow\!\!\downarrow_{w'} \underline{k_2} \wedge k_1 < k_2)
\end{aligned}
$$

Note that given t_1 and t_2 in $\mathbb{N}_{\lightning}$ that compute to $\underline{3}$ and $\underline{4}$ respectively in *some* world, one cannot derive $t_1 <_{\lightning} t_2$ as t_1 and t_2 could keep alternating between $\underline{3}$ and $\underline{4}$ such that t_2 computes to $\underline{4}$ when t_1 computes to $\underline{3}$, and vice versa. Though general rules for inferring such inequalities can be formalized[11], in what follows we only need a concrete instance of $t_1 <_{\lightning} t_2$ in which $t_1 \in \mathbb{N}$ and $t_2 = \texttt{wDepth} \in \mathbb{N}_{\lightning}$ (see Sec. 4.1.2, which makes use of $\texttt{wDepth} \in \mathbb{N}_{\lightning}$ to capture the modulus of continuity of a predicate at a choice sequence). In this case such alternations are avoided since `wDepth` is weakly monotonically increasing.

OpenTT also includes a type of free choice sequences, interpreted as follows.

▶ **Definition 10** (Choice Sequences). *The* `Free` *type is interpreted as follows:*

$$
\texttt{Free} \equiv_w \texttt{Free} \iff \texttt{True} \qquad t \equiv_w t' \in \texttt{Free} \iff \mathcal{O}(w, \lambda w'.\exists \eta.\ t \Downarrow\!\!\downarrow_{w'} \eta \wedge t' \Downarrow\!\!\downarrow_{w'} \eta)
$$

As mentioned in Sec. 2.1, OpenTT includes a $t\#T$ type, which states that the term t is a sealed member of T. For example $\texttt{True}\#\mathbb{U}_i$, $\texttt{False}\#\mathbb{U}_i$, and $\mathbb{N}\#\mathbb{U}_i$ are all inhabited types, whereas $(\eta \in \texttt{Free})\#\mathbb{U}_i$ is not inhabited because this type mentions the choice sequence η. Note that $t\#T$ and $\texttt{synSealed}(t)$ did not appear in BITT.

[11] Technically, our formalization includes both weakly monotonically increasing and decreasing numbers (denoted here $\mathbb{N}_{\lightning}^{\wedge}$ and $\mathbb{N}_{\lightning}^{\vee}$, respectively) allowing one to derive $t_1 <_{\lightning} t_2$ in w when $t_1 \in \mathbb{N}_{\lightning}^{\vee}$, $t_2 \in \mathbb{N}_{\lightning}^{\wedge}$, and t_2 computes to a number larger than t_1 in w.

▶ **Definition 11** (Free From Definitions). *The $a\#A$ type is interpreted as follows:*

$$
\begin{aligned}
a\#A \equiv_w b\#B &\iff A \equiv_{w'} B \wedge a \equiv_{w'} b \in A \\
t \equiv_w t' \in a\#A &\iff \mathcal{O}(w, \lambda w'.\exists x.\ a \equiv_{w'} x \in A \wedge \texttt{synSealed}(x))
\end{aligned}
$$

As mentioned above, the other type operators of OpenTT are interpreted in a similar fashion. This semantics of OpenTT satisfies the following properties, which are the standard properties expected for such a semantics [3; 18], including the monotonocity and locality properties expected for a possible-world semantics [50; 23; 21, Sec.5.4]—here monotonicity refers to types, and not to computations.[12]

▶ **Proposition 12** (Type System Properties). *The $T_1 \equiv_w T_2$ and $a \equiv_w b \in T$ relations satisfy the following properties (where free variables are universally quantified):*

transitivity:	$T_1 \equiv_w T_2 \Rightarrow T_2 \equiv_w T_3 \Rightarrow T_1 \equiv_w T_3$	$t_1 \equiv_w t_2 \in T \Rightarrow t_2 \equiv_w t_3 \in T \Rightarrow t_1 \equiv_w t_3 \in T$
symmetry:	$T_1 \equiv_w T_2 \Rightarrow T_2 \equiv_w T_1$	$t_1 \equiv_w t_2 \in T \Rightarrow t_2 \equiv_w t_1 \in T$
computation:	$T \equiv_w T \Rightarrow T \Downarrow_w T' \Rightarrow T \equiv_w T'$	$t \equiv_w t \in T \Rightarrow t \Downarrow_w t' \Rightarrow t \equiv_w t' \in T$
monotonicity:	$T_1 \equiv_w T_2 \Rightarrow w' \succeq w \Rightarrow T_1 \equiv_{w'} T_2$	$t_1 \equiv_w t_2 \in T \Rightarrow w' \succeq w \Rightarrow t_1 \equiv_{w'} t_2 \in T$
locality:	$\mathcal{O}(w, \lambda w'.T_1 \equiv_{w'} T_2) \Rightarrow T_1 \equiv_w T_2$	$\mathcal{O}(w, \lambda w'.t_1 \equiv_{w'} t_2 \in T) \Rightarrow t_1 \equiv_w t_2 \in T$

Using these properties, it follows that OpenTT is consistent w.r.t. the open bar model.

▶ **Theorem 13** (Soundness & Consistency). *OpenTT's inference rules are all sound w.r.t. the open bar model, which entails that OpenTT is consistent.*[13]

4 A Theory of Choice Sequences

This section focuses on OpenTT's inference rules that provide an axiomatization of a theory of choice sequences. This theory includes two variants of the Axiom of Open Data (Sec. 4.1.1 and 4.1.2), a density axiom (Sec. 4.2), and a discreteness axiom (Sec. 4.3). We focus our attention on the variants of the Axiom of Open Data that captures a form of continuity which is the core essence of choice sequences, as those where not handled in BITT.

4.1 The Axiom of Open Data (AOD)

The Axiom of Open Data (AOD) is perhaps the seminal axiom in the theory of choice sequences. It is a continuity axiom that states that the validity of properties of free choice sequences (with certain side conditions) can only depend on finite initial segments of these sequences. Let P be a sealed predicate on free choice sequences of numbers (i.e., $P\#(\texttt{Free} \to \mathbb{U}_i)$ for some universe i), $\mathbb{N}_n$ the type $\{x : \mathbb{N} \mid x < n\}$ of natural number strictly less than n, and $\mathcal{B}_n = \mathbb{N}_n \to \mathbb{N}$. The Axiom of Open Data can be formalized as follows:

$$\mathbf{\Pi}\alpha{:}\texttt{Free}.P(\alpha) \to \mathbf{\Sigma}n{:}\mathbb{N}.\mathbf{\Pi}\beta{:}\texttt{Free}.(\alpha =_{\mathcal{B}_n} \beta \to P(\beta)) \qquad (\texttt{AOD})$$

Since AOD is a form of continuity principle, and the non-squashed Continuity Principle is incompatible with CTT [41; 42] as well as with other computational theories [32; 45; 24], we only attempt to validate a squashed version of AOD. That is, since there is no way to compute the modulus of continuity of P at α, which is preserved over world extensions (as

[12] See per/nuprl_props.v for proofs of these properties.
[13] See rules.v and per/weak_consistency.v for more details.

required by the semantics of $\mathbb{N}$), we instead validate versions of AOD where the sum type is squashed. But there are two ways to squash it, as described in Sec. 4.1.1 and 4.1.2.

There are two additional restrictions we impose in order to validate the squashed variants of AOD. First, to validate the axiom we swap α and β in $P(\alpha)$. This has an impact on both the PER of this type and the world w.r.t. which it is validated. Given an inhabitant t of $P(\alpha)$, we can easily build a proof of $P(\beta)$ by swapping α and β in t. This is however a metatheoretical operation. Therefore, in our variants of AOD the $P(\beta)$ is squashed. Second, note that when swapping one needs to swap α and β in all definitions and choice sequences' choices in the world w.r.t. which it is validated, leading to a different world. Therefore, we require that choice sequences cannot occur in definitions and choice sequences' choices to ensure that swapping α and β in a world w leads to an equivalent world if α and β have the same choices. To see why this is necessary take P to be the predicate $P = \lambda y.\{x : \texttt{Free} \mid x =_{\texttt{Free}} y\}$, and the world w to contain the definition δ `==` α. Then, $P(\alpha)$ is equivalent to $\{x : \texttt{Free} \mid x =_{\texttt{Free}} \alpha\}$ and δ is a member of $P(\alpha)$ in w, while $P(\beta)$ is equivalent to $\{x : \texttt{Free} \mid x =_{\texttt{Free}} \beta\}$ in this world, and therefore δ is not a member of $P(\beta)$ if α and β are two different choice sequences.

Before presenting and validating the variants of AOD, we present a few intermediate results. First, we prove that from $\alpha =_{\mathcal{B}_n} \beta$, we can always construct a world in which α and β contain exactly the same choices.[14]

▶ **Lemma 14** (Intermediate World). *Let w_1 and w_2 be two worlds such that $w_2 \succeq w_1$ and* $\texttt{sing}(w_1)$ *(see Def. 2). If η_1 and η_2 are two free choice sequences that have the same choices up to $|w_1|$ in w_2, then there must exist a world w, such that $w_2 \succeq w \succeq w_1$, both η_1 and η_2 occur in w, they have the exact same choice in w, and all these choices are numbers.*

Furthermore, the following swapping operator swaps α and β in $P(\alpha)$ to obtain $P(\beta)$.[15]

▶ **Definition 15** (Swapping). *Let $X\cdot(\eta_1|\eta_2)$ be a swapping operation that swaps η_1 and η_2 everywhere in X, where X ranges over all the syntactic forms presented above.*

We can then prove that the various relations introduced in Sec. 3 are preserved by the above swapping operator. For example, crucially, we can prove that the $t_1 \equiv_w t_2 \in T$ relation, which expresses that t_1 and t_2 are equal members in T, is preserved by swapping.[16]

▶ **Lemma 16** (Swapping PERs). *If $t_1 \equiv_w t_2 \in T$ then $t_1\cdot(\eta_1|\eta_2) \equiv_{w\cdot(\eta_1|\eta_2)} t_2\cdot(\eta_1|\eta_2) \in T\cdot(\eta_1|\eta_2)$.*

4.1.1 The Space-Squashed Axiom of Open Data ($\texttt{AOD}_\downarrow$)

The first variant of AOD we validate is the a *space-squashed* one, called $\texttt{AOD}_\downarrow$.

▶ **Proposition 17.** *The following rule of OpenTT is valid w.r.t. the open bar model (where H is an arbitrary list of hypotheses):*

$$\frac{}{H \vdash \mathbf{\Pi}\alpha{:}\texttt{Free}.P(\alpha) \to \downarrow\mathbf{\Sigma}n{:}\mathbb{N}.\mathbf{\Pi}\beta{:}\texttt{Free}.(\alpha =_{\mathcal{B}_n} \beta \to \downarrow P(\beta))}$$

Proof. We here outline the proof, see rules/rules_ls3_v0.v for full details. Since the sum type is $\downarrow$-squashed, a realizer for this formula can simply be $\lambda\alpha, x.\star$ (see Sec. 2.4). Let P be a

[14] See Lemma `to_library_with_equal_cs` in rules/rules_choice_util4.v.
[15] See for example `swap_cs_term` in terms/swap_cs.v, which swaps two choice sequence names in a term.
[16] See `implies_equality_swap_cs` in rules/rules_choice_util4.v for the formal statement and proof.

sealed predicate on free choice sequences, α a free choice sequence, and instantiate n with $|w|$, the depth of the current world w. From $\alpha =_{\mathcal{B}_n} \beta$, we get that α and β have the same choices up to $|w|$ in the extension w' of w, and we have to show that $P(\beta)$ is true in w'. Lem. 14 entails that α and β have exactly the same choices in some world w'' between w and w'. Using Lem. 16 we swap α and β in $P(\alpha)$ and w''. Thus, because choice sequences cannot occur in definitions and choices, $P(\beta)$ is valid in a world equivalent to w'' and hence in w'' too.[17] Finally, using monotonicity (Lem. 12), we obtain that $P(\beta)$ is true also in w'. ◀

4.1.2 The Time-Squashed Axiom of Open Data ($\texttt{AOD}_{\downzigzagarrow}$)

Next, we present a *time-squashed* version of AOD, where instead of ↓-squashing the sum type the $\mathbb{N}_{\downzigzagarrow}$ time-squashed type is used, and $\mathcal{B}_{\downzigzagarrow n} = \{x : \mathbb{N} \mid x <_{\downzigzagarrow} n\} \rightarrow \mathbb{N}$ Is used instead of $\mathcal{B}_n$.[18]

$$\mathbf{\Pi}\alpha{:}\texttt{Free}.P(\alpha) \rightarrow \mathbf{\Sigma}n{:}\mathbb{N}_{\downzigzagarrow}.\mathbf{\Pi}\beta{:}\texttt{Free}.(\alpha =_{\mathcal{B}_{\downzigzagarrow n}} \beta \rightarrow \downarrow P(\beta)) \qquad (\texttt{AOD}_{\downzigzagarrow})$$

Note that because n is not a member of $\mathbb{N}$ anymore but of $\mathbb{N}_{\downzigzagarrow}$, we use $\mathcal{B}_{\downzigzagarrow n}$ instead of $\mathcal{B}_n$ here to state that α and β are equal sequences up to n. If $n \in \mathbb{N}_{\downzigzagarrow}$ then $x < n$, where $x \in \mathbb{N}$, and $\mathcal{B}_n$ are not types anymore: the semantics of $x < n$ requires both x and n to be time-invariant numbers (see Sec. 2.4). Therefore, we use $x <_{\downzigzagarrow} n$ here instead, which does not require numbers to be time-invariant as per its semantics presented in Def. 9.

Before diving into the proof of $\texttt{AOD}_{\downzigzagarrow}$'s validity, we first present a few intermediate results.

▶ **Lemma 18.** *The $\mathbb{N}$ type is a subtype of $\mathbb{N}_{\downzigzagarrow}$, in the sense that all equal members in $\mathbb{N}$ are also equal members in $\mathbb{N}_{\downzigzagarrow}$ (which implies that $t_1 <_{\downzigzagarrow} t_2$ is a type even when $t_1 \in \mathbb{N}$ and $t_2 \in \mathbb{N}_{\downzigzagarrow}$), and the* `wDepth` *expression is a member of $\mathbb{N}_{\downzigzagarrow}$ (i.e., it is equal to itself in $\mathbb{N}_{\downzigzagarrow}$).*[19] *I.e. the following rules are valid in OpenTT.*

$$\frac{H \vdash t_1 =_{\mathbb{N}} t_2}{H \vdash t_1 =_{\mathbb{N}_{\downzigzagarrow}} t_2} \qquad\qquad \frac{}{H \vdash \texttt{wDepth} =_{\mathbb{N}_{\downzigzagarrow}} \texttt{wDepth}}$$

For $\texttt{AOD}_{\downarrow}$, because its $\mathbf{\Sigma}$ type is ↓-squashed, we did not have to provide a witness for the modulus of continuity of P at α. Instead, we could simply find a suitable metatheoretical number in the proof of its validity, without having to provide an expression from the object theory that computes that number. In the metatheoretical proof, we computed the depth of the current world, which is a metatheoretical number k, and simply used $\underline{k}$, which is a number in the object theory, as an approximation of the modulus of continuity of P at α. The situation is different in $\texttt{AOD}_{\downzigzagarrow}$ because the $\mathbf{\Sigma}$ type is no longer ↓-squashed. We now have to provide an expression from the object theory that computes that modulus of continuity. As mentioned, we use `wDepth`, which is an expression of *OpenTT*, the object theory. Thus, we now have to prove that `wDepth` has the right type, namely, $\mathbb{N}_{\downzigzagarrow}$, which we proved in Lem. 18.

Using these results we prove that $\texttt{AOD}_{\downzigzagarrow}$ is valid w.r.t. the semantics presented in Sec. 3.

▶ **Proposition 19.** *The following rule of OpenTT is valid w.r.t. the open bar model:*

$$\frac{}{H \vdash \mathbf{\Pi}\alpha{:}\texttt{Free}.P(\alpha) \rightarrow \mathbf{\Sigma}n{:}\mathbb{N}_{\downzigzagarrow}.\mathbf{\Pi}\beta{:}\texttt{Free}.(\alpha =_{\mathcal{B}_{\downzigzagarrow n}} \beta \rightarrow \downarrow P(\beta))}$$

---

[17] See Lemma `member_swapped_css_libs` in rules/rules_choice_util4.v.

[18] Note that as in $\texttt{AOD}_{\downarrow}$, $P(\beta)$ is also ↓-squashed here. We leave for future work to derive a version where $P(\beta)$ is not squashed. Note also that the modulus of continuity n is here in $\mathbb{N}_{\downzigzagarrow}$. We have validated another version of this axiom in rules/rules_ls3_v1.v where $n \in \mathbb{N}^{\wedge}_{\downzigzagarrow}$, i.e., where n is required to be weakly monotonically increasing, which is true about `wDepth` (see Sec. 2.3 and 2.4).

[19] See `rule_qnat_subtype_nat_true` in rules/rules_ref.v and `rule_depth_true` in rules/rules_qnat.v.

Proof. We here outline the proof (which is similar to that of Prop.17), while full details are in rules/rules_ls3_v2.v. Since now the sum type is not $\downarrow$-squashed, we have to provide a witness for it. The realizer we provide for this formula is: $\lambda\alpha, x.\langle \mathtt{wDepth}, \lambda\beta, y.\star\rangle$. Let P be a sealed predicate on free choice sequences, and let α be a free choice sequence. We now have to prove that $\mathtt{wDepth} \in \mathbb{N}_{\wr}$, which follows from Lem. 18. Since $\mathtt{wDepth}$ computes to $|w|$, where w is the current world, we can then use $|w|$ as an approximation of the modulus of continuity of P at α, as in Prop. 17's proof. One difference with Prop. 17's proof is that we have here that $\alpha =_{\mathcal{B}_{\wr n}} \beta$ (which we prove to be a type using Lem. 18) instead of $\alpha =_{\mathcal{B}_n} \beta$. This however still suffices to show that α and β have the same choices up to $|w|$ in the extension w' of w. From here, the proof proceeds just as that of Prop. 17. ◀

4.2 The Density Axiom (DeA)

Another common free choice sequence axiom, sometimes called the *density* axiom [44], states that for any finite sequence of numbers f, there is a free choice sequence that contains f as initial segment (this is Axiom 2.1 in [31, Sec.2], also referred to as LS1 in [19]).

In BITT the following Density Axiom (DeA) was validated: $\boldsymbol{\Pi} n{:}\mathbb{N}.\boldsymbol{\Pi} f{:}\mathcal{B}_n.\boldsymbol{\Sigma}\alpha{:}\mathtt{Free}.(f =_{\mathcal{B}_n} \alpha)$ [10]. The proof of its validity was by generating an appropriate choice sequence space that contains the values of the finite sequence f as part of its name. More precisely, given a finite sequence f of n terms in $\mathbb{N}$ from the object theory, BITT includes computations to extract those n numbers, say $\underline{k_1}, \dots, \underline{k_n}$, and build a choice sequence with the metatheoretical list of numbers $[k_1, \dots, k_n]$ as part of its name, and which is used to witness DeA's $\boldsymbol{\Sigma}$ type. In OpenTT we opted against including such names for two reasons. First, in the open bar model it is possible to validate a squashed version of DeA (where the $\boldsymbol{\Sigma}$ type is squashed) without including lists of numbers in choice sequence names. This is because the open bar model allows for internal choices to be made (see Prop. 20 below). Moreover, deterministically generating choice sequence names is not preserved by swapping (which would be required for example for Lem. 16 to hold). Given a term t that deterministically generates η_1, it might be that swapping η_1 for η_2 turns η_1 into η_2 and leaves t unchanged, while t does not generate η_2.

Therefore, we do not include metatheoretical lists of numbers as part of choice sequence names in OpenTT and only validate the following $\downarrow$-squashed version of DeA, called $\mathtt{DeA}_{\downarrow}$.

▶ **Proposition 20.** *The following rule of OpenTT is valid w.r.t. the open bar model:*

$$\frac{}{H \vdash \boldsymbol{\Pi} n{:}\mathbb{N}.\boldsymbol{\Pi} f{:}\mathcal{B}_n.\downarrow\boldsymbol{\Sigma}\alpha{:}\mathtt{Free}.(f =_{\mathcal{B}_n} \alpha)}$$

Proof. As this axiom is $\downarrow$-squashed, we realize it using $\lambda n, f.\star$. To prove its validity in some world w, assume $n \in \mathbb{N}$ and $f \in \mathcal{B}_n$ in some $w' \succeq w$. We have to exhibit some $w'' \succeq w'$ that contains a free choice sequence that has f as its initial segment. This world w'' can simply be w' augmented with a fresh (w.r.t. w') choice sequence that has f as its initial segment.[20] ◀

Note that the Beth model in [10] requires exhibiting a choice sequence such that DeA holds *at a bar b of w*. Without a mechanism to enforce initial segments, it could be that the choice sequence picked to witness α does not include the correct choices in some of b's branches. This is why BITT features choice sequence names that enforce initial segments. Thanks to open bars, OpenTT is able to do without enforcing initial segments within choice sequence names while still featuring a version of DeA, at the detriment of requiring its $\boldsymbol{\Sigma}$

[20] See rules/rules_choice1.v for more details.

type be $\downarrow$-squashed. (Troelstra calls the free choice sequences that enforce initial segments *lawless*, and the ones where no initial segment is enforced *proto-lawless* [44, Sec.2.4].)

4.3 The Discreteness Axiom (DiA)

One final common free choice sequence axiom, sometimes called the *discreteness* axiom [39], states that equality between free choice sequences is decidable (it is Axiom 2.2 in [31, Sec.2], also referred to as LS2 in [19]). As for BITT, OpenTT features intensional and extensional versions of the Discreteness Axiom (DiA), which we have proven to be valid w.r.t. the open bar model (we only present the extensional version here due to space constraints).[21]

▶ **Proposition 21.** *The following rule of OpenTT is valid w.r.t. the open bar model (the conclusion is inhabited by* $\lambda\alpha,\beta.\texttt{if}\ \alpha{=}\beta\ \texttt{then tt else ff}$*):*

$$\frac{}{H \vdash \mathbf{\Pi}\alpha,\beta{:}\texttt{Free}.\alpha =_{\mathcal{B}} \beta + \neg\alpha =_{\mathcal{B}} \beta}$$

5 The Law of Excluded Middle

This section demonstrates that OpenTT provides a key axiom from classical logic, namely the Law of Excluded Middle (LEM). Even though various other classical principles could be considered here (and will be considered in future work), we focus on LEM as it is the central axiom differentiating classical logic from intuitionistic logic. Thus, we show that in addition to capturing the intuitionistic concept of choice sequences, OpenTT also includes the following $\downarrow$-squashed version of LEM, called $\texttt{LEM}_\downarrow$, that is validated w.r.t. the open bar model: $\mathbf{\Pi}P{:}\mathbb{U}_i.\downarrow(P+\neg P)$.

For BITT, even this weak $\texttt{LEM}_\downarrow$ axiom, *that does not have any computational content* (as it is realized by $\lambda P.\star$), is inconsistent [10]. More precisely, $\neg\texttt{LEM}_\downarrow$ is valid w.r.t. the Beth metatheory presented in [10]. Intuitively, this is because $\texttt{LEM}_\downarrow$ states that there exists a bar of the current world such that either: (1) P is true at the bar, or (2) it is false in all extensions of the bar. This is false (i.e., the negation is true) because, for example, for $P = (\mathbf{\Sigma}n{:}\mathbb{N}.\eta(n) =_{\mathbb{N}} \underline{1})$, where η is a free choice sequence, (1) is false because η could be the sequence that never chooses 1, and (2) is false because there is an extension of the bar where η chooses 1. Stronger versions of this axiom, such as the non-$\downarrow$-squashed version, are therefore also false. This counterexample for BITT does not serve as a counterexample for OpenTT because given a world w it is always possible to find an extension where η eventually holds 1. Hence, OpenTT is more amenable to classical logic than theories based on standard Beth models, such as BITT. As illustrated in Prop. 22's proof below, intuitively, this is due to the fact that the open bar model implements a notion of time which allows to select futures (i.e., extensions), thereby allowing for some internal choices to be made.

▶ **Proposition 22.** *The following rule of OpenTT is valid w.r.t. the open bar model (using LEM in the metatheory).*

$$\frac{}{H \vdash \mathbf{\Pi}P{:}\mathbb{U}_i.\downarrow(P+\neg P)}$$

Proof. We have to show that for every world w' that extends the current world w, there exists a world w'' that extends w' such that $P+\neg P$ is inhabited in all extensions of w''. Let w' be an extension of w. We need to find a $w'' \geq w'$ that makes the above true. Using classical

[21] See rules/rules_choice2.v and rules/rules_choice5.v for further details.

logic we assume that $\exists_{\texttt{EXT}}(w', \lambda w''.\texttt{inh}(w'', P))$ is either true of false. If it is true, we obtain a $w'' \succeq w'$ at which P is inhabited, and we therefore conclude. Otherwise, we use w', which is a trivial extension of w'. We must now show that $P+\neg P$ is inhabited in all extensions of w'. We prove that it is inhabited by $\texttt{inr}(\star)$ by showing that in all $w'' \succeq w'$, P is not inhabited at w''. Assuming that P is inhabited at w'', we get that $\exists_{\texttt{EXT}}(w', \lambda w''.\texttt{inh}(w'', P))$ is true, which contradicts our assumption.[22] ◀

6 Conclusion and Related Work

The paper presents OpenTT, a novel intuitionistic type theory that features both a theory of choice sequences and a variant of the classical Law of Excluded Middle. This was made possible thanks to the open bar model, which internalizes a more relaxed notion of time than traditional Beth models that allows selecting futures. Thus, OpenTT provides a theoretical framework for studying the interplay between intuitionistic and classical logic.

Much work has been done on combining classical and constructive logics. One standard method is to use double negation translations [25] to embed classical logic in constructive logic. Another approach is to mix the two logics within the same framework. For example, PIL [37] mixes both logics through a polarization mechanism. Of particular relevance is Moschovakis's theory that includes choice sequences and is consistent with all classically true arithmetic sentences via a Kripke model [40].

As mentioned in the Introduction, there is a long line of work on providing intuitionistic counterexamples to classically valid axioms using variants of choice sequences. For example, in [16] Markov's Principle is proved to be false in a Martin-Löf type theory extended with a "generic" element, which behaves as a free choice sequence of Booleans. Since we have shown that OpenTT is compatible with a variant of LEM, we plan to investigate the status of other classically valid principles, such as Markov's Principle and the Axiom of Choice.

As for the open bar model, Kripke (and Beth) models are often used to model stateful theories. For example, in [36] the Kripke semantics of function types allows the returned values of functions to extend the state at hand. In contrast, the open bar model allows all computations to extend worlds. Other examples include [1; 2; 12; 11], where Kripke semantics are used to interpret theories with reference cells. We leave the study of other forms of stateful computations for future work.

Unlike Kripke models, Beth models can interpret formulas that only *eventually* hold. The notion of "eventuality" in the open bar model slightly differs from the one in Beth models, and as hinted at in Sec. 3, is related to the "possibility" operator of modal logic [35]. A formal study of these connections is left for future work.

Several forms of choice sequence axioms have been studied in the literature. Some of them are currently time or space squashed in OpenTT. We plan on exploring versions of these axioms that are "less squashed" in the sense that they have more computational content.

Finally, the comprehensive account of choice sequences in OpenTT also opens the door for the exploration of the computational implications of the existence of such entities. For one, Brouwer used choice sequences to define the constructive real numbers as sequences of nested rational intervals. The computational account of choice sequences in OpenTT provides a framework for the formalization of Brouwerian constructive real analysis, and then comparing it to the more standard formalizations.

[22] See rules/rules_classical.v for more details.

References

[1] Amal J. Ahmed, Andrew W. Appel and Roberto Virga. "A Stratified Semantics of General References Embeddable in Higher-Order Logic". In: *LICS*. IEEE Computer Society, 2002, p. 75. DOI: 10.1109/LICS.2002.1029818.

[2] Amal Jamil Ahmed. "Semantics of Types for Mutable State". PhD thesis. Princeton University, 2004.

[3] Stuart F. Allen. "A Non-Type-Theoretic Definition of Martin-Löf's Types". In: *LICS*. IEEE Computer Society, 1987, pp. 215–221.

[4] Stuart F. Allen. "A Non-Type-Theoretic Semantics for Type-Theoretic Language". PhD thesis. Cornell University, 1987.

[5] Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo and Evan Moran. "Innovations in computational type theory using Nuprl". In: *J. Applied Logic* 4.4 (2006). http://www.nuprl.org/, pp. 428–469.

[6] Abhishek Anand and Vincent Rahli. "Towards a Formally Verified Proof Assistant". In: *ITP 2014*. Vol. 8558. LNCS. Springer, 2014, pp. 27–44. DOI: 10.1007/978-3-319-08970-6_3.

[7] Mark van Atten. *On Brouwer*. Wadsworth Philosophers. Cengage Learning, 2004.

[8] Mark van Atten and Dirk van Dalen. "Arguments for the continuity principle". In: *Bulletin of Symbolic Logic* 8.3 (2002), pp. 329–347.

[9] Evert Willem Beth. *The foundations of mathematics: A study in the philosophy of science*. Harper and Row, 1966.

[10] Mark Bickford, Liron Cohen, Robert L. Constable and Vincent Rahli. "Computability Beyond Church-Turing via Choice Sequences". In: *LICS 2018*. ACM, 2018, pp. 245–254. DOI: 10.1145/3209108.3209200.

[11] Lars Birkedal, Bernhard Reus, Jan Schwinghammer, Kristian Støvring, Jacob Thamsborg and Hongseok Yang. "Step-indexed kripke models over recursive worlds". In: *POPL*. ACM, 2011, pp. 119–132. DOI: 10.1145/1926385.1926401.

[12] Lars Birkedal, Kristian Støvring and Jacob Thamsborg. "Realisability semantics of parametric polymorphism, general references and recursive types". In: *Mathematical Structures in Computer Science* 20.4 (2010), pp. 655–703. DOI: 10.1017/S0960129510000162.

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

[14] Venanzio Capretta. "A polymorphic representation of induction-recursion". www.cs.ru.nl/~venanzio/publications/induction_recursion.ps. 2004.

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

[16] Thierry Coquand and Bassel Mannaa. "The Independence of Markov's Principle in Type Theory". In: *FSCD 2016*. Vol. 52. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 17:1–17:18. DOI: 10.4230/LIPIcs.FSCD.2016.17.

[17] Thierry Coquand, Bassel Mannaa and Fabian Ruch. "Stack semantics of type theory". In: *LICS 2017*. IEEE Computer Society, 2017, pp. 1–11. DOI: 10.1109/LICS.2017.8005130.

[18] Karl Crary. "Type-Theoretic Methodology for Practical Programming Languages". PhD thesis. Ithaca, NY: Cornell University, Aug. 1998.

[19] Dirk van Dalen. "An interpretation of intuitionistic analysis". In: *Annals of mathematical logic* 13.1 (1978), pp. 1–43.

[20] Jacques Dubucs and Michel Bourdeau. *Constructivity and Computability in Historical and Philosophical Perspective*. Vol. 34. Jan. 2014. DOI: 10.1007/978-94-017-9217-2.

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

[22] Peter Dybjer. "A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory". In: *J. Symb. Log.* 65.2 (2000), pp. 525–549.

[23] VH Dyson and Georg Kreisel. *Analysis of Beth's semantic construction of intuitionistic logic*. Stanford University. Applied Mathematics and Statistics Laboratories, 1961.

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

[25] Gilda Ferreira and Paulo Oliva. “On Various Negative Translations”. In: *CL&C*. Vol. 47. EPTCS. 2010, pp. 21–33. DOI: 10.4204/EPTCS.47.4.

[26] Arend Heyting. *Intuitionism: an introduction*. North-Holland Pub. Co., 1956.

[27] Douglas J. Howe. “Equality in Lazy Computation Systems”. In: *LICS 1989*. IEEE Computer Society, 1989, pp. 198–203.

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

[29] Alexei Kopylov and Aleksey Nogin. “Markov’s Principle for Propositional Type Theory”. In: *CSL 2001*. Vol. 2142. LNCS. Springer, 2001, pp. 570–584. DOI: 10.1007/3-540-44802-0_40.

[30] Georg Kreisel. “A Remark on Free Choice Sequences and the Topological Completeness Proofs”. In: *J. Symb. Log.* 23.4 (1958), pp. 369–388. DOI: 10.2307/2964012.

[31] Georg Kreisel. “Lawless sequences of natural numbers”. In: *Compositio Mathematica* 20 (1968), pp. 222–248.

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

[33] Georg Kreisel and Anne S. Troelstra. “Formal systems for some branches of intuitionistic analysis”. In: *Annals of Mathematical Logic* 1.3 (1970), pp. 229–387. DOI: http://dx.doi.org/10.1016/0003-4843(70)90001-X.

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

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

[36] Paul Blain Levy. “Possible World Semantics for General Storage in Call-By-Value”. In: *CSL 2002*. Vol. 2471. LNCS. Springer, 2002, pp. 232–246. DOI: 10.1007/3-540-45793-3_16.

[37] Chuck Liang and Dale Miller. “Kripke semantics and proof systems for combining intuitionistic logic and classical logic”. In: *Ann. Pure Appl. Log.* 164.2 (2013), pp. 86–111. DOI: 10.1016/j.apal.2012.09.005.

[38] Joan R. Moschovakis. “An intuitionistic theory of lawlike, choice and lawless sequences”. In: *Logic Colloquium’90: ASL Summer Meeting in Helsinki*. Association for Symbolic Logic. 1993, pp. 191–209.

[39] Joan Rand Moschovakis. *Choice Sequences and Their Uses*. 2015.

[40] Joan Rand Moschovakis. “Intuitionistic Analysis at the End of Time”. In: *Bulletin of Symbolic Logic* 23.3 (2017), pp. 279–295. DOI: 10.1017/bsl.2017.25.

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

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

[43] Vincent Rahli, Liron Cohen and Mark Bickford. “A Verified Theorem Prover Backend Supported by a Monotonic Library”. In: *LPAR-22*. Vol. 57. EPiC Series in Computing. EasyChair, 2018, pp. 564–582.

[44] A. S. Troelstra. “Analysing choice sequences”. In: *J. Philosophical Logic* 12.2 (1983), pp. 197–260. DOI: 10.1007/BF00247189.

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

[46] A.S. Troelstra. *Choice Sequences: A Chapter of Intuitionistic Mathematics*. Clarendon Press, 1977.

[47] Anne S. Troelstra. “Choice Sequences and Informal Rigour”. In: *Synthese* 62.2 (1985), pp. 217–227.

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

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

[50] Beth E. W. “Semantic Construction of Intuitionistic Logic”. In: *Journal of Symbolic Logic* 22.4 (1957), pp. 363–365.

A OpenTT's Semantics

Sec. 3 provided part of OpenTT's semantics. We presented there the semantics of distinguishing features of OpenTT. Let us now present the rest of its semantics. As mentioned in Sec. 3, this semantics has been formalized in Coq, and can be found in per/per.v and per/nuprl.v. Moreover, as the Coq formalization follows a slightly different presentation (as mentioned in Sec. 3 it combines the inductive relation $T_1 {\equiv_w} T_2$ and the recursive function $t_1 {\equiv_w} t_2 {\in} T$ into a single inductive definition following the method described in [4; 14]). An inductive-recursive formalization of the open bar semantics of OpenTT in Agda can be found in Appx. D.

▶ **Definition 23** (Products). *Product types are interpreted as follows:*

$$\begin{aligned}&\mathbf{\Pi} x_1{:}A_1.B_1 {\equiv_w} \mathbf{\Pi} x_2{:}A_2.B_2\\ &\quad\iff \forall_{\texttt{EXT}}(w, \lambda w'.A_1 {\equiv_{w'}} A_2 \wedge \forall a_1, a_2.\ a_1 {\equiv_{w'}} a_2 {\in} A_1 \Rightarrow B_1[x_1\backslash a_1] {\equiv_{w'}} B_2[x_2\backslash a_2])\end{aligned}$$

$$f_1 {\equiv_w} f_2 {\in} \mathbf{\Pi} x{:}A.B \iff \mathcal{O}(w, \lambda w'.\forall a_1, a_2.\ a_1 {\equiv_{w'}} a_2 {\in} A \Rightarrow f_1(a_1) {\equiv_{w'}} f_2(a_2) {\in} B[x_1\backslash a_1])$$

▶ **Definition 24** (Sums). *Sum types are interpreted as follows:*

$$\begin{aligned}&\mathbf{\Sigma} x_1{:}A_1.B_1 {\equiv_w} \mathbf{\Sigma} x_2{:}A_2.B_2\\ &\quad\iff \forall_{\texttt{EXT}}(w, \lambda w'.A_1 {\equiv_{w'}} A_2 \wedge \forall a_1, a_2.\ a_1 {\equiv_{w'}} a_2 {\in} A_1 \Rightarrow B_1[x_1\backslash a_1] {\equiv_{w'}} B_2[x_2\backslash a_2])\end{aligned}$$

$$\begin{aligned}t_1 {\equiv_w} t_2 {\in} \mathbf{\Sigma} x{:}A.B \iff \mathcal{O}(w, \lambda w'.\exists a_1, a_2, b_1, b_2.\ & t_1 \Downarrow_{w'} \langle a_1, b_1\rangle \wedge t_2 \Downarrow_{w'} \langle a_2, b_2\rangle \\ & \wedge a_1 {\equiv_{w'}} a_2 {\in} A \wedge b_1 {\equiv_{w'}} b_2 {\in} B[x_1\backslash a_1])\end{aligned}$$

▶ **Definition 25** (Universes). *To interpret universes, we need to use parameterized (by a universe level) $T_1 {\equiv_{i,w}} T_2$ and $t_1 {\equiv_{i,w}} t_2 {\in} T$ relations instead of the ones used so far. We can then define $T_1 {\equiv_w} T_2$ as $\exists i.\ T_1 {\equiv_{i,w}} T_2$ and $t_1 {\equiv_w} t_2 {\in} T$ as $\exists i.\ t_1 {\equiv_{i,w}} t_2 {\in} T$. We do not present the full construction here as it is standard. However, let us point out that using the above definitions we can then interpret universes inductively over i, resulting in the following interpretations:*

$$\mathbb{U}_i {\equiv_{j,w}} \mathbb{U}_i \iff i < j \qquad\qquad T_1 {\equiv_{j,w}} T_2 {\in} \mathbb{U}_i \iff T_1 {\equiv_{j,w}} T_2$$

▶ **Definition 26** (Equality). *Equality types are interpreted as follows:*

$$(a_1 = a_2 \in A) {\equiv_w} (b_1 = b_2 \in B) \iff A {\equiv_w} B \wedge a_1 {\equiv_w} b_1 {\in} A \wedge a_2 {\equiv_w} b_2 {\in} A$$

$$t_1 {\equiv_w} t_2 {\in} (a = b \in A) \iff \mathcal{O}(w, \lambda w'.t_1 \Downarrow_{w'} \star \wedge t_2 \Downarrow_{w'} \star \wedge a {\equiv_{w'}} b {\in} A)$$

▶ **Definition 27** (Disjoint Union). *Disjoint union types are interpreted as follows:*

$$A_1{+}A_2 {\equiv_w} B_1{+}B_2 \iff A_1 {\equiv_w} B_1 \wedge A_2 {\equiv_w} B_2$$

$$\begin{aligned}t_1 {\equiv_w} t_2 {\in} A{+}B \iff \mathcal{O}(w, \lambda w'.&(\exists x, y.\ t_1 \Downarrow_{w'} \texttt{inl}(x) \wedge t_2 \Downarrow_{w'} \texttt{inl}(y) \wedge x {\equiv_{w'}} y {\in} A) \\ & \vee (\exists x, y.\ t_1 \Downarrow_{w'} \texttt{inr}(x) \wedge t_2 \Downarrow_{w'} \texttt{inr}(y) \wedge x {\equiv_{w'}} y {\in} B))\end{aligned}$$

▶ **Definition 28** (Sets). *Set types are interpreted as follows:*

$$\begin{aligned}&\{x_1 : A_1 \mid B_1\} {\equiv_w} \{x_2 : A_2 \mid B_2\}\\ &\quad\iff \forall_{\texttt{EXT}}(w, \lambda w'.A_1 {\equiv_{w'}} A_2 \wedge \forall a_1, a_2.\ a_1 {\equiv_{w'}} a_2 {\in} A_1 \Rightarrow B_1[x_1\backslash a_1] {\equiv_{w'}} B_2[x_2\backslash a_2])\end{aligned}$$

$$t_1 {\equiv_w} t_2 {\in} \{x : A \mid B\} \iff \mathcal{O}(w, \lambda w'.t_1 {\equiv_{w'}} t_2 {\in} A \wedge \texttt{inh}(w', B[x\backslash t_1]))$$

▶ **Definition 29** (Less Than). *Less than types are interpreted as follows:*

$$t_1 < t_2 \equiv_w u_1 < u_2 \iff t_1 \equiv_w u_1 \in \mathbb{N} \wedge t_2 \equiv_w u_2 \in \mathbb{N}$$

$$t_1 \equiv_w t_2 \in (u_1 < u2) \iff \mathcal{O}(w, \lambda w'.\exists k_1, k_2.\ t_1 \Downarrow_w \underline{k_1} \wedge t_2 \Downarrow_w \underline{k_2} \wedge k_1 < k_2)$$

The time squashing type $\wr T$ is defined using Howe's computational equivalence [27], which is omitted from this paper for space reasons (see [27] for a definition of this relation, as well as cequiv/cequiv.v). It turns out that OpenTT is not only closed under computation but more generally under Howe's computational equivalence $\sim$, which we have proved to be a congruence following Howe's method [27]. We define $t_1 \approx_w t_2$ as $\forall_{\texttt{EXT}}(w, \lambda w'.t_1 \sim_w t_2)$.

▶ **Definition 30** (Time Squashing). *Time squashing types are interpreted as follows:*

$$\wr T \equiv_w \wr U \iff T \equiv_w U \in \mathbb{N}$$

$$t_1 \equiv_w t_2 \in (\wr T) \iff \mathcal{O}(w, \lambda w'.\exists u_1, u_2.\ w \sim t_1 u_1 \wedge w \sim t_2 u_2 \wedge t_1 \approx_w t_2 \wedge u_1 \equiv_{w'} u_2 \in T)$$

B OpenTT's Inference Rules

In OpenTT, sequents are of the form $h_1, \ldots, h_n \vdash T \lfloor \textbf{ext}\ t \rfloor$. Such a sequent denotes that, assuming $h_1, \ldots, h_n$, the term t is a member of the type T, and that therefore T is a type. The term t in this context is called the *extract* or *evidence* of T. Extracts are sometimes omitted when irrelevant to the discussion. In particular, we typically do so when the conclusion T of a sequent is an equality type of the form $a = b \in A$, since equality types can only be inhabited by the constant $\star$, we then typically omit the extract in such sequents. An hypothesis h is of the form $x : A$, where the variable x stands for the name of the hypothesis and A its type. A rule is a pair of a conclusion sequent S and a list of premise sequents, $S_1, \cdots, S_n$ (written as usual using a fraction notation, with the premises on top). Let us now provide a sample of OpenTT's key inference rules for some of its types not discussed above. The reader is invited to check https://github.com/vrahli/NuprlInCoq/blob/ls3/ for a complete list of rules, as well as [15], from which OpenTT borrowed most of its rules for its standard types.

B.1 Products

The following rules are the standard **Π**-elimination rule, **Π**-introduction rule, type equality for **Π** types, and λ-introduction rule, respectively.

$$\frac{H, f : \mathbf{\Pi} x{:}A.B, J \vdash a \in A \quad H, f : \mathbf{\Pi} x{:}A.B, J, z : f(a) \in B[x\backslash a] \vdash C \lfloor \textbf{ext}\ e \rfloor}{H, f : \mathbf{\Pi} x{:}A.B, J \vdash C \lfloor \textbf{ext}\ e[z\backslash\star] \rfloor}$$

$$\frac{H, z : A \vdash B[x\backslash z] \lfloor \textbf{ext}\ b \rfloor \quad H \vdash A \in \mathbb{U}_i}{H \vdash \mathbf{\Pi} x{:}A.B \lfloor \textbf{ext}\ \lambda z.b \rfloor}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H, y : A_1 \vdash B_1[x_1\backslash y] = B_2[x_2\backslash y] \in \mathbb{U}_i}{H \vdash \mathbf{\Pi} x_1{:}A_1.B_1 = \mathbf{\Pi} x_2{:}A_2.B_2 \in \mathbb{U}_i}$$

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

Note that the last rule requires to prove that A is a type because the conclusion requires to prove that $\mathbf{\Pi} x{:}A.B$ is a type, and the first hypothesis only states that B is a type family over A, but does not ensures that A is a type.

The following rule is the standard function extensionality rule:

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

The following captures that PERs are closed under β-reductions:

$$\frac{H \vdash t[x\backslash s] = u \in T}{H \vdash (\lambda x.t)\ s = u \in T}$$

B.2 Sums

The following rules are the standard $\mathbf{\Sigma}$-elimination rule, $\mathbf{\Sigma}$-introduction rule, type equality for the $\mathbf{\Sigma}$ type, and pair-introduction rule, respectively.

$$\frac{H, p : \mathbf{\Sigma} x{:}A.B, a : A, b : B[x\backslash a], J[p\backslash\langle a,b\rangle] \vdash C[p\backslash\langle a,b\rangle]\ \lfloor \mathbf{ext}\ e \rfloor}{H, p : \mathbf{\Sigma} x{:}A.B, J \vdash C\ \lfloor \mathbf{ext}\ \mathtt{let}\ a, b = p\ \mathtt{in}\ e \rfloor}$$

$$\frac{H \vdash a \in A \quad H \vdash b \in B[x\backslash a] \quad H, z : A \vdash B[x\backslash z] \in \mathbb{U}_i}{H \vdash \mathbf{\Sigma} x{:}A.B\ \lfloor \mathbf{ext}\ \langle a, b\rangle \rfloor}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H, y : A_1 \vdash B_1[x_1\backslash y] = B_2[x_2\backslash y] \in \mathbb{U}_i}{H \vdash \mathbf{\Sigma} x_1{:}A_1.B_1 = \mathbf{\Sigma} x_2{:}A_2.B_2 \in \mathbb{U}_i}$$

$$\frac{H, z : A \vdash B[x\backslash z] \in \mathbb{U}_i \quad H \vdash a_1 = a_2 \in A \quad H \vdash b_1 = b_2 \in B[x\backslash a_1]}{H \vdash \langle a_1, b_1\rangle = \langle a_2, b_2\rangle \in \mathbf{\Sigma} x{:}A.B}$$

The following rule states that PERs are closed under spread-reductions:

$$\frac{H \vdash u[x\backslash s; y\backslash t] = t_2 \in T}{H \vdash \mathtt{let}\ x, y = \langle s, t\rangle\ \mathtt{in}\ u = t_2 \in T}$$

B.3 Equality

The following rules are the standard equality-introduction rule:[23], equality-elimination rule (which states that equality types are inhabited by the $\star$ constant), hypothesis rule, symmetry and transitivity rules, respectively.

$$\frac{H \vdash A = B \in \mathbb{U}_i \quad H \vdash a_1 = b_1 \in A \quad H \vdash a_2 = b_2 \in B}{H \vdash a_1 = a_2 \in A = b_1 = b_2 \in B \in \mathbb{U}_i}$$

$$\frac{H, z : a = b \in A, J[z\backslash\star] \vdash C[z\backslash\star]\ \lfloor \mathbf{ext}\ e \rfloor}{H, z : a = b \in A, J \vdash C\ \lfloor \mathbf{ext}\ e \rfloor}$$

$$\frac{}{H, x : A, J \vdash x \in A}$$

$$\frac{H \vdash b = a \in T}{H \vdash a = b \in T} \qquad \frac{H \vdash a = c \in T \quad H \vdash c = b \in T}{H \vdash a = b \in T}$$

[23] The actual rule is slightly more general as it allows a_1 and b_1 to be "computationally equivalent" (and similarly for a_2 and b_2). However, since we have not introduced this concept here, we present a simpler version of this rule only.

The following rule allows fixing the extract of a sequent:

$$\frac{H \vdash T \lfloor \textbf{ext } t \rfloor}{H \vdash t \in T}$$

The following rule allows rewriting with an equality in an hypothesis:

$$\frac{H, x : B, J \vdash C \lfloor \textbf{ext } t \rfloor \quad H \vdash A = B \in \mathbb{U}_i}{H, x : A, J \vdash C \lfloor \textbf{ext } t \rfloor}$$

B.4 Universes

Let i is a lower universe than j. The following rules are the standard universe-introduction rule and the universe cumulativity rule, respectively.

$$\frac{}{H \vdash \mathbb{U}_i = \mathbb{U}_i \in \mathbb{U}_j} \qquad \frac{H \vdash T \in \mathbb{U}_j}{H \vdash T \in \mathbb{U}_i}$$

B.5 Sets

The following rule is the standard set-elimination rule:

$$\frac{H, z : \{x : A \mid B\}, a : A, \boxed{b : B[x\backslash a]}, J[z\backslash a] \vdash C[z\backslash a] \lfloor \textbf{ext } e \rfloor}{H, z : \{x : A \mid B\}, J \vdash C \lfloor \textbf{ext } e[a\backslash z] \rfloor}$$

Note that we have used a new construct in the above rule, namely the hypothesis $\boxed{b : B[x\backslash a]}$, which is called a hidden hypothesis. The main feature of hidden hypotheses is that their names cannot occur in extracts (which is why we "box" those hypotheses). Intuitively, this is because the proof that B is true is discarded in the proof that the set type $\{x : A \mid B\}$ is true and therefore cannot occur in computations. Hidden hypotheses can be unhidden using the following rule:

$$\frac{H, x : T, J \vdash a = b \in A \lfloor \textbf{ext } \star \rfloor}{H, \boxed{x : T}, J \vdash a = b \in A \lfloor \textbf{ext } \star \rfloor}$$

which is valid since the extract is $\star$ and therefore does not make use of x.

The following rules are the standard set-introduction rule, type equality for the set type, and introduction rule for members of set types, respectively.

$$\frac{H \vdash a \in A \quad H \vdash B[x\backslash a] \quad H, z : A \vdash B[x\backslash z] \in \mathbb{U}_i}{H \vdash \{x : A \mid B\} \lfloor \textbf{ext } a \rfloor}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H, y : A_1 \vdash B_1[x_1\backslash y] = B_2[x_2\backslash y] \in \mathbb{U}_i}{H \vdash \{x_1 : A_1 \mid B_1\} = \{x_2 : A_2 \mid B_2\} \in \mathbb{U}_i}$$

$$\frac{H, z : A \vdash B[x\backslash z] \in \mathbb{U}_i \quad H \vdash a = b \in A \quad H \vdash B[x\backslash a]}{H \vdash a = b \in \{x : A \mid B\}}$$

B.6 Disjoint Unions

The following rules are the standard disjoint union-elimination rule, disjoint union-introduction rules, type equality for the disjoint union type, and injection-introduction rules, respectively.

$$\frac{\begin{array}{c} H, d : A{+}B, x : A, J[d\backslash \texttt{inl}(x)] \vdash C[d\backslash \texttt{inl}(x)] \lfloor \textbf{ext } t \rfloor \\ H, d : A{+}B, y : B, J[d\backslash \texttt{inr}(y)] \vdash C[d\backslash \texttt{inr}(y)] \lfloor \textbf{ext } u \rfloor \end{array}}{H, d : A{+}B, J \vdash C \lfloor \textbf{ext } \texttt{case } d \texttt{ of inl}(x) \Rightarrow t \mid \texttt{inr}(y) \Rightarrow u \rfloor}$$

$$\frac{H \vdash A \; \lfloor \textbf{ext } a \rfloor \quad H \vdash B \in \mathbb{U}_i}{H \vdash A{+}B \; \lfloor \textbf{ext } \texttt{inl}(a) \rfloor} \qquad \frac{H \vdash B \; \lfloor \textbf{ext } b \rfloor \quad H \vdash A \in \mathbb{U}_i}{H \vdash A{+}B \; \lfloor \textbf{ext } \texttt{inr}(b) \rfloor}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H \vdash B_1 = B_2 \in \mathbb{U}_i}{H \vdash A_1{+}B_1 = A_2{+}B_2 \in \mathbb{U}_i}$$

$$\frac{H \vdash a_1 = a_2 \in A \quad H \vdash B \in \mathbb{U}_i}{H \vdash \texttt{inl}(a_1) = \texttt{inl}(a_2) \in A{+}B} \qquad \frac{H \vdash b_1 = b_2 \in B \quad H \vdash A \in \mathbb{U}_i}{H \vdash \texttt{inr}(b_1) = \texttt{inr}(b_2) \in A{+}B}$$

The following rules state that PERs are closed under decide-reductions:

$$\frac{H \vdash t[x\backslash s] = t_2 \in T}{H \vdash (\texttt{case inl}(s) \texttt{ of inl}(x) \Rightarrow t \mid \texttt{inr}(y) \Rightarrow u) = t_2 \in T}$$

$$\frac{H \vdash u[y\backslash s] = t_2 \in T}{H \vdash (\texttt{case inr}(s) \texttt{ of inl}(x) \Rightarrow t \mid \texttt{inr}(y) \Rightarrow u) = t_2 \in T}$$

C Squashing

As mentioned in Sec. 2.4, OpenTT includes a *squashing* mechanism, which is used to erase the computational content of a type by turning its PER into a trivial one.[24] More precisely, given a type T, the type $\downarrow T$, defined as $\{x : \texttt{True} \mid T\}$, is true iff T is true. However, while the type T might have a trivial PER, i.e., it might be inhabited by arbitrarily complex programs, $\downarrow T$ can only be inhabited by $\star$, which is $\texttt{True}$'s only inhabitant. Indeed, as shown in Def. 28 and Appx. B.5, a member of $\{x : \texttt{True} \mid T\}$ is a member of $\texttt{True}$, such that T is true. However, T's realizer is thrown away and is not part of $\{x : \texttt{True} \mid T\}$'s realizer.

More precisely, one can derive $\downarrow T$ from T because given a member t of T, one can trivially show that that $\star$ is a member of $\downarrow T$. We can capture this by the following derived rule:

$$\frac{H \vdash T \; \lfloor \textbf{ext } t \rfloor}{H \vdash \downarrow T \; \lfloor \textbf{ext } \star \rfloor}$$

However, the opposite is not true in general. One cannot in general derive T from $\downarrow T$ because given the realizer $\star$ of $\downarrow T$, it is not always possible to recover a realizer of T. We can capture this by the following derived rule:

$$\frac{H, z : \downarrow T, \boxed{x : T}, J[z\backslash\star] \vdash C[z\backslash\star] \; \lfloor \textbf{ext } e \rfloor}{H, z : \downarrow T, J \vdash C \; \lfloor \textbf{ext } e \rfloor}$$

To illustrate the point that we cannot in general derive T from $\downarrow T$, let us see how far we can go when trying to prove:

$$x : \downarrow T \vdash T$$

Using the above squash-elimination derived rule, we have to prove:

$$x : \downarrow T, \boxed{z : T} \vdash T$$

However, we are now stuck, as we have in general no way of deriving an extract of T given these hypotheses. The unhiding rule mentioned Appx. B.5 can only be used when the conclusion is an equality type, and the hypothesis rule mentioned in Appx. B.3, requires the z hypothesis to be "visible" (not hidden) in order to use z as a realizer of the conclusion.

[24] See for example [41] for more details on squashing.

D Semantics of OpenTT in Agda

Let us now provides a formalization of the open bar semantics of OpenTT in Agda. The code provided in this section can also be found here: agda/worldi.lagda

We first postulate and define enough about worlds to interpret OpenTT w.r.t. open bars.

```
postulate
  - Worlds
  world : Set
  - w2 extends w1
  _≥_ : (w2 : world) → (w1 : world) → Set
  - extension is reflexive
  extRefl : ∀ w → w ≥ w
  - extension is transitive
  extTrans : ∀ {w3 w2 w1} (e2 : w3 ≥ w2) (e1 : w2 ≥ w1) → w3 ≥ w1

- f holds in all extensions
allW : ∀ (w : world) (f : ∀ (w' : world) (e : w' ≥ w) → Set) → Set
allW w f = ∀ (w' : world) (e : w' ≥ w) → f w' e

- f holds in one extensions
exW : ∀ (w : world) (f : ∀ (w' : world) (e : w' ≥ w) → Set) → Set
exW w f = ∃ world (λ w' → ∃ (w' ≥ w) (λ e → f w' e))

- f holds in an open bar
inOpenBar : ∀ (w : world) (f : ∀ (w' : world) (e : w' ≥ w) → Set) → Set
inOpenBar w f =
  allW w (λ w1 e1 → exW w1 (λ w2 e2 → allW w2 (λ w3 e3 →
    f w3 (extTrans e3 (extTrans e2 e1)))))

- f holds in an open bar that depends on another open bar h
inOpenBar' : ∀ w {g} (h : inOpenBar w g) (f : ∀ w' (e : w' ≥ w) (x : g w' e) → Set) → Set
inOpenBar' w h f =
  allW w (λ w0 e0 →
            let p = h w0 e0 in
            let w1 = proj₁ p in
            let e1 = proj₁ (proj₂ p) in
            let q = proj₂ (proj₂ p) in
            exW w1 (λ w2 e2 → allW w2 (λ w3 e3 →
              let e' = extTrans e3 e2 in
              f w3 (extTrans e' (extTrans e1 e0)) (q w3 e'))))
```

We now define part of OpenTT's syntax and postulate its operational semantics.

```
postulate
  choice_sequence_name : Set

data Var : Set where
  var : ℕ → Var - variable are simply numbers

data Term : Set where
  - Numbers
  NAT : Term
  QNAT : Term
  LT : Term → Term → Term
  QLT : Term → Term → Term
  NUM : ℕ → Term
  - Products
  PI : Term → Var → Term → Term
  LAMBDA : Var → Term → Term
  APPLY : Term → Term → Term
  - Sums
  SUM : Term → Var → Term → Term
  PAIR : Term → Term → Term
  SPREAD : Term → Var → Var → Term
  - Sets -- they don't have constructors/destructors
  SET : Term → Var → Term → Term
  - Disjoint unions
  UNION : Term → Term → Term
  INL : Term → Term
  INR : Term → Term
  DECIDE : Term → Var → Term → Var → Term
  - Equality
  EQ : Term → Term → Term → Term
  AX : Term
  - Choice sequences
  FREE : Term
  CS : choice_sequence_name → Term
  - Time squashing
  TSQUASH : Term → Term
  - Free from definitions
  FFDEFS : Term → Term → Term
  - Universes
  UNIV : ℕ → Term

postulate
  - standard substitution function on terms
  subst : Var → Term → Term → Term
  - operational semantics of the language
  _⇓_at_ : ∀ (T T' : Term) (w : world) → Set
  - 'computes to' is reflexive
  compRefl : ∀ (T : Term) (w : world) → T ⇓ T at w
  - Howe's computational equivalence relation
  _~_at_ : ∀ (T T' : Term) (w : world) → Set
  - states that the argument does not contain any definition or choice sequence
  nodefs : Term → Set
infix 30 _⇓_at_
infix 30 _~_at_

- T computes to T' in all extensions of w
_⇓_at_ : ∀ (T T' : Term) (w : world) → Set
T ⇓ T' at w = allW w (λ w' _ → T ⇓ T' at w')
infix 30 _⇓_at_

- T computationally equivalent to T' in all extensions of w
_≈_at_ : ∀ (T T' : Term) (w : world) → Set
T ≈ T' at w = allW w (λ w' _ → T ~ T' at w')
infix 30 _≈_at_

compAllRefl : ∀ (T : Term) (w : world) → T ⇓ T at w
compAllRefl T w = λ w' e → compRefl T w'

- t1 and t2 compute to the same number and stay the same number in all extensions
strongMonEq : ∀ (w : world) (t1 t2 : Term) → Set
strongMonEq w t1 t2 = ∃ ℕ (λ n → t1 ⇓ (NUM n) at w × t2 ⇓ (NUM n) at w)

- t1 and t2 compute to the same number but that number can change over time
weakMonEq : ∀ (w : world) (t1 t2 : Term) → Set
weakMonEq w t1 t2 = allW w (λ w' _ → ∃ ℕ (λ n → t1 ⇓ (NUM n) at w' × t2 ⇓ (NUM n) at w'))
```

We now provide an inductive-recursive realizability semantics of OpenTT.

```
- PERs and world dependent PERs
per : Set₁
per = Term → Term → Set

wper : Set₁
wper = world → per

- eqTypes and eqInType provide meaning to types w.r.t.  already interpreted universes,
- given by univs (1st conjunct defines the equality between such universes, while the
- second conjunct defines the equality in such universes)
univs : Set₁
univs = ∃ ℕ (λ n → wper × wper)
```

```

- equality between types (an inductive definition)
- and equality in types (a recursive function)
data eqTypes (u : univs) (w : world) (T1 T2 : Term) : Set
eqInType : (u : univs) (w : world) {T1 T2 : Term} → (eqTypes u w T1 T2) → per
```

Equality between type is defined as the following inductive definition

```
data eqTypes u w T1 T2 where
  EQTNAT : T1 ⇓ NAT at w → T2 ⇓ NAT at w → eqTypes u w T1 T2
  EQTQNAT : T1 ⇓ QNAT at w → T2 ⇓ QNAT at w → eqTypes u w T1 T2
  EQTLT : (a1 a2 b1 b2 : Term)
    → T1 ⇓ (LT a1 b1) at w
    → T2 ⇓ (LT a2 b2) at w
    → strongMonEq w a1 a2
    → strongMonEq w b1 b2
    → eqTypes u w T1 T2
  EQTQLT : (a1 a2 b1 b2 : Term)
    → T1 ⇓ (QLT a1 b1) at w
    → T2 ⇓ (QLT a2 b2) at w
    → weakMonEq w a1 a2
    → weakMonEq w b1 b2
    → eqTypes u w T1 T2
  EQTFREE : T1 ⇓ FREE at w → T2 ⇓ FREE at w → eqTypes u w T1 T2
  EQTPI : (A1 B1 A2 B2 : Term) (v1 v2 : Var)
    → T1 ⇓ (PI A1 v1 B1) at w
    → T2 ⇓ (PI A2 v2 B2) at w
    → (eqta : allW w (λ w' _ → eqTypes u w' A1 A2))
    → (eqtb : allW w (λ w' e → ∀ a1 a2 → eqInType u w' (eqta w' e) a1 a2
                              → eqTypes u w' (subst v1 a1 B1) (subst v2 a2 B2)))
    → eqTypes u w T1 T2
  EQTSUM : (A1 B1 A2 B2 : Term) (v1 v2 : Var)
    → T1 ⇓ (SUM A1 v1 B1) at w
    → T2 ⇓ (SUM A2 v2 B2) at w
    → (eqta : allW w (λ w' _ → eqTypes u w' A1 A2))
    → (eqtb : allW w (λ w' e → ∀ a1 a2 → eqInType u w' (eqta w' e) a1 a2
                              → eqTypes u w' (subst v1 a1 B1) (subst v2 a2 B2)))
    → eqTypes u w T1 T2
  EQTSET : (A1 B1 A2 B2 : Term) (v1 v2 : Var)
    → T1 ⇓ (SET A1 v1 B1) at w
    → T2 ⇓ (SET A2 v2 B2) at w
    → (eqta : allW w (λ w' _ → eqTypes u w' A1 A2))
    → (eqtb : allW w (λ w' e → ∀ a1 a2 → eqInType u w' (eqta w' e) a1 a2
                              → eqTypes u w' (subst v1 a1 B1) (subst v2 a2 B2)))
    → eqTypes u w T1 T2
  EQTEQ : (a1 b1 a2 b2 A B : Term)
    → T1 ⇓ (EQ a1 a2 A) at w
    → T2 ⇓ (EQ b1 b2 B) at w
    → (eqtA : allW w (λ w' _ → eqTypes u w' A B))
    → (eqt1 : allW w (λ w' e → eqInType u w' (eqtA w' e) a1 b1))
    → (eqt2 : allW w (λ w' e → eqInType u w' (eqtA w' e) a2 b2))
    → eqTypes u w T1 T2
  EQTUNION : (A1 B1 A2 B2 : Term)
    → T1 ⇓ (UNION A1 B1) at w
    → T2 ⇓ (UNION A2 B2) at w
    → (eqtA : allW w (λ w' _ → eqTypes u w' A1 A2))
    → (eqtB : allW w (λ w' _ → eqTypes u w' B1 B2))
    → eqTypes u w T1 T2
  EQTSQUASH : (A1 A2 : Term)
    → T1 ⇓ (TSQUASH A1) at w
    → T2 ⇓ (TSQUASH A2) at w
    → (eqtA : allW w (λ w' _ → eqTypes u w' A1 A2))
    → eqTypes u w T1 T2
  EQFFDEFS : (A1 A2 x1 x2 : Term)
    → T1 ⇓ (FFDEFS A1 x1) at w
    → T2 ⇓ (FFDEFS A2 x2) at w
    → (eqtA : allW w (λ w' _ → eqTypes u w' A1 A2))
    → (eqx : allW w (λ w' e → eqInType u w' (eqtA w' e) x1 x1))
    → eqTypes u w T1 T2
  EQTUNIV : proj₁ (proj₂ u) w T1 T2 → eqTypes u w T1 T2
  EQTBAR : inOpenBar w (λ w' _ → eqTypes u w' T1 T2) → eqTypes u w T1 T2
```

Equality in types is defined as the following recursive function.

```
eqInType _ w (EQTNAT _ _) t1 t2 = inOpenBar w (λ w' _ → strongMonEq w' t1 t2)
eqInType _ w (EQTQNAT _ _) t1 t2 = inOpenBar w (λ w' _ → weakMonEq w' t1 t2)
eqInType _ w (EQTLT a1 _ b1 _ _ _ _ _ _) t1 t2 =
  inOpenBar w (λ w' _ → ∃ ℕ (λ n → ∃ ℕ (λ m → t1 ⇓ (NUM n) at w' × t2 ⇓ (NUM m) at w' × n < m)))
eqInType _ w (EQTQLT a1 _ b1 _ _ _ _ _ _) t1 t2 =
  inOpenBar w (λ w' _ → ∃ ℕ (λ n → ∃ ℕ (λ m → t1 ⇓ (NUM n) at w' × t2 ⇓ (NUM m) at w' × n < m)))
eqInType _ w (EQTFREE _ _) t1 t2 =
  inOpenBar w (λ w' _ → ∃ choice_sequence_name (λ n → t1 ⇓ (CS n) at w' × t2 ⇓ (CS n) at w'))
eqInType u w (EQTPI _ _ _ _ _ _ _ _ _ _ eqta eqtb) f1 f2 =
  inOpenBar w (λ w' e → ∀ (a1 a2 : Term) (eqa : eqInType u w' (eqta w' e) a1 a2)
                     → eqInType u w' (eqtb w' e a1 a2 eqa) (APPLY f1 a1) (APPLY f2 a2))
eqInType u w (EQTSUM _ _ _ _ _ _ _ _ _ _ eqta eqtb) t1 t2 =
  inOpenBar w (λ w' e → ∃ Term (λ a1 → ∃ Term (λ a2 → ∃ Term (λ b1 → ∃ Term (λ b2 →
                         ∃ (eqInType u w' (eqta w' e) a1 a2) (λ ea →
                         t1 ⇓ (PAIR a1 b1) at w'
                         × t2 ⇓ (PAIR a2 b2) at w'
                         × eqInType u w' (eqtb w' e a1 a2 ea) b1 b2))))))
eqInType u w (EQTSET _ _ _ _ _ _ _ _ _ _ eqta eqtb) t1 t2 =
  inOpenBar w (λ w' e → ∃ Term (λ b → ∃ (eqInType u w' (eqta w' e) t1 t2) (λ ea →
                         eqInType u w' (eqtb w' e t1 t2 ea) b b)))
eqInType u w (EQTEQ a1 b1 _ _ _ _ _ _ _ eqtA eqt1 eqt2) t1 t2 =
  inOpenBar w (λ w' e → t1 ⇓ AX at w' × t2 ⇓ AX at w' × eqInType u w' (eqtA w' e) a1 b1)
eqInType u w (EQTUNION _ _ _ _ _ _ _ eqtA eqtB) t1 t2 =
  inOpenBar w (λ w' e → ∃ Term (λ a → ∃ Term (λ b →
                 (t1 ⇓ (INL a) at w' × t2 ⇓ (INR b) at w' × eqInType u w' (eqtA w' e) a b)
                 ⊎
                 (t1 ⇓ (INR a) at w' × t2 ⇓ (INR b) at w' × eqInType u w' (eqtB w' e) a b))))
eqInType u w (EQTSQUASH _ _ _ _ eqtA) t1 t2 =
  inOpenBar w (λ w' e → ∃ Term (λ a1 → ∃ Term (λ a2 →
                 (t1 ∼ a1 at w') × (t2 ∼ a2 at w') × (t1 ≈ t2 at w')
                 × eqInType u w' (eqtA w' e) a1 a2)))
eqInType u w (EQFFDEFS _ _ x1 _ _ _ eqtA _) t1 t2 =
  inOpenBar w (λ w' e → ∃ Term (λ x →
                (t1 ⇓ AX at w') × (t2 ⇓ AX at w')
                × eqInType u w' (eqtA w' e) x1 x × nodefs x))
eqInType u w (EQTUNIV _) T1 T2 = proj₂ (proj₂ u) w T1 T2
eqInType u w (EQTBAR f) t1 t2 =
  {- inOpenBar' w f (λ w' _ (x :  eqTypes u w' _ _) → eqInType u w' x t1 t2)-}
  {- This is an unfolding of the above, as agda doesn't like the above -}
  allW w (λ w0 e0 →
           let p = f w0 e0 in
           let w1 = proj₁ p in
           let e1 = proj₁ (proj₂ p) in
           let q = proj₂ (proj₂ p) in
           exW w1 (λ w2 e2 → allW w2 (λ w3 e3 → eqInType u w3 (q w3 (extTrans e3 e2)) t1 t2)))
```

We finally close the construction as follows:

```
- Two level-m universes are equal if they compute to (UNIV m)
eqUnivi : (m : ℕ) → wper
eqUnivi m w T1 T2 = inOpenBar w (λ w' _ → T1 ⇓ (UNIV m) at w' × T2 ⇓ (UNIV m) at w')

- Two terms are equal in universe m if they are equal according to eqTypes
eqInUnivi : (m : ℕ) → wper
eqInUnivi 0 = λ _ _ _ → ⊥
eqInUnivi (suc m) w T1 T2 = eqTypes (m , (eqUnivi m , eqInUnivi m)) w T1 T2 ⊎ eqInUnivi m w T1 T2

uni : ℕ → univs
uni n = (n , (eqUnivi n , eqInUnivi n))

- Finally, the 'equal types' and 'equal in types' relations
```

```
eqtypes : (w : world) (T1 T2 : Term) → Set
eqtypes w T1 T2 = ∃ ℕ (λ n → eqTypes (uni n) w T1 T2)

eqintype : (w : world) (T a b : Term) → Set
eqintype w T a b = ∃ ℕ (λ n → ∃ (eqTypes (uni n) w T T) (λ p → eqInType (uni n) w p a b))
```