Guarded Computational Type Theory

Jonathan Sterling[*]
Carnegie Mellon University

Robert Harper[†]
Carnegie Mellon University

April 23, 2018

Abstract

Nakano's *later* modality can be used to specify and define recursive functions which are causal or synchronous; in concert with a notion of clock variable, it is possible to also capture the broader class of productive (co)programs. Until now, it has been difficult to combine these constructs with dependent types in a way that preserves the operational meaning of type theory and admits a hierarchy of universes $\mathtt{U}_i$.

We present an operational account of guarded dependent type theory with clocks called **CTT**$_\odot$, featuring a novel clock intersection connective $\{k \div \mathtt{clk}\} \to A$ that enjoys the clock irrelevance principle, as well as a predicative hierarchy of universes $\mathtt{U}_i$ which does not require any indexing in clock contexts. **CTT**$_\odot$ is simultaneously a programming language with a rich specification logic, as well as a computational metalanguage that can be used to develop semantics of other languages and logics.

1 Introduction

In a functional programming language, every definable function is continuous in the following sense: each finite quantity of output is induced by some finite quantity of input. To make this more precise, if we consider the case of stream transformers $F : \mathbb{S} \to \mathbb{S}$, we can see that finite prefixes of the output depend only on finite prefixes of the input:

$$\forall \alpha : \mathbb{S}.\ \forall i : \mathbb{N}.\ \exists n : \mathbb{N}.\ \forall \beta : \mathbb{S}.\ \alpha \equiv_n \beta \Rightarrow F(\alpha)_i \equiv F(\beta)_i \tag{1}$$

From a programming perspective, this can be rephrased in terms of *reads* and *writes*: for each write, the program is permitted to perform a finite but unbounded number of reads.

[*]jmsterli@cs.cmu.edu
[†]rwh@cs.cmu.edu

Causality Another possible class of functionals are the ones that can be implemented by a program which performs at most one read for every write. These are called the *causal* functionals, and in the case of stream transformers, they are characterized by the following causality principle:

$$\forall \alpha : \mathbb{S}.\ \forall i : \mathbb{N}.\ \forall \beta : \mathbb{S}.\ \alpha \equiv_i \beta \Rightarrow F(\alpha)_i \equiv F(\beta)_i \tag{2}$$

In other words, causal programs are the ones whose reads and writes proceed in lock-step. While we can surely carve out this class of functionals using predicates like (2) above, it is actually possible to define a new notion of stream $\mathbb{S}_{\blacktriangleright}$ such that all functionals $F : \mathbb{S}_{\blacktriangleright} \to \mathbb{S}_{\blacktriangleright}$ are *automatically* causal in the sense of (2). This kind of stream is called a "guarded stream", and we will use the term "sequence" to refer to ordinary streams.

Whereas ordinary streams or sequences are usually formed as the greatest solution to the isomorphism $\mathbb{S} \cong \mathbb{N} \times \mathbb{S}$, the guarded streams are formed using a special "later modality" $\blacktriangleright$ due to Nakano,[1] solving the isomorphism $\mathbb{S}_{\blacktriangleright} \cong \mathbb{N} \times \blacktriangleright\mathbb{S}_{\blacktriangleright}$. Modalities of this kind usually enjoy at least the following principles:

$$A \to \blacktriangleright A \qquad \blacktriangleright(A \times B) \cong (\blacktriangleright A \times \blacktriangleright B) \qquad \blacktriangleright(A \to B) \to (\blacktriangleright A \to \blacktriangleright B) \qquad (\blacktriangleright A \to A) \to A$$

The ratio of reads and writes specified in the type of a stream transformer can be modulated by adjusting the number of later modalities in the input and the output to the function.

Nakano's modality in semantics What is remarkable about Nakano's later modality is that fixed points for functions $F : \blacktriangleright A \to A$ always exist, without placing any restriction on F (such as monotonicity or positivity). Applied within a type-theoretic metalanguage, then, the later modality induces solutions to recursive domain equations which are not set-theoretically interpretable, such as the following classic definition of semantic types for a programming language with mutable store [Appel et al., 2007, Birkedal et al., 2011]:

$$\mathtt{Type} \cong \left(\mathtt{Loc} \xrightarrow{fin} \blacktriangleright\mathtt{Type}\right) \to \mathcal{P}(\mathtt{Val})$$

The later modality captures and internalizes the basic features of less abstract techniques like step-indexing, enabling more streamlined definitions and proofs that elide the bureaucratic performance of explicit indexing and monotonicity obligations. Today, modalities of this kind are of the essence for modern program logics like **Iris** [Jung et al., 2015].

Programming applications The fact that functions $F : \blacktriangleright A \to A$ always have fixed points has beneficial consequences for the practice of (total) functional programming on infinite data. In particular, clumsy syntactic guardedness conditions which ensure productivity (such as those used in Coq [The Coq Development Team, 2016], Agda [Norell, 2009] and Idris [Brady, 2013]) can be replaced with type structure, enabling more compositional styles of programming.[2]

[1] The notation $\bullet$ was originally used in Nakano [2000].

[2] A very closely related idea, sized types, has been deployed in the Agda proof assistant for exactly this purpose [Vezzosi, 2015].

$$\frac{\Delta, \kappa; \Gamma \vdash e : A}{\Delta; \Gamma \vdash \Lambda\kappa.\, e : \forall\kappa.\, A} \qquad \frac{\Delta; \Gamma \vdash e : \forall\kappa.\, A \quad \kappa' \in \Delta \quad \kappa' \notin \mathbf{FreeClocks}(A)}{\Delta; \Gamma \vdash e[\kappa'] : A[\kappa \hookrightarrow \kappa']} \qquad \frac{\Delta; \Gamma \vdash e : A}{\Delta; \Gamma \vdash \mathtt{pure}(e) : \blacktriangleright_\kappa A}$$

$$\frac{\Delta; \Gamma \vdash f : \blacktriangleright_\kappa (A \to B) \quad \Delta; \Gamma \vdash e : \blacktriangleright_\kappa A}{\Delta; \Gamma \vdash f \circledast e : \blacktriangleright_\kappa B} \qquad \frac{\Delta; \Gamma \vdash e : \forall\kappa.\, \blacktriangleright_\kappa A}{\Delta; \Gamma \vdash \mathtt{force}(e) : \forall\kappa.\, A} \qquad \frac{\Delta; \Gamma \vdash f : \blacktriangleright_\kappa A \to A}{\Delta; \Gamma \vdash \mathtt{fix}(f) : A}$$

$$(\forall\kappa.\, A) \equiv A \quad (\kappa \notin \mathbf{FreeClocks}(A)) \qquad \forall\kappa.\, A \times B \equiv (\forall\kappa.\, A) \times (\forall\kappa.\, B)$$

Figure 1: Selection of rules from Atkey and McBride [2013].

However, the later modality is too restrictive to be used on its own, because it rules out the functions which are not causal; but acausal functions on infinite data are perfectly sensible, and are very common in the real world. Consider, for instance, the function which drops every second element from a stream! To define this function, one would need a way to delete the modality; but without suitable restrictions, such an elimination principle would trivialize the modality and render it useless.

To resolve this problem, Atkey and McBride have introduced a notion of abstract clock κ to represent "time streams" together with universal quantification $\forall\kappa$ over clocks, replacing Nakano's modality with a clock-indexed family of modalities $\blacktriangleright_\kappa$ [Atkey and McBride, 2013].

Defining the type of κ-guarded streams as the solution to the equation $\mathbb{S}_\kappa \equiv \mathbb{N} \times \blacktriangleright_\kappa \mathbb{S}_\kappa$, it is possible to define the acausal function that drops every other element of a stream, with type $(\forall\kappa.\, \mathbb{S}_\kappa) \to (\forall\kappa.\, \mathbb{S}_\kappa)$. The reason that this is possible is that their calculus exhibits the isomorphism $(\forall\kappa.\, \blacktriangleright_\kappa A) \cong (\forall\kappa.\, A)$, as well as a *clock irrelevance* principle: $(\forall\kappa.\, A) \equiv A$ assuming that κ is not free in A; we summarize the constructs of this calculus in Figure 1.

1.1 Dependent type theory and guarded recursion

It has been surprisingly difficult to cleanly extend the account of guarded recursion with clocks to a full-spectrum dependently typed programming language which enjoys any combination of the following properties:

1. *Computational canonicity:* any closed element of type `bool` computes to either `tt` or `ff`.

2. *Simple universes:* a single predicative and cumulative hierarchy of universes $\mathtt{U}_i$ closed under base types, dependent function types, dependent pair types, lower universes, **later modalities** and **clock quantifiers**.

3. *Clock irrelevance:* if k is not mentioned in A and A is a type, then $\forall k.\, A$ is equal to

A.[3]

However, a dependent type theory with support for guarded recursion and clocks is desirable for multiple reasons; here, we have focused on causality as a useful construct for developing types qua behavioral specifications on program behavior, but there is also the potential to use such a dependent type theory as a computational metalanguage for developing and proving the semantics of other languages and logics, vaporizing the highly-bureaucratic step-indexed Kripke Logical Relations which usually must be employed.

The latter perspective is elaborated in the context of guarded dependent type theory without clocks in Paviotti et al. [2015] as well as Bizjak et al. [2014], and we anticipate that the addition of clocks will enable further developments along these lines.

1.2 Guarded Computational Type Theory

We contribute a new extensional and behavioral dependent type theory $\mathbf{CTT}_{\circledcirc}$ (pronounced "Guarded Computational Type Theory") for guarded recursion and clocks in the Nuprl tradition [Allen et al., 2006], enjoying the following characteristics:

1. Operational semantics and an immediate canonicity result at base types.
2. A clock-indexed later modality $\blacktriangleright_k A$ which requires no special syntax for introduction or destruction.
3. A decomposition of the clock quantifier from Bizjak and Møgelberg [2017] into a parametric part $\{k \div \mathtt{clk}\} \to A$ and a non-parametric part $(k : \mathtt{clk}) \to A$. The former is an intersection, and enjoys the crucial clock irrelevance principle; the latter is the cartesian product of a clock-indexed family of sets (right adjoint to weakening).
4. A guarded fixed point combinator which can be assigned the type $(\blacktriangleright_k A \to A) \to A$.
5. A predicative hierarchy of universes $\mathtt{U}_i$ closed under all the connectives, free of indexing by clock contexts.

Our operational account and canonicity result (Theorem 20) means that $\mathbf{CTT}_{\circledcirc}$ can be regarded simultaneously as a programming language with a rich specification logic, *and* as a computational metalanguage for developing operational and denotational semantics of other languages and logics.

Coq formalization and synthetic approach

Using the Coq proof assistant, we have formalized the fragment of our type theory that contains universes, dependent function and pair types, booleans, the later modality, and the two clock quantifiers (intersection and product); the full Coq development is available in Sterling and Harper [2018]. Throughout this paper, theorems and rules will be related to their Coq analogues using a reference like `Module.theorem_name`.

[3] Depending on the specific type theory, it may be desirable to realize this principle either as an isomorphism or as a definitional equality.

The principal difference between our informal presentation and the Coq formalization is that in the formalization of the formal term language, we use De Bruijn indices for both variables and clock names, whereas here we use concrete names for readability. This simplified the lemmas that we needed to prove about syntax, and about the elaboration of formal terms into programs.

We have used Coq's type theory as a proxy for the internal language of the presheaf topos that we develop herein, axiomatizing in Coq whatever objects and principles come not from the standard type theoretic constructions, but are instead imported into the system via forcing. The entire construction of **CTT**$_{\circleddash}$, then, is carried out within the internal language of the topos, an anti-bureaucratic measure which has made an otherwise daunting formalization effort feasible.

The idea of developing operational models of programming languages within the internal language of a topos is not new; see for instance Staton [2007], Bizjak et al. [2014] and Paviotti et al. [2015]. However, we believe that ours is the first instance of this technique being applied toward the development of semantics for a full-spectrum dependent type theory.

Acknowledgments

We are thankful to Carlo Angiuli, Lars Birkedal, Aleš Bizjak, Jonas Frey, Daniel Gratzer, Adrien Guatto, Pieter Hofstra, Bas Spitters, Sam Staton, and Joseph Tassarotti for helpful discussions on the semantics of guarded recursion, clock names and universe hierarchies. Thanks to David Christiansen for his comments on a draft of this paper.

The authors gratefully acknowledge the support of the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR.

2 Programming in CTT$_{\circleddash}$

Following the *computational meaning-theoretic* tradition initiated by Martin-Löf [1979], and developed further in the Nuprl project [Allen et al., 2006], we build Guarded Computational Type theory (**CTT**$_{\circleddash}$) on the basis of an untyped programming language, whose syntax is summarized in Figure 2.

In this paper, we distinguish between the syntax of `formal terms` and the language of **programs**; formal terms are used by clients of a formalism for type theory, whereas programs are the things which are actually endowed with operational meaning. For many languages, the difference between formal terms and programs is not so great, but for us the difference is essential; to avoid confusion, we distinguish between these levels using colors.

Formal Terms The grammar includes operators for both terms and types, which are not distinguished syntactically in any way. Typehood, equality and type membership are *semantic* properties which will be imposed after we propound the meaning explanation in Section 3.6. We include syntax for dependent function types $(x : A) \to B$, dependent

$$
\begin{array}{rcll}
k & ::= & k & \text{(clocks)} \\
M, A & ::= & x \mid \lambda x.\, M \mid \lambda k.\, M \mid M\, N \mid M\, k \mid \langle M, N \rangle \mid M.1 \mid M.2 & \text{(terms)} \\
& & \mathtt{fix}\, x\, \mathtt{in}\, M \mid \bigstar \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}(M; N; O) & \\
& & \mathtt{ze} \mid \mathtt{su}(M) \mid \mathtt{ifze}(M; N; x.\, O) \mid \mathtt{sup}(M; x.N) \mid \mathtt{rec}_{\mathtt{W}}(M; x, y, z.N) & \\
& & (x : A) \to B \mid (x : A) \times B \mid \mathtt{W}(x : A)B \mid \mathtt{Eq}_A(M; N) & \\
& & (k : \mathtt{clk}) \to A \mid \{k \div \mathtt{clk}\} \to A \mid \blacktriangleright_k A & \\
& & \mathtt{void} \mid \mathtt{unit} \mid \mathtt{bool} \mid \mathtt{nat} \mid \mathtt{U}_i & \\
\Delta & ::= & \cdot \mid \Delta, k & \text{(clock contexts)} \\
\Psi & ::= & \cdot \mid \Psi, x & \text{(variable contexts)} \\
\Gamma & ::= & \cdot \mid \Gamma, x : A & \text{(typing contexts)}
\end{array}
$$

Figure 2: The syntax of formal terms in Guarded Computational Type Theory (**CTT**$_{\circleddash}$). Formal terms M are identified up to renamings of their bound variables; by convention, bound variables are always assumed fresh.

pair types $(x : A) \times B$, wellordering types $\mathtt{W}(x : A)B$, extensional equality types $\mathtt{Eq}_A(M; N)$, clock-indexed later modalities $\blacktriangleright_k A$, clock product types $(k : \mathtt{clk}) \to A$, clock intersection types $\{k \div \mathtt{clk}\} \to A$, booleans, natural numbers, and a countable hierarchy of type universes $\mathtt{U}_i$. We define the following derived forms for non-dependent function and pair types:

$$
A \to B \triangleq (x : A) \to B \qquad\qquad A \times B \triangleq (x : A) \times B
$$

Forming fixed points and primitive recursors General fixed points can be programmed in **CTT**$_{\circleddash}$ exactly as in the untyped λ-calculus, but in order to simplify our metatheorems we have provided a primitive fixed point operator $\mathtt{fix}\, x\, \mathtt{in}\, M$. This can, for instance, be used to realize the induction principle for the natural numbers.

When a function has type $\blacktriangleright_k A \to A$, its *guarded* fixed point always exists and has type A. Because **CTT**$_{\circleddash}$ is dependently typed, it is very easy for us to write a program that computes the type of guarded streams of bits relative to a clock k, using the fixed point operator in concert with the later modality; and using the clock intersection type, we can transform this into the type of infinite sequences of bits:

$$
\begin{aligned}
&\mathtt{stream} \in (k : \mathtt{clk}) \to \mathtt{U}_i \\
&\mathtt{stream} \triangleq \lambda k.\, \mathtt{fix}\, A\, \mathtt{in}\, \mathtt{bool} \times \blacktriangleright_k A
\end{aligned}
$$

$$
\begin{aligned}
&\mathtt{sequence} \in \mathtt{U}_i \\
&\mathtt{sequence} \triangleq \{k \div \mathtt{clk}\} \to \mathtt{stream}\, k
\end{aligned}
$$

We will see in Section 3.8 that these expressions are indeed types in **CTT**$_{\circleddash}$.

3 Mathematical Meaning Explanation

In the type-theoretic tradition of Martin-Löf, formal language is endowed with computational meaning through what is called a “meaning explanation”; this style of definition, which was first deployed by Martin-Löf in his seminal paper *Constructive Mathematics and Computer Programming* [Martin-Löf, 1979], is closely related to PER semantics and the method of computability. This computational perspective was developed to its fullest extent in Nuprl's **CTT** [Constable et al., 1986, Allen et al., 2006], which adds a theory of computational congruence to the picture, together with many new connectives including intersections, unions, subset comprehensions, quotients and image types.

A meaning explanation provides a semantics for types as specifications of the execution behavior of untyped programs. As such, the judgments of type theory express the compliance of a program with a specification, which can be of arbitrary quantifier complexity, and will not generally be decidable. Any implementation of type theory involves, in one form or another, a formal system for deriving correct judgments that is, by definition, recursively enumerable and often decidable.

To achieve various properties that are desirable of a formal system (sometimes including decidability), programs are often decorated with type information that is not needed during execution. The meaning explanation is, then, lifted to the formalism along an erasure map $\|-\|$ that removes these decorations.

A similar, but more elaborate transformation of syntax (from formal terms to programs) is used here to facilitate the meaning explanation for guarded type theory in terms of the settings of a collection of clocks. During the verification of a program specification, the value of a clock may change (for instance, underneath the later modality); the most direct way to express this is to explicitly formulate the meaning explanation using a Kripke or presheaf-style semantics: a "possible world" consists of a collection of clocks and their settings, and we require specifications to account for the expansion of the world with new clocks and the alterations of their settings.

Doing so tends to clutter the meaning explanation by distributing the conditioning on clocks throughout the semantics, and disrupts a basic principle of type theory in the Martin-Löf tradition, which is that types should do little more than internalize the structures which are already present in the judgmental base.

An alternative, which we adopt here, is to formulate the semantics in a presheaf topos $\mathbf{S}_{\circledcirc}$ which accounts all at once for clocks and the passage of time, so that the specifications given by types are implicitly conditioned on them. This conditioning, which is implicit when viewed from inside the topos, can be externalized and made explicit using the Kripke-Joyal forcing semantics of $\mathbf{S}_{\circledcirc}$ [Mac Lane and Moerdijk, 1992].

To ensure that programs evolve appropriately along the transitions between clock worlds simultaneously with their specifications, we introduce a kind of "higher-order abstract syntax" which links clocks in programs directly to their meaning in the presheaf topos, as elements of the presheaf of clocks $\mathbb{K} : \mathbf{S}_{\circledcirc}$. The passage to this new kind of syntax at the interface between the formalism and the semantics is managed by an elaboration function $\|-\|$.

3.1 The semantic universe $\mathbf{S}_{\circledcirc}$

We will develop our semantic universe as a presheaf topos called $\mathbf{S}_{\circledcirc}$ over a category of clock contexts and clock context morphisms. We will require the following things to exist in $\mathbf{S}_{\circledcirc}$:

1. An object $\mathbb{K} : \mathbf{S}_{\circlearrowleft}$ of *clock names*.
2. A family of logical modalities $\triangleright_\kappa \varphi$ for clock names $\kappa : \mathbb{K}$ and predicates φ in $\mathbf{S}_{\circlearrowleft}$.

When we define $\mathbf{S}_{\circlearrowleft}$, we will arrange for the following principles to hold in its internal logic:

$$\exists \kappa : \mathbb{K}.\ \top \qquad \text{(Theorem 25)}$$
$$\forall \varphi : \Omega^{\mathbb{K}}.\ (\forall \kappa : \mathbb{K}.\ \triangleright_\kappa \varphi(\kappa)) \Rightarrow \forall \kappa : \mathbb{K}.\ \varphi(\kappa) \qquad \text{(Theorem 27)}$$
$$\forall \kappa : \mathbb{K}.\ \forall \varphi : \Omega.\ \varphi \Rightarrow \triangleright_\kappa \varphi \qquad \text{(Theorem 28)}$$
$$\forall \kappa : \mathbb{K}.\ \forall \varphi, \psi : \Omega.\ \triangleright_\kappa(\varphi \wedge \psi) \equiv (\triangleright_\kappa \varphi \wedge \triangleright_\kappa \psi) \qquad \text{(Theorem 29)}$$
$$\forall \kappa : \mathbb{K}.\ \forall \varphi, \psi : \Omega.\ \triangleright_\kappa(\varphi \Rightarrow \psi) \equiv (\triangleright_\kappa \varphi \Rightarrow \triangleright_\kappa \psi) \qquad \text{(Theorem 31)}$$
$$\forall \kappa : \mathbb{K}.\ \forall \varphi : \Omega.\ (\triangleright_\kappa \varphi \Rightarrow \varphi) \Rightarrow \varphi \qquad \text{(Theorem 32)}$$

We require one additional axiom to hold for any object $Y : \mathbf{S}_{\circlearrowleft}$ which is *total* and inhabited in a sense that we will define (Definitions 33, 34), analogous to the notion from Birkedal et al. [2011]:

$$\forall \kappa : \mathbb{K}.\ \forall \varphi : \Omega^{Y}.\ \triangleright_\kappa(\exists y : Y.\ \varphi(y)) \Rightarrow \exists y : Y.\ \triangleright_\kappa \varphi(y) \qquad \text{(Theorem 35)}$$

To construct $\mathbf{S}_{\circlearrowleft}$ as a topos of presheaves, first define $\mathbb{F}_+ : \mathbf{Cat}$ as the free category with strictly associative binary products generated by a single object; explicitly, objects of $\mathbb{F}_+$ are $U \equiv \bullet^n$ for $n > 0$. A map $f : \bullet^n \to \bullet^m$ is a vector of projections, but can dually be regarded as a function between finite sets $\mathbb{N}_{<m} \to \mathbb{N}_{<n}$.

Observe that the opposite category $\mathbb{F}_+^{\mathsf{op}}$ is a skeleton of the category of non-empty finite sets and all functions between them. $\mathbb{F}_+$ is also a full subcategory of $\mathbb{F} : \mathbf{Cat}$, the free strict cartesian category generated by a single object (whose opposite is likewise a skeleton of the category of finite sets and all maps between them).

Remark 1. *The category of presheaves $\widehat{\mathbb{F}_+}$ is equivalent to the sheaf subcategory of $\widehat{\mathbb{F}}$ under the coverage generated by singleton families of epimorphisms [Staton, 2007]. This sheaf subcategory is completely analogous to the Schanuel topos (i.e. the category of nominal sets), except that names are subject to identification/contraction. When names are used to represent clocks, this phenomenon has been referred to as "synchronization" by Bizjak and Møgelberg [2015].*

Define the presheaf of clock names $\mathbb{I} : \widehat{\mathbb{F}_+}$ as the representable functor $\mathbf{y}(\bullet^1)$. Next, define a functor $\circlearrowleft[-] : \mathbb{F}_+ \to \mathbf{Pos}$ (with $\mathbf{Pos}$ the category of partially ordered sets) which will interpret assignments of *times* to clock names:

$$\circlearrowleft[-] : \mathbb{F}_+ \to \mathbf{Pos}$$
$$\circlearrowleft[U : \mathbb{F}_+] \triangleq \omega^{\mathbb{I}(U)}$$
$$\circlearrowleft[f : V \to U](\partial_V : \omega^{\mathbb{I}(V)}) \triangleq (\kappa : \mathbb{I}(U)) \mapsto \partial_V(f^*\kappa)$$

Thinking of elements of $\mathbb{F}_+$ as signifying finite and non-empty cardinalities of clock names, the action of $\circlearrowleft[-]$ on objects takes such a cardinality $U : \mathbb{F}_+$ to the U-fold product of the poset ω, ordered pointwise: in other words, it assigns the amount of "time left" to each clock.

Finally, using the covariant Grothendieck construction [Crole, 1993] we can build the total category $\circlearrowleft : \mathbf{Cat} \triangleq \int^{\mathbb{F}_+} \circlearrowleft[-]$ in the following way. Objects are pairs $(U : \mathbb{F}_+, \partial_U : \circlearrowleft[U])$, i.e. collections of clock names together with an assignment; morphisms $f : (V, \partial_V) \to (U, \partial_U)$ are $\mathbb{F}_+$-morphisms $f : V \to U$ such that $\circlearrowleft[f](\partial_V) \leq \partial_U$ in $\circlearrowleft[U]$. At this time it will be helpful to impose some notation: we will write $\ell : \circlearrowleft \to \mathbb{F}_+$ for the induced projection functor, and we will use boldface letters $\mathbf{U}, \mathbf{V}$ to range over objects $(U, \partial_U), (V, \partial_V) : \circlearrowleft$.

The semantic universe $\mathbf{S}_{\circlearrowleft}$ Finally, we define our semantic universe as the presheaf topos $\mathbf{S}_{\circlearrowleft} \triangleq \widehat{\circlearrowleft}$. This "topos of clocks" defined above inherits a rich internal logic which corresponds to a combination of cartesian/structural nominal logic[4] and guarded recursion.

The topos $\mathbf{S}_{\circlearrowleft}$ is related to the models considered by Bizjak and Møgelberg [2015], except that rather than constructing a family of presheaf toposes fibered over clock contexts, we combine clock contexts with time assignments into a single base category, and take the topos of presheaves over that; our topos is nearly identical to the presheaf category considered independently in Bizjak and Møgelberg [2017].

One minor difference between our model and those of Bizjak and Møgelberg is that in order to close the internal logic of $\mathbf{S}_{\circlearrowleft}$ under the clock irrelevance axiom described above, we decided to rule out empty clock contexts; this condition is equivalent to taking a sheaf subtopos of the presheaves over *all* clock contexts.

The object of clock names We need to exhibit an object in the presheaf topos $\mathbf{S}_{\circlearrowleft}$ whose elements are the "available" clock *names* (without regard to their time assignments). First observe that the representable object $\mathbf{1}$ plays exactly this role in the category $\widehat{\mathbb{F}_+}$: at clock context $\bullet^n$ it consists in the set of morphisms $\bullet^n \to \bullet^1$, which has cardinality n. However, this object resides in the wrong topos, since we need to define an object $\mathbb{K} : \mathbf{S}_{\circlearrowleft}$. To achieve this, we use the reindexing functor $\ell^* : \widehat{\mathbb{F}_+} \to \mathbf{S}_{\circlearrowleft}$ induced by precomposing the projection $\ell : \mathbf{S}_{\circlearrowleft} \to \mathbb{F}_+$, defining $\mathbb{K} \triangleq \ell^* \mathbf{1}$.

Notations and morphisms We write $\mathbf{U}[\kappa \mapsto n]$ to mean $(U, \partial_U[\kappa \mapsto n])$, where $\partial_U[\kappa \mapsto n]$ means the adjustment to ∂_U which replaces $\partial_U(\kappa)$ with n. Finally, for the map that increments the time assigned to a clock, we write $[\kappa \mathrel{+}= 1] : \mathbf{U} \to \mathbf{U}[\kappa \mapsto \partial_U(\kappa) + 1]$.

Defining the $\triangleright_\kappa$ ***modalities*** We define the $\triangleright_\kappa$ modalities by their forcing clause in the Kripke-Joyal semantics of $\mathbf{S}_{\circlearrowleft}$:[5]

$$\mathbf{U} \Vdash \triangleright_\kappa \varphi(\alpha) \triangleq \begin{cases} \top & \textbf{if } \partial_U(\kappa) \equiv 0 \\ \mathbf{U}[\kappa \mapsto n] \Vdash \varphi([\kappa \mathrel{+}= 1]^* \alpha) & \textbf{if } \partial_U(\kappa) \equiv n + 1 \end{cases}$$

[4] That is, the logic of *nominal substitution sets* [Staton, 2007, Gabbay and Hofmann, 2008].

[5] Usually the forcing clauses should be taken as theorems rather than as definitions. However, in a Grothendieck topos, it is possible to define a subobject by its forcing clause: the result is well-defined when the definition is monotone (and also local, in the case of sheaf toposes).

By a similar definition, it is possible to define an analogous operator in the internal type theory of $\mathbf{S}_{\oplus}$, i.e. a fibered endofunctor $\blacktriangleright : \mathbf{S}_{\oplus}/X \times \mathbb{K} \to \mathbf{S}_{\oplus}/X \times \mathbb{K}$; however, we have only needed the logical modality in our construction.

All the other forcing clauses are completely standard; for a reference on Kripke-Joyal forcing, see Mac Lane and Moerdijk [1992].

3.2 Programming language and operational semantics

In Section 2 (Figure 2) we gave a grammar for the formal terms of $\mathbf{CTT}_{\oplus}$; however, in our semantics, we employ a second notion of syntax which is constructed as an inductive definition internal to $\mathbf{S}_{\oplus}$; this is the language of programs, and differs from the syntax of formal terms in two respects:

1. Clocks in programs are imported directly from the metatheoretic object of clocks $\mathbb{K} : \mathbf{S}_{\oplus}$; so the family of operators $\blacktriangleright_{\kappa} -$ is indexed in $\kappa : \mathbb{K}$ in exactly the same way that $\mathtt{U}_i$ is indexed in $i : \mathbb{N}$.
2. The binding of clocks (such as in the clock intersection operator) is represented using the exponential $-^{\mathbb{K}} : \mathbf{S}_{\oplus} \to \mathbf{S}_{\oplus}$.[6]

Remark 2 (Generalized Syntax). *The idea of using the exponential of the metalanguage in the syntax of a programming language is not new. Infinitary notions of program syntax can be traced back as far as Brouwer's F-inference in the justification of the Bar Principle [Brouwer, 1981], and have more recently been developed in Nuprl semantics [Rahli et al., 2017], as well as in the context of higher-order focusing Zeilberger [2009].*

We will define the inductive family $\mathbf{Prog}_n$ of *programs with n free variables* in $\mathbf{S}_{\oplus}$ using an internal inductive definition, summarized in Figure 3.

Substitution structure Writing $\mathbf{SET}$ to mean the internal category of small sets in $\mathbf{S}_{\oplus}$, observe that $\mathbf{Var}_{\_}$ can be regarded as an internal functor from $\mathbf{FIN}$ to $\mathbf{SET}$, where $\mathbf{FIN}$ is the internal category of finite cardinals and all functions between them. We can equip $\mathbf{Prog}_{\_}$ with the structure of a relative monad on $\mathbf{Var}_{\_} : \mathbf{FIN} \to \mathbf{SET}$ [Altenkirch et al., 2010].

The unit of the relative monad is the injection of variables $\mathbf{p}_{(-)}$; its Kleisli extension implements substitutions $M{\cdot}\gamma : \mathbf{Prog}_n$ for $M : \mathbf{Prog}_m$ and $\gamma : \mathbf{Prog}_n^{\mathbf{Var}_m}$. We omit the definition of the Kleisli extension because it is completely standard.

Internal operational semantics Programs are endowed with operational meaning through the definition of a transition system, summarized in Figure 4. This defines predicates $-\ \mathit{val} : \mathcal{P}(\mathbf{Prog}_0)$ and $- \mapsto - : \mathcal{P}(\mathbf{Prog}_0 \times \mathbf{Prog}_0)$ in $\mathbf{S}_{\oplus}$. Write $\mathbf{Val} : \mathbf{S}_{\oplus}$ for the subobject $\{M : \mathbf{Prog}_0 \mid M\ \mathit{val}\}$.

Write $- \mapsto^{\star} -$ for the reflexive-transitive closure of $- \mapsto -$. We now define approximation and computational equivalence judgments $- \preccurlyeq -$, $- \approx - : \mathcal{P}(\mathbf{Prog}_0 \times \mathbf{Prog}_0)$

[6] While this construction cannot be called "ordinary syntax", it is an inductive definition that can be built up explicitly using the fact that $\mathbf{S}_{\oplus}$ models indexed W-types [Moerdijk and Palmgren, 2000].

$$\mathbf{Var}_n \triangleq \{i \mid i < n\}$$

$$\frac{i : \mathbf{Var}_n}{\mathtt{Var}_i : \mathbf{Prog}_n} \qquad \frac{M : \mathbf{Prog}_{n+1}}{\boldsymbol{\lambda}(M) : \mathbf{Prog}_n} \qquad \frac{M_0 : \mathbf{Prog}_n \qquad M_1 : \mathbf{Prog}_n}{M_0(M_1) : \mathbf{Prog}_n} \qquad \frac{M : \mathbf{Prog}_{n+1}}{\mathtt{fix}(M) : \mathbf{Prog}_n}$$

$$\frac{M_0 : \mathbf{Prog}_n \qquad M_1 : \mathbf{Prog}_n}{\langle M_0, M_1 \rangle : \mathbf{Prog}_n} \qquad \frac{M : \mathbf{Prog}_n}{M.1 : \mathbf{Prog}_n} \qquad \frac{M : \mathbf{Prog}_n}{M.2 : \mathbf{Prog}_n} \qquad \frac{}{\bigstar : \mathbf{Prog}_n}$$

$$\frac{}{\mathtt{tt} : \mathbf{Prog}_n} \qquad \frac{}{\mathtt{ff} : \mathbf{Prog}_n} \qquad \frac{}{\mathtt{ze} : \mathbf{Prog}_n} \qquad \frac{M : \mathbf{Prog}_n}{\mathtt{su}(M) : \mathbf{Prog}_n}$$

$$\frac{M_b : \mathbf{Prog}_n \qquad M_t : \mathbf{Prog}_n \qquad M_f : \mathbf{Prog}_n}{\mathtt{if}(M_b; M_t; M_f) : \mathbf{Prog}_n} \qquad \frac{M_n : \mathbf{Prog}_n \qquad M_z : \mathbf{Prog}_n \qquad M_s : \mathbf{Prog}_{n+1}}{\mathtt{ifze}(M_n; M_z; M_s) : \mathbf{Prog}_n}$$

$$\frac{M : \mathbf{Prog}_n \qquad N : \mathbf{Prog}_{n+1}}{\mathtt{sup}(M; N) : \mathbf{Prog}_n} \qquad \frac{M : \mathbf{Prog}_n \qquad N : \mathbf{Prog}_{n+3}}{\mathtt{rec}_{\mathtt{W}}(M; N) : \mathbf{Prog}_n} \qquad \frac{A : \mathbf{Prog}_n \qquad B : \mathbf{Prog}_{n+1}}{\boldsymbol{\Pi}(A; B) : \mathbf{Prog}_n}$$

$$\frac{A : \mathbf{Prog}_n \qquad B : \mathbf{Prog}_{n+1}}{\boldsymbol{\Sigma}(A; B) : \mathbf{Prog}_n} \qquad \frac{A : \mathbf{Prog}_n \qquad B : \mathbf{Prog}_{n+1}}{\mathtt{W}(A; B) : \mathbf{Prog}_n}$$

$$\frac{A : \mathbf{Prog}_n \qquad M_0 : \mathbf{Prog}_n \qquad M_1 : \mathbf{Prog}_n}{\mathtt{Eq}_A(M_0; M_1) : \mathbf{Prog}_n} \qquad \frac{\kappa : \mathbb{K} \qquad A : \mathbf{Prog}_n}{\blacktriangleright_\kappa A : \mathbf{Prog}_n} \qquad \frac{A : \mathbf{Prog}_n^{\mathbb{K}}}{\circlearrowleft A : \mathbf{Prog}_n}$$

$$\frac{}{\mathtt{void} : \mathbf{Prog}_n} \qquad \frac{}{\mathtt{unit} : \mathbf{Prog}_n} \qquad \frac{}{\mathtt{bool} : \mathbf{Prog}_n} \qquad \frac{}{\mathtt{nat} : \mathbf{Prog}_n} \qquad \frac{i : \mathbb{N}}{\mathtt{U}_i : \mathbf{Prog}_n}$$

Figure 3: The inductive definition of the programs with n free variables $\mathbf{Prog}_n : \mathbf{S}_{\odot}$.

$$\frac{}{\mathtt{\lambda}(M)\ \mathtt{val}} \qquad \frac{}{\langle M_0, M_1\rangle\ \mathtt{val}} \qquad \frac{}{\bigstar\ \mathtt{val}} \qquad \frac{}{\mathtt{tt}\ \mathtt{val}} \qquad \frac{}{\mathtt{ff}\ \mathtt{val}} \qquad \frac{}{\mathtt{ze}\ \mathtt{val}}$$

$$\frac{}{\mathtt{su}(M)\ \mathtt{val}} \qquad \frac{}{\mathtt{\Pi}(A; B)\ \mathtt{val}} \qquad \frac{}{\mathtt{\Sigma}(A; B)\ \mathtt{val}} \qquad \frac{}{\mathtt{Eq}_A(M_0; M_1)\ \mathtt{val}} \qquad \frac{}{\blacktriangleright_\kappa A\ \mathtt{val}}$$

$$\frac{}{\circlearrowleft A\ \mathtt{val}} \qquad \frac{}{\mathtt{void}\ \mathtt{val}} \qquad \frac{}{\mathtt{unit}\ \mathtt{val}} \qquad \frac{}{\mathtt{bool}\ \mathtt{val}} \qquad \frac{}{\mathtt{nat}\ \mathtt{val}} \qquad \frac{}{\mathtt{U}_i\ \mathtt{val}}$$

$$\frac{M_0 \mapsto M_0'}{M_0\ M_1 \mapsto M_0'\ M_1} \qquad \frac{M_0 \mapsto M_0'}{M_0\ k \mapsto M_0'\ k} \qquad \frac{M \mapsto M'}{M.1 \mapsto M'.1} \qquad \frac{M \mapsto M'}{M.2 \mapsto M'.2}$$

$$\frac{M_b \mapsto M_b'}{\mathtt{if}(M_b; M_t; M_f) \mapsto \mathtt{if}(M_b'; M_t; M_f)} \qquad \frac{M_n \mapsto M_n'}{\mathtt{ifze}(M_n; M_z; M_s) \mapsto \mathtt{ifze}(M_n'; M_z; M_s)}$$

$$\frac{M \mapsto M'}{\mathtt{rec}_{\mathtt{W}}(M; N) \mapsto \mathtt{rec}_{\mathtt{W}}(M'; N)} \qquad \frac{}{(\mathtt{\lambda}(M_f))\ M \mapsto M_f \cdot M} \qquad \frac{}{(\mathtt{\lambda}_{\odot}(M))\ \kappa \mapsto M(\kappa)}$$

$$\frac{}{\langle M_0, M_1\rangle.1 \mapsto M_0} \qquad \frac{}{\langle M_0, M_1\rangle.2 \mapsto M_1} \qquad \frac{}{\mathtt{if}(\mathtt{tt}; M_t; M_f) \mapsto M_t} \qquad \frac{}{\mathtt{if}(\mathtt{ff}; M_t; M_f) \mapsto M_f}$$

$$\frac{}{\mathtt{ifze}(\mathtt{ze}; M_z; M_s) \mapsto M_z} \qquad \frac{}{\mathtt{ifze}(\mathtt{su}(M_n); M_z; M_s) \mapsto M_s \cdot M_n}$$

$$\frac{}{\mathtt{rec}_{\mathtt{W}}(\mathtt{sup}(M; N); O) \mapsto O \cdot [M, N, \mathtt{rec}_{\mathtt{W}}(N \cdot M; O)]} \qquad \frac{}{\mathtt{fix}(M) \mapsto M \cdot \mathtt{fix}(M)}$$

Figure 4: Structural operational semantics of closed $\mathbf{CTT}_{\odot}$ programs.

respectively for closed programs as follows:

$$M_0 \preccurlyeq M_1 \triangleq \forall M_v : \mathbf{Val}.\ M_0 \mapsto^\star M_v \Rightarrow M_1 \mapsto^\star M_v$$
$$M_0 \approx M_1 \triangleq M_0 \preccurlyeq M_1 \wedge M_1 \preccurlyeq M_0$$

The latter is extended to a computational equivalence judgment for open programs $- \approx_n - : \mathcal{P}(\mathbf{Prog}_n \times \mathbf{Prog}_n)$ by quantifying over total substitutions.

$$M_0 \approx_n M_1 \triangleq \forall \gamma : \mathbf{Prog}_0^n.\ M_0 \cdot \gamma \approx M_1 \cdot \gamma$$

It would be desirable to extend this relation to a theory of computational congruence, as pioneered by Howe [1989]; however, for our immediate purposes it has sufficed to require types only to respect the approximation relation defined above.

Definition 3 (Computational PERs)**.** A partial equivalence relation is a binary relation which is both symmetric and transitive. Such a relation $\boldsymbol{R}$ on $\mathbf{Prog}_0$ is called *computational* when it respects approximation in the following sense: if $(M_0, M_1) \in \boldsymbol{R}$ and $M_0 \preccurlyeq M_0'$, then $(M_0', M_1) \in \boldsymbol{R}$.

Telescopes To capture the syntax of contexts and we define the inductive family $\mathbf{Tl}_n$ of *telescopes of length* n as follows:

$$\frac{}{\cdot : \mathbf{Tl}_0} \qquad \frac{\Gamma : \mathbf{Tl}_n \qquad A : \mathbf{Prog}_n}{\Gamma.A : \mathbf{Tl}_{n+1}}$$

Elaborating terms We now sketch the elaboration of the program terms of Section 2 into programs; approximately, a term M with free formal clock variables Δ and free term variables Ψ will be elaborated to a morphism $\|\Delta \mid \Psi \vdash M\| : \mathbb{K}^{|\Delta|} \to \mathbf{Prog}_{|\Psi|}$.

Notation 4. When Ψ is a list, we write $|\Psi|$ for its length, and we write $\Psi[x]$ for the index $i < |\Psi|$ of the element x in Ψ, presupposing $\Psi \ni x$.

$$\begin{aligned}
\|\Delta \mid \Psi \vdash x\|\varrho &= \mathbf{p}_{\Psi[x]} \\
\|\Delta \mid \Psi \vdash \lambda x.\, M\|\varrho &= \boldsymbol{\lambda}(\|\varrho \mid \Psi, x \vdash M\|\varrho) \\
\|\Delta \mid \Psi \vdash \lambda k.\, M\|\varrho &= \boldsymbol{\lambda}_{\odot}(\kappa \mapsto \|\Delta, k \mid \Psi \vdash M\|(\varrho, \kappa)) \\
\|\Delta \mid \Psi \vdash M_0\, M_1\|\varrho &= (\|\Delta \mid \Psi \vdash M_0\|\varrho)(\|\Delta \mid \Psi \vdash M_1\|\varrho) \\
\|\Delta \mid \Psi \vdash M\, k\|\varrho &= (\|\Delta \mid \Psi \vdash M\|\varrho)(\rho_{\Delta[k]}) \\
\|\Delta \mid \Psi \vdash \langle M_0, M_1\rangle\|\varrho &= \langle \|\Delta \mid \Psi \vdash M_0\|\varrho, \|\Delta \mid \Psi \vdash M_1\|\varrho\rangle \\
\|\Delta \mid \Psi \vdash M.1\|\varrho &= (\|\Delta \mid \Psi \vdash M\|\varrho).1 \\
\|\Delta \mid \Psi \vdash M.2\|\varrho &= (\|\Delta \mid \Psi \vdash M\|\varrho).2 \\
\|\Delta \mid \Psi \vdash \mathsf{sup}(M; x.N)\|\varrho &= \mathbf{sup}(\|\Delta \mid \Psi \vdash M\|\varrho; \|\Delta \mid \Psi, x \vdash N\|\varrho) \\
\|\Delta \mid \Psi \vdash \mathsf{rec}_{\mathsf{W}}(M; x, y, z.N)\|\varrho &= \mathbf{rec}_{\mathbf{W}}(\|\Delta \mid \Psi \vdash M\|\varrho; \|\Delta \mid \Psi, x, y, z \vdash N\|\varrho)
\end{aligned}$$

$$
\begin{aligned}
\|\Delta \mid \Psi \vdash \bigstar\|\varrho &= \bigstar \\
\|\Delta \mid \Psi \vdash \mathtt{tt}\|\varrho &= \mathbf{tt} \\
\|\Delta \mid \Psi \vdash \mathtt{ff}\|\varrho &= \mathbf{ff} \\
\|\Delta \mid \Psi \vdash \mathtt{if}(M_b; M_t; M_f)\|\varrho &= \mathbf{if}(\|\Delta \mid \Psi \vdash M_b\|\varrho; \|\Delta \mid \Psi \vdash M_t\|\varrho; \|\Delta \mid \Psi \vdash M_f\|\varrho) \\
\|\Delta \mid \Psi \vdash \mathtt{ze}\|\varrho &= \mathbf{ze} \\
\|\Delta \mid \Psi \vdash \mathtt{su}(M)\|\varrho &= \mathbf{su}(\|\Delta \mid \Psi \vdash M\|\varrho) \\
\|\Delta \mid \Psi \vdash \mathtt{ifze}(M_n; M_z; x.M_s)\|\varrho &= \mathbf{ifze}(\|\Delta \mid \Psi \vdash M_n\|\varrho; \|\Delta \mid \Psi \vdash M_z\|\varrho; \|\Delta \mid \Psi, x \vdash M_s\|\varrho) \\
\|\Delta \mid \Psi \vdash (x : A) \to B\|\varrho &= \mathbf{\Pi}(\|\Delta \mid \Psi \vdash A\|\varrho; \|\Delta \mid \Psi, x \vdash B\|\varrho) \\
\|\Delta \mid \Psi \vdash (x : A) \times B\|\varrho &= \mathbf{\Sigma}(\|\Delta \mid \Psi \vdash A\|\varrho; \|\Delta \mid \Psi, x \vdash B\|\varrho) \\
\|\Delta \mid \Psi \vdash \mathtt{W}(x : A)B\|\varrho &= \mathbf{W}(\|\Delta \mid \Psi \vdash A\|\varrho; \|\Delta \mid \Psi, x \vdash B\|\varrho) \\
\|\Delta \mid \Psi \vdash \mathtt{Eq}_A(M_0; M_1)\|\varrho &= \mathbf{Eq}_{\|\Delta \mid \Psi \vdash A\|\varrho}(\|\Delta \mid \Psi \vdash M_0\|\varrho; \|\Delta \mid \Psi \vdash M_1\|\varrho) \\
\|\Delta \mid \Psi \vdash \blacktriangleright_k A\|\varrho &= \blacktriangleright_{\varrho_{\Delta[k]}} \|\Delta \mid \Psi \vdash A\|\varrho \\
\|\Delta \mid \Psi \vdash (k : \mathtt{clk}) \to A\|\varrho &= \mathbf{\Pi}_{\odot}(\kappa \mapsto \|\Delta, k \mid \Psi \vdash A\|(\varrho, \kappa)) \\
\|\Delta \mid \Psi \vdash \{k \div \mathtt{clk}\} \to A\|\varrho &= \circlearrowleft(\kappa \mapsto \|\Delta, k \mid \Psi \vdash A\|(\varrho, \kappa)) \\
\|\Delta \mid \Psi \vdash \mathtt{void}\|\varrho &= \mathbf{void} \\
\|\Delta \mid \Psi \vdash \mathtt{unit}\|\varrho &= \mathbf{unit} \\
\|\Delta \mid \Psi \vdash \mathtt{bool}\|\varrho &= \mathbf{bool} \\
\|\Delta \mid \Psi \vdash \mathtt{nat}\|\varrho &= \mathbf{nat} \\
\|\Delta \mid \Psi \vdash \mathtt{U}_i\|\varrho &= \mathbf{U}_i
\end{aligned}
$$

Elaborating contexts Next, we elaborate contexts Γ with free formal clock variables Δ as morphisms $\|\Delta \mid \Gamma\| : \mathbb{K}^{|\Delta|} \to \mathbf{Tl}_{|\Gamma|}$, writing $\pi(\Gamma)$ for the sequence $\vec{x_i}$ when $\Gamma \equiv \overrightarrow{x_i : A_i}$.

$$
\begin{aligned}
\|\Delta \mid \cdot\|\varrho &= \cdot \\
\|\Delta \mid \Gamma, x : A\|\varrho &= (\|\Delta \mid \Gamma\|\varrho).(\|\Delta \mid \pi(\Gamma) \vdash A\|\varrho)
\end{aligned}
$$

To save space, we may write $\|M\|$ or $\|\Gamma\|$ for the elaboration of a term or a context respectively, when the parameters are obvious.

3.3 Full type system hierarchy

At a high level, a *type system* in the sense of Allen [1987] is an object which distinguishes some programs as types, and specifies what programs will be the elements of those types, and when they will be considered equal. Writing $\mathbf{rel}(X)$ for $\mathcal{P}(X \times X)$, we define a *candidate type system* to be a relation $\tau : \mathcal{P}(\mathbf{Prog}_0 \times \mathbf{rel}(\mathbf{Prog}_0))$ in $\mathbf{S}_{\odot}$. We will write $\mathbf{TS}_{cand}$ for the collection of such candidate type systems, i.e. $\mathbf{TS}_{cand} : \mathbf{S}_{\odot} \triangleq \mathcal{P}(\mathbf{Prog}_0 \times \mathbf{rel}(\mathbf{Prog}_0))$.

Let us now define notation for some assertions about candidate type systems τ : $\mathbf{TS}_{cand}$:

$$
\begin{aligned}
\tau \models A \doteq B &\triangleq \exists \mathcal{A} : \mathbf{rel}(\mathbf{Prog}_0).\ (A, \mathcal{A}) \in \tau \wedge (B, \mathcal{A}) \in \tau \\
\tau \models M_0 \doteq M_1 \in A &\triangleq \exists \mathcal{A} : \mathbf{rel}(\mathbf{Prog}_0).\ (A, \mathcal{A}) \in \tau \wedge (M_1, M_2) \in \mathcal{A}
\end{aligned}
$$

A candidate type system $\tau : \mathbf{TS}_{cand}$ can have the following characteristics:

1. It is called *extensional* if it is the graph of a partial function $\mathbf{Prog}_0 \rightharpoonup \mathbf{rel}(\mathbf{Prog}_0)$.
2. It is called *computational PER-valued* if whenever $(A, \mathbb{A}) \in \tau$, the relation $\mathbb{A}$ is a computational PER (see Definition 3).
3. It is called *type-computational* when, if $(A, \mathbb{A}) \in \tau$ and $A \preccurlyeq A'$, then also $(A', \mathbb{A}) \in \tau$.

Finally a candidate type system is called a *type system* if it is extensional, computational PER-valued, and type-computational. We write $\mathbf{TS} : \mathbf{S}_{\odot}$ for the collection of such type systems.

Sequents and functionality Next, we briefly sketch the meaning of type functionality sequents $\Gamma \gg A_0 \doteq A_1$ and functionality sequents $\Gamma \gg M_0 \doteq M_1 \in A$ using a simple notion of functionality derived from Martin-Löf [1979], with respect to any candidate type system $\tau : \mathbf{TS}_{cand}$.

When $\Gamma : \mathbf{Tl}_n$ is a telescope and $\gamma_0, \gamma_1 : \mathbf{Prog}_0^n$ are sequences of programs, we define similarity of instantiations $\gamma_0 \doteq \gamma_1 \in^\star \Gamma$ by recursion on Γ. $\cdot \doteq \cdot \in^\star \cdot$ is true, and $\gamma_0.M_0 \doteq \gamma_1.M_1 \in^\star \Gamma.A$ is true when both $\gamma_0 \doteq \gamma_1 \in^\star \Gamma$ and $M_0 \cdot \gamma_0 \doteq M_1 \cdot \gamma_1 \in A \cdot \gamma_0$ are true.

Open type similarity $\Gamma \gg A_0 \doteq A_1$ is true when for all instantiations $\gamma_0 \doteq \gamma_1 \in^\star \Gamma$, we have $A_0 \cdot \gamma_0 \doteq A_1 \cdot \gamma_1$. Likewise, open member smilarity $\Gamma \gg M_0 \doteq M_1 \in A$ is true when for all such instantiations, we have $M_0 \cdot \gamma_0 \doteq M_1 \cdot \gamma_1 \in A \cdot \gamma_0$.

Finally, context validity Γ ctx is given by recursion on Γ using open type similarity in the inductive case.

3.4 Closure under type formers other than universes

Next, we will show how to *close* a candidate type system under the type formers of $\mathbf{CTT}_{\odot}$, namely booleans, natural numbers, dependent functions types, dependent pair types, equality types, later modalities, clock intersection types and universes.

The simplest way to carry out this construction, as pioneered by Crary [1998] and formalized by Anand and Rahli [2014], is to use an inductive definition of a closure operator $\mathbf{C}[-] : \mathbf{TS}_{cand} \to \mathbf{TS}_{cand}$ on candidate type systems. However, this method does not immediately extend to the type systems that we consider in this paper, because it is not clear how to fit the clause for the *later modality* into the usual schemata for inductive definitions based on strictly positive signatures.

Therefore, as advocated by Allen [1987], we will build up our closure operator manually by taking the least fixed point of a monotone operator on candidate type systems; this construction can be carried out in any topos, because the Knaster-Tarski theorem guarantees a least fixed point for any monotone operator on a complete lattice [Davey and Priestley, 1990].

First, we define some notation for closing relations and type systems under evaluation to canonical form:

$$-^{\Downarrow} : \mathbf{rel}(\mathbf{Prog}_0) \to \mathbf{rel}(\mathbf{Prog}_0)$$
$$\mathbb{A}^{\Downarrow} \triangleq \left\{ (M_0, M_1) \mid \exists M_0^v, M_1^v : \mathbf{Val}.\ M_i \mapsto^\star M_i^v \wedge (M_0^v, M_1^v) \in \mathbb{A} \right\}$$

$$-^{\Downarrow} : \mathbf{TS}_{cand} \to \mathbf{TS}_{cand}$$

$$\tau^{\Downarrow} \triangleq \{(A, \mathbb{A}) \mid \exists A_v : \mathtt{Val}.\ A \mapsto^{\star} A_v \wedge (A_v, \mathbb{A}) \in \tau\}$$

In Figure 5, for an initial candidate type system $\sigma : \mathbf{TS}_{cand}$, we define an endomorphism on candidate type systems $\mathfrak{F}_\sigma : \mathbf{TS}_{cand} \to \mathbf{TS}_{cand}$ which extends a type system with all the non-universe connectives of $\mathbf{CTT}_{\odot}$.

A few remarks on our style of definition are in order. First, observe that we have *not* required that A be a type in order for $\blacktriangleright_\kappa A$ to be a type: we only require that this premise obtain *later*. This is crucial for the interaction of the later modality with the dependent product and function types.

Moreover, we have chosen a negative definition of dependent pair and function types, based on projections and application rather than on pairing and abstraction. This choice appears to likewise be forced for the same reason.

Finally, in the type-functionality clauses for dependent pair and function types, we require the family of relations $\mathbb{B}$ to be not only functional in $\mathbb{A}$ in the obvious sense, but also in a "criss-crossed" sense: for $(M_0, M_1) \in \mathbb{A}$ we additionally require $(B \cdot M_0, \mathbb{B}(M_1)) \in \tau$ and $(B \cdot M_1, \mathbb{B}(M_0)) \in \tau$. Ultimately this is redundant in case $\mathbb{A}$ is symmetric and τ is extensional; however, we found that building these extra instances into the definition made it simpler to prove that the closure of a type system is both extensional and CPER-valued under suitable conditions.

Theorem 5 (`Closure.Clo.monotonicity`). *For any candidate type system $\sigma : \mathbf{TS}_{cand}$, the function $\mathfrak{F}_\sigma : \mathbf{TS}_{cand} \to \mathbf{TS}_{cand}$ is monotone.*

Proof. By case on the type closure clauses above, which are themselves each monotone. □

Corollary 6 (`Closure.Clo.t`, `Closure.Clo.roll`). *By the Knaster-Tarski theorem, the function $\mathfrak{F}_\sigma$ has a least fixed point $\mu(\mathfrak{F}_\sigma)$.*

We will write $\mathbf{c}[-] : \mathbf{TS}_{cand} \to \mathbf{TS}_{cand}$ for the operator that takes $\sigma : \mathbf{TS}_{cand}$ to the fixed point $\mu(\mathfrak{F}_\sigma)$.

Lemma 7 (`Closure.Clo.extensionality`). *For any $\sigma : \mathbf{TS}_{cand}$ an extensional candidate type system which contains only types that evaluate to universes, the closure $\mathbf{c}[\sigma]$ is extensional.*

Proof. By the universal property of the closure operator. □

Lemma 8 (`Closure.Clo.cext_per`, `Closure.Clo.cext_computational`). *If the relation $\mathbb{A} : \mathtt{rel}(\mathtt{Prog}_0)$ is a PER, then $\mathbb{A}^{\Downarrow}$ is a computational PER.*

Proof. By the determinacy of evaluation. □

Lemma 9 (`Closure.Clo.cper_valued`). *If $\sigma : \mathbf{TS}_{cand}$ is CPER-valued, extensional and contains only types that evaluate to universes, then its closure $\mathbf{c}[\sigma]$ is CPER-valued.*

Proof. By the universal property of the closure operator, using Theorem 29. □

$$\mathfrak{F}_\sigma : \mathbf{TS}_{cand} \to \mathbf{TS}_{cand}$$

$$\mathfrak{F}_\sigma(\tau) \triangleq \sigma \cup \mathfrak{Conn}(\tau)^{\Downarrow}$$

where

$$\mathfrak{Conn}(\tau) \triangleq \mathfrak{Void}(\tau) \cup \mathfrak{Unit}(\tau) \cup \mathfrak{Bool}(\tau) \cup \mathfrak{Nat}(\tau) \cup \mathfrak{Prod}(\tau) \cup \mathfrak{Fun}(\tau) \cup \mathfrak{Fun}_{\circleddash}(\tau) \cup \mathfrak{Eq}(\tau) \cup \mathfrak{Ltr}(\tau) \cup \mathfrak{Isect}(\tau) \cup \mathfrak{Tree}(\tau)$$

$$\mathfrak{Void}(\tau) \triangleq \{(\mathtt{void}, \boldsymbol{X}) \mid \boldsymbol{X} \equiv \varnothing\}$$

$$\mathfrak{Unit}(\tau) \triangleq \{(\mathtt{unit}, \boldsymbol{X}^{\Downarrow}) \mid \boldsymbol{X} \equiv \{(\star, \star)\}\}$$

$$\mathfrak{Bool}(\tau) \triangleq \{(\mathtt{bool}, \boldsymbol{X}^{\Downarrow}) \mid \boldsymbol{X} \equiv \{(\mathtt{tt}, \mathtt{tt}), (\mathtt{ff}, \mathtt{ff})\}\}$$

$$\mathfrak{Nat}(\tau) \triangleq \{(\mathtt{nat}, \boldsymbol{X}^{\Downarrow}) \mid \boldsymbol{X} \equiv \mu\boldsymbol{Y}.\ \{(\mathtt{ze}, \mathtt{ze})\} \cup \{(\mathtt{su}(M_0), \mathtt{su}(M_1)) \mid (M_0, M_1) \in \boldsymbol{Y}^{\Downarrow}\}\}$$

$$\mathfrak{Prod}(\tau) \triangleq \left\{ \begin{array}{l} (\Sigma(A;B), \boldsymbol{X}) \mid \\ \quad \exists \boldsymbol{A}{:}\mathtt{rel}(\mathtt{Prog}_0), \boldsymbol{B}{:}\mathtt{rel}(\mathtt{Prog}_0)^{\mathtt{Prog}_0}. \\ \qquad (A, \boldsymbol{A}) \in \tau \\ \qquad \wedge\ \forall (M_0, M_1) \in \boldsymbol{A}.\ (B \cdot M_0, \boldsymbol{B}(M_0)), (B \cdot M_1, \boldsymbol{B}(M_0)), (B \cdot M_1, \boldsymbol{B}(M_1)), (B \cdot M_0, \boldsymbol{B}(M_1)) \in \tau \\ \qquad \wedge\ \boldsymbol{X} \equiv \{(M_0, M_1) \mid (M_0.1, M_1.1) \in \boldsymbol{A} \wedge (M_0.2, M_1.2) \in \boldsymbol{B}(M_0.1)\} \end{array} \right\}$$

$$\mathfrak{Fun}(\tau) \triangleq \left\{ \begin{array}{l} (\Pi(A;B), \boldsymbol{X}) \mid \\ \quad \exists \boldsymbol{A}{:}\mathtt{rel}(\mathtt{Prog}_0), \boldsymbol{B}{:}\mathtt{rel}(\mathtt{Prog}_0)^{\mathtt{Prog}_0}. \\ \qquad (A, \boldsymbol{A}) \in \tau \\ \qquad \wedge\ \forall (M_0, M_1) \in \boldsymbol{A}.\ (B \cdot M_0, \boldsymbol{B}(M_0)), (B \cdot M_1, \boldsymbol{B}(M_0)), (B \cdot M_1, \boldsymbol{B}(M_1)), (B \cdot M_0, \boldsymbol{B}(M_1)) \in \tau \\ \qquad \wedge\ \boldsymbol{X} \equiv \{(M_0, M_1) \mid \forall (N_0, N_1) \in \boldsymbol{A}.\ (M_0(N_0), M_1(N_1)) \in \boldsymbol{B}(N_0)\} \end{array} \right\}$$

$$\mathfrak{Tree}(\tau) \triangleq \left\{ \begin{array}{l} (\mathtt{W}(A;B), \boldsymbol{X}^{\Downarrow}) \mid \\ \quad \exists \boldsymbol{A}{:}\mathtt{rel}(\mathtt{Prog}_0), \boldsymbol{B}{:}\mathtt{rel}(\mathtt{Prog}_0)^{\mathtt{Prog}_0}. \\ \qquad (A, \boldsymbol{A}) \in \tau \\ \qquad \wedge\ \forall (M_0, M_1) \in \boldsymbol{A}.\ (B \cdot M_0, \boldsymbol{B}(M_0)), (B \cdot M_1, \boldsymbol{B}(M_0)), (B \cdot M_1, \boldsymbol{B}(M_1)), (B \cdot M_0, \boldsymbol{B}(M_1)) \in \tau \\ \qquad \wedge\ \boldsymbol{X} \equiv \mu\boldsymbol{Y}.\ \left\{ \begin{array}{l} (\mathtt{sup}(M_0; N_0), \mathtt{sup}(M_1; N_1)) \mid \\ \quad (M_0, M_1) \in \boldsymbol{A} \wedge \forall (O_0, O_1) \in \boldsymbol{B}(M_0).\ (N_0 \cdot O_0, N_1 \cdot O_1) \in Y^{\Downarrow} \end{array} \right\} \end{array} \right\}$$

$$\mathfrak{Eq}(\tau) \triangleq \{(\mathtt{Eq}_A(M_0; M_1), \boldsymbol{X}^{\Downarrow}) \mid \exists \boldsymbol{A}{:}\mathtt{rel}(\mathtt{Prog}_0).(A, \boldsymbol{A}) \in \tau \wedge (M_0, M_0), (M_1, M_1) \in \boldsymbol{A} \wedge \boldsymbol{X} \equiv \{(\star, \star) \mid (M_0, M_1) \in \boldsymbol{A}\}\}$$

$$\mathfrak{Ltr}(\tau) \triangleq \{(\blacktriangleright_\kappa A, \boldsymbol{X}) \mid \exists \boldsymbol{A}{:}\mathtt{rel}(\mathtt{Prog}_0).\ \triangleright_\kappa((A, \boldsymbol{A}) \in \tau) \wedge \boldsymbol{X} \equiv \{(M_0, M_1) \mid \triangleright_\kappa((M_0, M_1) \in \boldsymbol{A})\}\}$$

$$\mathfrak{Isect}(\tau) \triangleq \left\{(\bigcap A, \boldsymbol{X}) \mid \exists \boldsymbol{A}{:}\mathtt{rel}(\mathtt{Prog}_0)^{\mathbb{K}}.\ (\forall \kappa{:}\mathbb{K}.\ (A(\kappa), \boldsymbol{A}(\kappa)) \in \tau) \wedge \boldsymbol{X} \equiv \{(M_0, M_1) \mid \forall \kappa{:}\mathbb{K}.\ (M_0, M_1) \in \boldsymbol{A}(\kappa)\}\right\}$$

$$\mathfrak{Fun}_{\circleddash}(\tau) \triangleq \left\{(\Pi_{\circleddash} A, \boldsymbol{X}) \mid \exists \boldsymbol{A}{:}\mathtt{rel}(\mathtt{Prog}_0)^{\mathbb{K}}.\ (\forall \kappa{:}\mathbb{K}.\ (A(\kappa), \boldsymbol{A}(\kappa)) \in \tau) \wedge \boldsymbol{X} \equiv \{(M_0, M_1) \mid \forall \kappa{:}\mathbb{K}.\ (M_0(\kappa), M_1(\kappa)) \in \boldsymbol{A}(\kappa)\}\right\}$$

Figure 5: A monotone operator on candidate type systems.

Lemma 10 (`Closure.Clo.type_computational`). *If $\sigma : \mathbf{TS}_{cand}$ is type-computational, then so is its closure $\mathbf{c}[\sigma]$.*

Proof. By the universal property of the closure operator, using Theorem 29. ☐

Theorem 11 (`Closure.Clo.monotonicity`). *For any candidate type system $\sigma : \mathbf{TS}_{cand}$, the function $\mathfrak{F}_\sigma : \mathbf{TS}_{cand} \to \mathbf{TS}_{cand}$ is monotone.*

Proof. By case on the type closure clauses, which are themselves monotone. ☐

Corollary 12 (`Closure.Clo.t`, `Closure.Clo.roll`). *By the Knaster-Tarski theorem, the function $\mathfrak{F}_\sigma$ has a least fixed point $\mu(\mathfrak{F}_\sigma)$.*

We will write $\mathbf{c}[-] : \mathbf{TS}_{cand} \to \mathbf{TS}_{cand}$ for the operator that takes $\sigma : \mathbf{TS}_{cand}$ to the fixed point $\mu(\mathfrak{F}_\sigma)$.

3.5 The full universe hierarchy

The next step in the construction is to build up the universe hierarchy. Following Allen [1987], we define the "spine" of the universe hierarchy as a sequence of type systems $\nu : \mathbf{TS}^{\mathbb{N}}_{cand}$ that contains at each level only types which evaluate to universes:

$$\nu_0 = \bot$$
$$\nu_{n+1} = \left\{ (\mathtt{U}_i, \mathcal{U}) \mid i \leq n \wedge \mathcal{U} \equiv \left\{ (A_0, A_1) \mid \mathbf{c}[\nu_i] \models A_0 \doteq A_1 \right\} \right\}^{\Downarrow}$$

The sequence above is well-defined by complete induction on the index.

Lemma 13 (`Tower.Spine.monotonicity`). *If $i \leq j$, then $\nu_i \sqsubseteq \nu_j$.*

Proof. By induction on i. ☐

Lemma 14 (`Tower.Spine.extensionality`). *Every spine level ν_i is extensional in the sense that it is the graph of a partial function $\mathbf{Prog}_0 \rightharpoonup \mathbf{rel}(\mathbf{Prog}_0)$.*

Proof. By case on i. ☐

Lemma 15 (`Tower.Spine.type_computational`). *Every spine level ν_i is type-computational.*

Proof. By case on i. ☐

Lemma 16 (`Tower.Spine.cper_valued`). *Every spine is valued in CPERs.*

Proof. By induction on i, using Lemmas 7, 10, 14 and Theorem 15. ☐

We are now equipped to define a new sequence of type systems which is at each level closed under all the ordinary type formers as well as smaller universes:

$$\tau_n \triangleq \mathbf{c}[\nu_n]$$

Lemma 17 (`Tower.monotonicity`). *If $i \leq j$, then $\tau_i \sqsubseteq \tau_j$.*

Proof. By the universal property of the closure operator and Lemma 13. ☐

Theorem 18 (Tower.extensionality,Tower.type_computational, Tower.cper_valued). *Each candidate type system τ_i is in fact a type system.*

Proof. τ_i is extensional immediately from Lemma 7 and the fact that the spine ν_i contains only types that evaluate to universes. It is type-computational by Lemmas 10 and 15. It is CPER-valued by Lemmas 9 and 16. □

Finally, we can capture the entire countable hierarchy in a single type system τ_ω, which is the join of the entire sequence:

$$\tau_\omega \triangleq \bigvee_{i:\mathbb{N}} \tau_i$$

When we explain the meaning of judgments, it will always be done with respect to this maximal type system.

Theorem 19 (τ_ω type system). *The ultimate candidate type system τ_ω is in fact a type system.*

3.6 Meaning explanation

In this section, we give a mathematical meaning explanation to the formal judgments of **CTT**$_{\circleddash}$:

1. Functional equality of elements $\Delta \mid \Gamma \gg M_0 \doteq M_1 \in A$ means that in clock context Δ and variable context Γ, M_0 and M_1 are equal elements of type A. This form of judgment requires that Γ, M_0, M_1, A mention only clocks from Δ, and that M_0, M_1, A mention only variables from Γ.
2. Untyped open conversion $\Delta \mid \Psi \vdash M_0 \leftrightarrow M_1$ means that M_0 and M_1 are Kleene equivalent in all their instantiations. This form of judgment requires that M_0, M_1 mention only clocks from Δ and variables from Ψ.

The meaning of judgments We interpret each formal judgment $\mathcal{J}$ as a proposition $[\![\mathcal{J}]\!] : \Omega$ in $\mathbf{S}_{\circleddash}$.

$$\begin{aligned}
&[\![\Delta \mid \Gamma \gg M_0 \doteq M_1 \in A]\!] \triangleq \\
&\quad \forall \varrho : \mathbb{K}^{|\Delta|}. \\
&\qquad \tau_\omega \models \|\Gamma\|\varrho \ \mathsf{ctx} \\
&\qquad \Rightarrow \tau_\omega \models \|\Gamma\|\varrho \gg \|A_0\|\varrho \doteq \|A_1\|\varrho \\
&\qquad \Rightarrow \tau_\omega \models \|\Gamma\|\varrho \gg \|M_0\|\varrho \doteq \|M_1\|\varrho \in \|A\|\varrho
\end{aligned}$$

$$[\![\Delta \mid \Psi \vdash M_0 \leftrightarrow M_1]\!] \triangleq \forall \varrho : \mathbb{K}^{|\Delta|}.\ \|M_0\|\varrho \approx_{|\Psi|} \|M_1\|\varrho$$

Observe that the usual presuppositions of the equality judgment (context validity and type functionality) are taken as *assumptions*: the principle can be summarized as "garbage in, garbage out". Dually, we could have chosen to regard them as consequences, which would lead to a slightly different collection of validated rules.

Canonicity at base type Write $\mathsf{2} : \mathbf{S}_{\circleddash}$ for the boolean object in our semantic framework which has two global elements $\mathsf{2}_0, \mathsf{2}_1 : \mathsf{2}$. Define an embedding $\lfloor - \rfloor_2$ from this object into our formal term language as follows:

$$\lfloor \mathsf{2}_0 \rfloor_2 = \mathtt{tt}$$
$$\lfloor \mathsf{2}_1 \rfloor_2 = \mathtt{ff}$$

Now we can state the canonicity theorem for $\mathbf{CTT}_{\circleddash}$.

Theorem 20 (Canonicity.canonicity)**.** *For any closed expression M such that $[\![\cdot \mid \cdot \gg M \doteq M \in \mathtt{bool}]\!]$, there exists some $b \in \mathsf{2}$ such that $[\![\cdot \mid \cdot \vdash M \leftrightarrow \lfloor b \rfloor_2]\!]$.*

Corollary 21. *The type theory $\mathbf{CTT}_{\circleddash}$ is consistent in the sense that there is no inhabitant of* void.

Theorem 20 is not immediately as strong as one would hope, but it implies a strong external result. Unfolding the $\forall\exists$ statement of Theorem 20, it is easy to see that at each individual world there *externally* exists a real boolean which has the desired property. To see that there is constructively a way to choose such a boolean externally (which is not automatically implied by the Kripke-Joyal semantics of $\forall\exists$ statements), it suffices to make the following observations.

In what follows, we will write $\mathbf{FTm}$ for the object of formal terms in $\mathbf{S}_{\circleddash}$.

1. Writing $[\![\mathtt{bool}]\!]$ for the subobject $\{M : \mathbf{FTm} \mid [\![\cdot \mid \cdot \gg M \doteq M \in \mathtt{bool}]\!]\}$, Theorem 20 states the following:

$$\mathbf{S}_{\circleddash} \models \forall M \in [\![\mathtt{bool}]\!].\ \exists b : \mathsf{2}.\ [\![\cdot \mid \cdot \vdash M \leftrightarrow \lfloor b \rfloor_2]\!]$$

2. Observe that internally, the boolean b is uniquely determined. This follows from the fact that $\lfloor b \rfloor_2$ is a value, and from the determinacy of the evaluation relation.

3. Therefore, we can strengthen the above to the following:

$$\mathbf{S}_{\circleddash} \models \forall M \in [\![\mathtt{bool}]\!].\ \exists! b : \mathsf{2}.\ [\![\cdot \mid \cdot \vdash M \leftrightarrow \lfloor b \rfloor_2]\!]$$

4. By the axiom of unique choice (which holds in every topos), the above is equivalent to the following:

$$\mathbf{S}_{\circleddash} \models \exists F : \mathsf{2}^{[\![\mathtt{bool}]\!]}.\ \forall M \in [\![\mathtt{bool}]\!].\ [\![\cdot \mid \cdot \vdash M \leftrightarrow \lfloor F(M) \rfloor_2]\!]$$

5. Unfolding this existential in the Kripke-Joyal semantics, choosing any world $\mathbf{U}$, we can exhibit externally a section of the presheaf exponential $\mathsf{2}^{[\![\mathtt{bool}]\!]}(\mathbf{U})$. Examining the construction of the presheaf exponential, this gives us a metatheoretic function to read back, from any definable formal term M which satisfies the typing judgment, the exact metatheoretic boolean it evaluates to.

This can be thought of as an *admissible statement* about the topos logic: from a formal term M and a proof that it is an element of type bool, we can extract an external boolean which has the desired property.

3.7 Validated rules

We have validated the following rules for $\mathbf{CTT}_{\odot}$ in our Coq formalization.

Conversion.symm
$$\frac{\Delta \mid \Psi \vdash M_0 \leftrightarrow M_1}{\Delta \mid \Psi \vdash M_1 \leftrightarrow M_0}$$

Conversion.Trans
$$\frac{\Delta \mid \Psi \vdash M_0 \leftrightarrow M_1 \qquad \Delta \mid \Psi \vdash M_1 \leftrightarrow M_2}{\Delta \mid \Psi \vdash M_0 \leftrightarrow M_2}$$

General.weakening
$$\frac{\Delta \mid \Gamma \gg M_0 \doteq M_1 \in A}{\Delta \mid \Gamma, x : B \gg M_0 \doteq M_1 \in A}$$

General.hypothesis
$$\frac{}{\Delta \mid \Gamma, x : \alpha \gg x \in \alpha}$$

General.conv_mem
$$\frac{\Delta \mid \Gamma \gg M_{01} \doteq M_1 \in \alpha \qquad \pi(\Gamma) \equiv \Psi \qquad \Delta \mid \Psi \vdash M_{00} \leftrightarrow M_{01}}{\Delta \mid \Gamma \gg M_{00} \doteq M_1 \in \alpha}$$

General.conv_ty
$$\frac{\Delta \mid \Gamma \gg M_0 \doteq M_1 \in A_1 \qquad \pi(\Gamma) \equiv \Psi \qquad \Delta \mid \Psi \vdash A_0 \leftrightarrow A_1}{\Delta \mid \Gamma \gg M_0 \doteq M_1 \in A_0}$$

General.eq_symm
$$\frac{\Delta \mid \Gamma \gg M_0 \doteq M_1 \in A}{\Delta \mid \Gamma \gg M_1 \doteq M_0 \in A}$$

General.eq_trans
$$\frac{\Delta \mid \Gamma \gg M_1 \doteq M_2 \in A \qquad \Delta \mid \Gamma \gg M_0 \doteq M_1 \in A}{\Delta \mid \Gamma \gg M_0 \doteq M_2 \in A}$$

General.replace_ty
$$\frac{\Delta \mid \Gamma \gg A_0 \doteq A_1 \in \mathsf{U}_i \qquad \Delta \mid \Gamma \gg M_0 \doteq M_1 \in A_0}{\Delta \mid \Gamma \gg M_0 \doteq M_1 \in A_1}$$

General.univ_formation
$$\frac{(i < j)}{\Delta \mid \Gamma \gg \mathsf{U}_i \in \mathsf{U}_j}$$

Unit.ax_equality
$$\frac{}{\Delta \mid \Gamma \gg \bigstar \in \mathsf{unit}}$$

Bool.univ_eq
$$\frac{}{\Delta \mid \Gamma \gg \mathsf{bool} \in \mathsf{U}_i}$$

Bool.tt_equality
$$\frac{}{\Delta \mid \Gamma \gg \mathsf{tt} \in \mathsf{bool}}$$

Bool.ff_equality
$$\frac{}{\Delta \mid \Gamma \gg \mathsf{ff} \in \mathsf{bool}}$$

Prod.univ_eq
$$\frac{\Delta \mid \Gamma \gg A_0 \doteq A_1 \in \mathsf{U}_i \qquad \Delta \mid \Gamma, x : A_0 \gg B_0 \doteq B_1 \in \mathsf{U}_i}{\Delta \mid \Gamma \gg (x : A_0) \times B_0 \doteq (x : A_1) \times B_1 \in \mathsf{U}_i}$$

Prod.intro
$$\frac{\begin{array}{c}\Delta \mid \Gamma \gg A \in \mathsf{U}_i \qquad \Delta \mid \Gamma, x : A \gg B \in \mathsf{U}_i \\ \Delta \mid \Gamma \gg M_{00} \doteq M_{10} \in A \qquad \Delta \mid \Gamma \gg M_{01} \doteq M_{11} \in [M_{00}/x]B\end{array}}{\Delta \mid \Gamma \gg \langle M_{00}, M_{01} \rangle \doteq \langle M_{10}, M_{11} \rangle \in (x : A) \times B}$$

Arr.univ_eq
$$\frac{\Delta \mid \Gamma \gg A_0 \doteq A_1 \in \mathsf{U}_i \qquad \Delta \mid \Gamma, x : A_0 \gg B_0 \doteq B_1 \in \mathsf{U}_i}{\Delta \mid \Gamma \gg (x : A_0) \to B_0 \doteq (x : A_1) \to B_1 \in \mathsf{U}_i}$$

Arr.intro
$$\frac{\begin{array}{c}\Delta \mid \Gamma \gg A \in \mathsf{U}_i \qquad \Delta \mid \Gamma, x : A \gg B \in \mathsf{U}_i \\ \Delta \mid \Gamma, x : A \gg M_0 \doteq M_1 \in B\end{array}}{\Delta \mid \Gamma \gg \lambda x. M_0 \doteq \lambda x. M_1 \in (x : A) \to B}$$

$$\frac{\Delta \mid \Gamma \gg A \in \mathtt{U}_i \qquad \Delta \mid \Gamma, x : A \gg B \in \mathtt{U}_i \qquad \Delta \mid \Gamma \gg M_0 \doteq M_1 \in (x : A) \to B \qquad \Delta \mid \Gamma \gg N_0 \doteq N_1 \in A}{\Delta \mid \Gamma \gg M_0(N_0) \doteq M_1(N_1) \in [N_0/x]B}\ \texttt{Arr.elim}$$

$$\frac{\Delta, k \mid \Gamma \gg A_0 \doteq A_1 \in \mathtt{U}_i}{\Delta \mid \Gamma \gg (k : \mathtt{clk}) \to A_0 \doteq (k : \mathtt{clk}) \to A_1 \in \mathtt{U}_i}\ \texttt{KArr.univ\_eq}$$

$$\frac{\Delta, k \mid \Gamma \gg A \doteq A \in \mathtt{U}_i \qquad \Delta, k \mid \Gamma \gg M_0 \doteq M_1 \in A}{\Delta \mid \Gamma \gg \lambda k. M_0 \doteq \lambda k. M_1 \in (k : \mathtt{clk}) \to A}\ \texttt{KArr.intro}$$

$$\frac{\Delta, k', k \mid \Gamma \gg A \doteq A \in \mathtt{U}_i \qquad \Delta, k' \mid \Gamma \gg M_0 \doteq M_1 \in (k : \mathtt{clk}) \to A}{\Delta, k' \mid \Gamma \gg M_0(k') \doteq M_1(k') \in [k'/k]A}\ \texttt{KArr.elim}$$

$$\frac{\Delta, k \mid \Gamma \gg A_0 \doteq A_1 \in \mathtt{U}_i}{\Delta \mid \Gamma \gg \{k \div \mathtt{clk}\} \to A_0 \doteq \{k \div \mathtt{clk}\} \to A_1 \in \mathtt{U}_i}\ \texttt{Isect.univ\_eq}$$

$$\frac{\Delta, k \mid \Gamma \gg M_0 \doteq M_1 \in A \qquad \Delta, k \mid \Gamma \gg A \in \mathtt{U}_i}{\Delta \mid \Gamma \gg M_0 \doteq M_1 \in \{k \div \mathtt{clk}\} \to A}\ \texttt{Isect.intro}$$

$$\frac{\Delta \mid \Gamma \gg A \in \mathtt{U}_i \qquad (k \notin \Delta)}{\Delta \mid \Gamma \gg A \doteq \{k \div \mathtt{clk}\} \to A \in \mathtt{U}_i}\ \texttt{Isect.irrelevance}$$

$$\frac{\Delta, k \mid \Gamma \gg A_0 \doteq A_1 \in \mathtt{U}_i \qquad \Delta, k \mid \Gamma \gg B_0 \doteq B_1 \in \mathtt{U}_i}{\Delta \mid \Gamma \gg \{k \div \mathtt{clk}\} \to ((x : A_0) \times B_0) \doteq (x : \{k \div \mathtt{clk}\} \to A_0) \times \{k \div \mathtt{clk}\} \to B_0 \in \mathtt{U}_i}\ \texttt{Isect.preserves\_sigma}$$

$$\frac{\Delta, k \mid \Gamma \gg A_0 \doteq A_1 \in \blacktriangleright_k \mathtt{U}_i}{\Delta, k \mid \Gamma \gg \blacktriangleright_k A_0 \doteq \blacktriangleright_k A_1 \in \mathtt{U}_i}\ \texttt{Later.univ\_eq}$$

$$\frac{\Delta, k \mid \Gamma \gg M_0 \doteq M_1 \in A \qquad \Delta, k \mid \Gamma \gg A \in \mathtt{U}_i}{\Delta, k \mid \Gamma \gg M_0 \doteq M_1 \in \blacktriangleright_k A}\ \texttt{Later.intro}$$

$$\frac{\Delta \mid \Gamma \gg \{k \div \mathtt{clk}\} \to A_0 \doteq \{k \div \mathtt{clk}\} \to A_1 \in \mathtt{U}_i}{\Delta \mid \Gamma \gg \{k \div \mathtt{clk}\} \to \blacktriangleright_k A_0 \doteq \{k \div \mathtt{clk}\} \to A_1 \in \mathtt{U}_i}\ \texttt{Later.force}$$

$$\frac{\Delta \mid \Gamma \gg A_0 \doteq A_1 \in \mathtt{U}_i \qquad \Delta \mid \Gamma, x : A \gg B_0 \doteq B_1 \in \blacktriangleright_k \mathtt{U}_i}{\Delta \mid \Gamma \gg \blacktriangleright_\kappa ((x : A_0) \to B_0) \doteq (x : \blacktriangleright_k A_1) \to \blacktriangleright_k B_1 \in \mathtt{U}_i}\ \texttt{Later.preserves\_pi}$$

$$\frac{\Delta \mid \Gamma \gg A_0 \doteq A_1 \in \mathtt{U}_i \qquad \Delta \mid \Gamma, x : A \gg B_0 \doteq B_1 \in \blacktriangleright_k \mathtt{U}_i}{\Delta \mid \Gamma \gg \blacktriangleright_\kappa ((x : A_0) \times B_0) \doteq (x : \blacktriangleright_k A_1) \times \blacktriangleright_k B_1 \in \mathtt{U}_i}\ \texttt{Later.preserves\_sigma}$$

`Later.induction`

$$\frac{\Delta, k \mid \Gamma, x : \blacktriangleright_k A \gg M_0 \doteq M_1 \in A}{\Delta, k \mid \Gamma \gg \mathtt{fix}\ x\ \mathtt{in}\ M_0 \doteq \mathtt{fix}\ x\ \mathtt{in}\ M_1 \in A}$$

3.8 Examples: revisiting streams

Using these rules, we can derive some typing lemmas for guarded streams and coinductive sequences of bits.

$$\begin{aligned} \mathtt{stream} &\triangleq \lambda k.\, \mathtt{fix}\ A\ \mathtt{in}\ \mathtt{bool} \times \blacktriangleright_k A \\ \mathtt{sequence} &\triangleq \{k \div \mathtt{clk}\} \to \mathtt{stream}\ k \\ \mathtt{ones} &\triangleq \mathtt{fix}\ x\ \mathtt{in}\ \langle \mathtt{tt}, x \rangle \end{aligned}$$

`Examples.BitStream_wf`
$\Delta \mid \Gamma \gg \mathtt{stream} \in (k : \mathtt{clk}) \to \mathtt{U}_i$

`Examples.BitSeq_wf`
$\Delta \mid \Gamma \gg \mathtt{sequence} \in \mathtt{U}_i$

`Examples.BitStream_unfold`
$\Delta, k \mid \Gamma \gg \mathtt{stream}\ k \doteq \mathtt{bool} \times \blacktriangleright_k \mathtt{stream}\ k \in \mathtt{U}_i$

`Examples.BitSeq_unfold`
$\Delta \mid \Gamma \gg \mathtt{sequence} \doteq \mathtt{bool} \times \mathtt{sequence} \in \mathtt{U}_i$

`Examples.Ones_wf_guarded`
$\Delta, k \mid \Gamma \gg \mathtt{ones} \in \mathtt{stream}\ k$

`Examples.Ones_wf_infinite`
$\Delta \mid \Gamma \gg \mathtt{ones} \in \mathtt{sequence}$

4 Survey of Related Work

4.1 Guarded Dependent Type Theory

The standard model of guarded recursion without clocks is the *topos of trees* $\widehat{\omega}$, the presheaves on the poset of natural numbers regarded as a category [Birkedal et al., 2011]. This topos can be regarded as a denotational model for a variant of Martin-Löf's extensional type theory equipped with the $\blacktriangleright$ modality. By indexing this topos over a category of clock contexts Δ, it is possible to develop a model of extensional type theory with clock quantification called **GDTT** [Bizjak et al., 2016, Bizjak and Møgelberg, 2015]. In order to justify a crucial *clock irrelevance* principle, it is necessary to index universes in clock contexts, i.e. $\mathcal{U}_\Delta$.

In the dependent setting, some difficulties arise when devising a *syntax* for the semantic type theory of this indexed category. In order to make sense of the "delayed application" operator $\circledast$ in the context of dependent function types, it was necessary to introduce a notion of *delayed substitution* $\xi \equiv [\overrightarrow{x \leftarrow e}]$ which pervades the term language, introducing term formers like $\triangleright^k \xi.A$ and $\mathrm{next}^k \xi.e$. On the bright side, delayed application can be defined in terms of delayed substitution.

However, the equational theory for delayed substitutions is fairly sophisticated, and an operational (computational) interpretation of **GDTT** has not yet been proposed at the time this article was written; as such, a canonicity theorem for this system is still forthcoming.

4.2 Orthogonality and clock irrelevance

In a more recent development [Bizjak and Møgelberg, 2017], a denotational model of **GDTT** has been developed that differs from that of Bizjak and Møgelberg [2015] in a few crucial ways.

Unified base category The fibered topos presentation of the Bizjak and Møgelberg [2015] work has been replaced with a presheaf topos over a single unified base category, discovered independently from the unified base category which we introduce in Section 3.1. Taking presheaves over this unified base category simplifies the model significantly, and also makes available the standard solution to the substitution coherence problem for (denotational) presheaf models of dependent type theory.[7]

The proposed base category of Bizjak and Møgelberg [2017] differs from ours mainly in that they allow empty worlds, whereas we restrict our base category to those worlds which contain at least a single clock.

Orthogonality Bizjak and Møgelberg define a presheaf of clocks $\boldsymbol{C}$ which is the same as our object of clocks $\mathbb{K}$ which we introduce in Section 3.1; then, the clock quantifier is represented in the internal language of their presheaf topos as a dependent product over $\boldsymbol{C}$, i.e. $\prod_{x:\boldsymbol{C}} A(x)$.

Defined in this way, the clock quantifier cannot be a priori parametric with respect to clocks / time objects; therefore, in order to validate the clock irrelevance axiom, the authors have identified an orthogonality condition on objects, which in essence closes the internal language of the presheaf topos under just those types which are compatible with the irrelevance principle for the clock quantifier.

Unfortunately, the subtopos of time-orthogonal objects does not contain the standard Hofmann-Streicher universes, because universes necessarily classify types that depend on clocks in an essential way. In order to resolve this problem, the standard presheaf-theoretic universe $\boldsymbol{U}$ is replaced with a family of universes $\boldsymbol{U}_\Delta$ for each clock context Δ; each universe $\boldsymbol{U}_\Delta$ classifies the types which may depend only on the clocks in Δ.

Discussion Temporarily abstracting away from the differences between a denotational account of **GDTT** and our operational account of type theory, we can briefly summarize the difference between our approaches to clock quantification and irrelevance.

The approach of Bizjak and Møgelberg [2017] is in essence to define clock quantification as a dependent (cartesian) product, and then restrict the available semantic

[7] This is to use an alternative construction of the slice categories $\widehat{\mathbb{C}}/X$, as the presheaves on the total category of X.

constructions to precisely those which treat clocks parametrically; then, within this subcategory, the clock quantifier can itself be regarded as a parametric quantifier (because all counterexamples have been muted).

Our approach is instead to define clock quantifiers which *intrinsically* behave in the desired way, rather than starting with only a proof-relevant quantifier and ruling out observations of its non-parametric character using a global orthogonality condition. To that end, we have defined two separate clock quantifiers which decompose the two disjoint uses of $\forall\kappa$ from **GDTT**:

1. A parametric quantifier $\{k \div \mathtt{clk}\} \to A$ for expressing that a program exhibits a behavior relative to all clocks simultaneously. Semantically, this is an intersection, though we expect that a more refined perspective will arise as we explore other kinds of model where the intersection may not be available.

2. A non-parametric quantifier $(k : \mathtt{clk}) \to A$ for internalizing a family of objects which varies in a clock; semantically this is the cartesian product of a clock-indexed family of types (i.e. the right adjoint to weakening). *A priori* there is no need for this quantifier to behave parametrically, as this is neither demanded nor desired when forming families of objects.

In this way, we have managed to avoid imposing any global orthogonality condition on the objects of our semantic model, leading to a smoother treatment of universes that avoids indexing in clock contexts.

4.3 Guarded Cubical Type Theory

One way to achieve a decidable typing judgment for **GDTT** is to adopt an intensional equality, and replace various *judgmental* principles with propositional axioms (such as the unfolding rule for `fix`, as well as several other principles having to do with identity types which are validated in extensional **GDTT**). However, such axioms are disruptive to the computational character of type theory.

A more refined and well-behaved version of this idea can be found in Guarded Cubical Type Theory (**GCTT**) by Birkedal et al. [2016], where `fix` is actually exhibited as a higher-dimensional term, a *line* or *path* between a formal fixed point and its one-step unfolding.

GCTT currently supports only a single clock, but it is plausible that it could be extended in the same way as **GDTT** extends the internal type theory of the topos of trees. Although **GCTT** does not at the time of writing have a decidable typing result, nor a strong normalization theorem, we are confident that these can be achieved in the future in light of the intensional judgmental equality and the restricted unfoldings of fixed points.

4.4 Clocked Type Theory

Recently, an alternative to **GDTT** called *Clocked Type Theory* (**CloTT**) has been proposed, which enjoys a computational interpretation with a canonicity result [Bahr et al., 2017]; it is plausible that Clocked Type Theory shall have a decidable typing relation. Notably, Clocked Type Theory does not validate any clock irrelevance rule; the authors propose to address this in a *cubical* version of **CloTT** by adding a special path axiom which realizes this principle, by analogy with the technique used in **GCTT** to account for restricted unfoldings of fixed points. In the presence of this axiom, canonicity for **CloTT** can still be made to hold in the context which contains only a single clock.

Discussion Clocked Type Theory looks like a promising path toward a well-behaved intrinsic account of guarded recursion with clocks. In the present paper, our efforts have been focused exclusively on developing the behavioral account of guarded type theory in the style of Martin-Löf's meaning explanation, in which programs can be regarded as existing separately from their types; here, general recursive programs can be written and shown to be (causal, productive, total) in a semantic sense, using the type theory as a program logic.

We perceive, however, that virtue lies in pursuing the intrinsic path, especially as far as implementability are concerned. The calculus developed in Bahr et al. [2017] (and more recently, the ideas contained in Clouston et al. [2018]) are likely to provide the basis for a syntactic account of guarded recursion which is sound for our model, but closer to implementation.

4.5 Sized Types and size quantifiers

Our decomposition of the quantifier $\forall\kappa$ from **GDTT** into a parametric part $\{k \div \mathtt{clk}\} \to A$ and a non-parametric part $(k : \mathtt{clk}) \to A$ mirrors the state of affairs in the literature on sized types, which is another account of type-based guarded recursion [Abel et al., 2017].

5 Perspective and Future Work

We have developed and formalized a computational account of guarded dependent type theory with clocks, enjoying several desirable characteristics not found together in other existing models: computational canonicity, clock irrelevance and ordinary universes. We have made the following contributions toward a simpler, more computational account of guarded dependent type theory:

Implementation, proof theory, and syntax We have not yet tackled the project of developing an ergonomic proof theory for $\mathbf{CTT}_{\odot}$ which can be used to interact with the semantics presented here. The natural deduction style rules which we have given here are, while convenient for paper presentations, not what one would use in a serious implementation. To build a proof theory for $\mathbf{CTT}_{\odot}$, we must negotiate new forms of judgment with decidable presupposition.

Therefore, while we have indeed developed a programming language for guarded type theory with clocks that omits explicit syntax for delayed substitutions, this should be understood in terms of the conceptual order of semantics and proof theory which is endemic in computational type theory. In particular, while our programming language and type theory has no need for such a construct, in a proof language for $\mathbf{CTT}_{\odot}$ it would be necessary to account for the syntactic structure of the later modality's elimination;

we anticipate that ideas from Bahr et al. [2017] and Clouston et al. [2018] will be highly relevant.

Application to denotational semantics In the future, we are interested in extending our work to a denotational account of guarded dependent type theory with clocks which uses the ordinary non-indexed presheaf-topos-theoretic universe. While our results have been developed in the context of computational type theory and operational semantics, we believe that the insight which enabled us to combine clock irrelevance with ordinary universes is more broadly applicable.

A Semantic Universe

In this appendix, we give some further details of the semantic universe $\mathbf{S}_{\circleddash}$.

A.1 Internal Logic and Kripke-Joyal Semantics

Using a tool called Kripke-Joyal semantics (a topos-theoretic generalization of Beth/Kripke-forcing) it is possible to interpret statements in the internal language of $\mathbf{S}_{\circleddash}$ into ordinary, external mathematical language. We will write forcing clauses $\mathbf{U} \Vdash \varphi(\alpha)$ meaning that at world $\mathbf{U} : \circleddash$, the predicate φ holds of the element $\alpha : X(\mathbf{U})$. The forcing clauses for the predicates of our internal logic are summarized in Figure 6.

It will simplify many of our proofs to formalize some proof techniques for establishing that a formula headed by multiple universal quantifiers is valid in $\mathbf{S}_{\circleddash}$, i.e. true at each world.

Lemma 22. *To show that a formula* $\forall y : Y.\ \varphi(\alpha, y)$ *is true for all worlds* $\mathbf{U}$ *and elements* $\alpha \in X(\mathbf{U})$ *in* $\mathbf{S}_{\circleddash}$*, it suffices to establish externally the following statement:*

$$\forall \mathbf{U} : \circleddash.\ \forall \alpha \in X(\mathbf{U}).\ \forall \beta \in Y(\mathbf{U}).\ \mathbf{U} \Vdash \varphi(\alpha, \beta)$$

Proof. Fixing a world $\mathbf{U}$ and an element $\alpha \in X(\mathbf{U})$, our original formula unfolds to the following in the Kripke-Joyal semantics:

$$\forall \mathbf{V} : \circleddash.\ \forall \rho : \mathbf{V} \to \mathbf{U}.\ \forall \beta \in Y(\mathbf{V}).\ \mathbf{V} \Vdash \varphi(\rho^* \alpha, \beta)$$

Fix $\mathbf{V} : \circleddash$, $\rho : \mathbf{V} \to \mathbf{U}$ and $\beta \in Y(\mathbf{V})$. By instantiating our assumption with $\mathbf{V}$, $\rho^*\alpha$ and β, we have $\mathbf{V} \Vdash \varphi(\rho^*\alpha, \beta)$. □

Lemma 23. *To show that a formula* $\forall \overrightarrow{y_i : Y_i}.\ \varphi(\overrightarrow{y_i}, \alpha)$ *is true at all worlds* $\mathbf{U}$ *and elements* $\alpha \in X(\mathbf{U})$*, it suffices to establish the following external statement:*

$$\forall \mathbf{U} : \circleddash.\ \forall \overrightarrow{y_i \in y_i(\mathbf{U})}.\ \mathbf{U} \Vdash \varphi(\overrightarrow{y_i}, \alpha)$$

Proof. Observe that our original formula is logically equivalent to the following one with only a single quantifier:

$$\forall y : \textstyle\prod_i Y_i.\ \varphi(\overrightarrow{\pi_i(y)}, \alpha)$$

Therefore, our goal follows from Lemma 22. □

$$\boxed{\mathbf{U} \Vdash \varphi(\alpha) \;\textit{presupposing}\; \varphi \rightarrowtail X : \mathbf{S}_{\circleddash}\,,\;\; \alpha \in X(\mathbf{U})}$$

$$\begin{aligned}
\mathbf{U} \Vdash \varphi(\alpha) \vee \psi(\alpha) &\equiv \mathbf{U} \Vdash \varphi(\alpha) \vee \mathbf{U} \Vdash \psi(\alpha) \\
\mathbf{U} \Vdash \varphi(\alpha) \wedge \psi(\alpha) &\equiv \mathbf{U} \Vdash \varphi(\alpha) \wedge \mathbf{U} \Vdash \psi(\alpha) \\
\mathbf{U} \Vdash \varphi(\alpha) \Rightarrow \psi(\alpha) &\equiv \forall \rho : \mathbf{V} \to \mathbf{U}.\, \mathbf{V} \Vdash \varphi(\rho^*\alpha) \Rightarrow \mathbf{V} \Vdash \psi(\rho^*\alpha) \\
\mathbf{U} \Vdash \forall y : Y.\, \varphi(\alpha, y) &\equiv \forall \rho : \mathbf{V} \to \mathbf{U}.\, \forall \beta \in Y(\mathbf{V}).\, \mathbf{V} \Vdash \varphi(\rho^*\alpha, \beta) \\
\mathbf{U} \Vdash \exists y : Y.\, \varphi(\alpha, y) &\equiv \exists \beta \in Y(\mathbf{U}).\, \mathbf{U} \Vdash \varphi(\alpha, \beta) \\
\mathbf{U} \Vdash \triangleright_\kappa \varphi(\alpha) &\equiv \begin{cases} \top & \textbf{if } \partial_U(\kappa) \equiv 0 \\ \mathbf{U}[\kappa \mapsto n] \Vdash \varphi([\kappa \mathrel{+}= 1]^*\alpha) & \textbf{if } \partial_U(\kappa) \equiv n+1 \end{cases}
\end{aligned}$$

Figure 6: Forcing clauses for the internal logic of $\mathbf{S}_{\circleddash}$.

Lemma 24. *To show that a formula $\forall \overrightarrow{y_i : Y_i}.\, \overrightarrow{\varphi_j(\overrightarrow{y_i}, \alpha)} \Rightarrow \psi(\overrightarrow{y_i}, \alpha)$ is true at all worlds $\mathbf{U}$ and elements $\alpha \in X(\mathbf{U})$, it suffices to establish the following external statement:*

$$\forall \mathbf{U} : \circleddash.\, \forall \overrightarrow{y_i \in Y_i(\mathbf{U})}.\, \overrightarrow{\mathbf{U} \Vdash \varphi(\overrightarrow{y_i}, \alpha)} \Rightarrow \mathbf{U} \Vdash \psi(\overrightarrow{y_i}, \alpha)$$

Proof. Observe that any implication $\varphi \Rightarrow \psi$ in the internal logic can be equivalently written as a universal quantification over a subobject comprehension $\forall x : \{x : 1 \mid \varphi\}.\, \psi$ Therefore, our lemma follows from Lemma 22. □

A.2 Semantic Lemmas

Theorem 25 (Local clock)**.** *The formula $\exists \kappa : \mathbb{K}.\, \top$ is true in the internal logic of $\mathbf{S}_{\circleddash}$.*

Proof. It suffices to validate this formula at each world $\mathbf{U}$, i.e. to establish $\mathbf{U} \Vdash \exists \kappa : \mathbb{K}.\, \top$, which is to say (externally) that $\exists \kappa : \mathbb{K}(\mathbf{U}).\, \top$. This reduces to showing that the hom set $U \to \bullet^1$ in $\mathbb{F}_+$ is non-empty, which is true because $\mathbb{F}_+$ is a category of *non-empty* finite products. □

Note that Theorem 25 does *not* entail the existence of a global element of $\mathbb{K}$ (i.e. a morphism $1 \to \mathbb{K}$). In our development, we have no need for a global clock; we only require that a clock "merely exists" according to the existential quantifier of the topos logic.

Corollary 26 (Clock irrelevance)**.** *The formula $\forall \varphi : \Omega.\, \varphi \equiv \forall \kappa : \mathbb{K}.\, \varphi$ holds in the internal logic.*

Proof. We will reason internally: fix $\varphi : \Omega$. By propositional extensionality we need to show that $\varphi \Rightarrow \forall \kappa : \mathbb{K}.\, \varphi$ and $\forall \kappa : \mathbb{K}.\, \varphi \Rightarrow \varphi$. The first direction is trivial; for the second direction, observe that from Theorem 25, using the elimination rule for the existential quantifier, we may fix a clock $\kappa_0 : \mathbb{K}$; using this clock, by the elimination rule of the universal quantifier, we have our goal φ. □

Theorem 27. *We can delete a later modality from under an appropriate quantification, in the sense that the following formula is true in the internal logic:*

$$\forall \varphi : \Omega^{\mathbb{K}}.\ (\forall \kappa : \mathbb{K}. \triangleright_\kappa \varphi(\kappa)) \Rightarrow \forall \kappa : \mathbb{K}. \varphi(\kappa)$$

Proof. We will establish this principle using the Kripke-Joyal semantics; using Lemma 24, we fix a world $\mathbf{U}$ and a predicate $\varphi \in \Omega^{\mathbb{K}}(\mathbf{U})$ such that $\mathbf{U} \Vdash \forall \kappa : \mathbb{K}. \triangleright_\kappa \varphi(\kappa)$, to show $\mathbf{U} \Vdash \forall \kappa : \mathbb{K}. \varphi(\kappa)$.

Observe that our goal is equivalent to the following external statement, writing $\pi_1[n], \pi_2[n]$ for the projections of $\mathbf{U}$ and $(1, [n])$, respectively, from the extended world $(U+1, [\partial_U, n])$:[8]

$$\forall n \in \omega.\ (U + 1, [\partial_U, n]) \Vdash (\pi_1[n])^* \varphi(\pi_2[n]) \tag{G1}$$

In the same way, our premise can be rewritten as follows:

$$\forall n \in \omega.\ (U + 1, [\partial_U, n]) \Vdash \triangleright_{(\pi_2[n])} (\pi_1[n])^* \varphi(\pi_2[n]) \tag{H1}$$

To establish (G1), fix $m \in \omega$; our goal now becomes:

$$(U + 1, [\partial_U, m]) \Vdash (\pi_1[m])^* \varphi(\pi_2[m]) \tag{G2}$$

Next instantiate (H1) with $n \equiv m + 1$, yielding:

$$(U + 1, [\partial_U, m + 1]) \Vdash \triangleright_{(\pi_2[m+1])} (\pi_1[m + 1])^* \varphi(\pi_2[m + 1]) \tag{H2}$$

Using the forcing clause for the later modality, we see that (H2) is actually the same as the goal (G2). □

Theorem 28. *We have the following unit law in the internal logic:*

$$\forall \kappa : \mathbb{K}. \forall \varphi : \Omega.\ \varphi \Rightarrow \triangleright_\kappa \varphi$$

Proof. By Lemma 24, it suffices to fix a world $\mathbf{U}$ and elements $\kappa \in \mathbb{K}(\mathbf{U})$, $\varphi \in \Omega(\mathbf{U})$ such that $\mathbf{U} \Vdash \varphi$. We need to show that $\mathbf{U} \Vdash \triangleright_\kappa \varphi$. Proceed by case on $\partial_U(\kappa)$:

Case $\partial_U(\kappa) \equiv 0$. Immediate.

Case $\partial_U(\kappa) \equiv n + 1$. We need to show that $\mathbf{U}[\kappa \mapsto n] \Vdash [\kappa \mathrel{+}= 1]^* \varphi$; this follows by reindexing our assumption that $\mathbf{U} \Vdash \varphi$.

□

Theorem 29. *The later modality commutes with conjunction:*

$$\forall \kappa : \mathbb{K}. \forall \varphi, \psi : \Omega.\ \triangleright_\kappa(\varphi \wedge \psi) \equiv (\triangleright_\kappa \varphi \wedge \triangleright_\kappa \psi)$$

[8] This is a special case of the "alternative" forcing clause (vi′) for the universal quantifier in Kripke-Joyal semantics, as given in Mac Lane and Moerdijk [1992, p. 305].

Proof. It suffices to prove that each direction of this quantified equation is valid at all worlds:

$$\forall \kappa : \mathbb{K}. \forall \varphi, \psi : \Omega. \rhd_\kappa(\varphi \wedge \psi) \Rightarrow (\rhd_\kappa \varphi \wedge \rhd_\kappa \psi) \tag{$\Rightarrow$}$$

$$\forall \kappa : \mathbb{K}. \forall \varphi, \psi : \Omega. (\rhd_\kappa \varphi \wedge \rhd_\kappa \psi) \Rightarrow \rhd_\kappa(\varphi \wedge \psi) \tag{$\Leftarrow$}$$

($\Rightarrow$) Using Lemma 24, we fix a world $\mathbf{U}$ and elements $\kappa \in \mathbb{K}(\mathbf{U})$, $\varphi, \psi \in \Omega(\mathbf{U})$ such that $\mathbf{U} \Vdash \rhd_\kappa(\varphi \wedge \psi)$. We need to show that $\mathbf{U} \Vdash \rhd_\kappa \varphi \wedge \rhd_\kappa \psi$. Proceed by case on $\partial_U(\kappa)$:

Case $\partial_U(\kappa) \equiv 0$. Immediate.

Case $\partial_U(\kappa) \equiv n + 1$. Then our assumption is equal to $\mathbf{U}[\kappa \mapsto n] \Vdash [\kappa \mathrel{+}= 1]^* \varphi \wedge [\kappa \mathrel{+}= 1]^* \psi$, which is exactly the same as our goal.

($\Leftarrow$) This direction is analogous. □

Corollary 30. *The later modality is monotonic:*

$$\forall \kappa : \mathbb{K}. \forall \varphi, \psi : \Omega. (\varphi \Rightarrow \psi) \Rightarrow \rhd_\kappa \varphi \Rightarrow \rhd_\kappa \psi$$

Proof. This is a well-known corollary of Theorem 29, following for purely algebraic reasons. Reasoning internally, fix $\kappa : \mathbb{K}$ and $\varphi, \psi : \Omega$ such that $\varphi \Rightarrow \psi$ and $\rhd_\kappa \varphi$; we need to show $\rhd_\kappa \psi$.

First, observe that $(\rhd_\kappa \varphi \wedge \rhd_\kappa \psi) \equiv \rhd_\kappa \varphi$. To show that this is the case, by Theorem 29 it suffices to show that $\rhd_\kappa(\varphi \wedge \psi) \equiv \rhd_\kappa \varphi$. This holds, because $\varphi \wedge \psi \equiv \varphi$: $\varphi \wedge \psi \Rightarrow \varphi$ is trivial, and $\varphi \Rightarrow \varphi \wedge \psi$ follows from our assumption $\varphi \Rightarrow \psi$.

Returning to our main goal $\rhd_\kappa \psi$, using the above, we may replace our assumption $\rhd_\kappa \varphi$ with $\rhd_\kappa \varphi \wedge \rhd_\kappa \psi$, whence we have immediately $\rhd_\kappa \psi$. □

Theorem 31. *The later modality commutes with implication:*

$$\forall \kappa : \mathbb{K}. \forall \varphi, \psi : \Omega. \rhd_\kappa(\varphi \Rightarrow \psi) \equiv (\rhd_\kappa \varphi \Rightarrow \rhd_\kappa \psi)$$

Proof. As in Theorem 29, it will be simplest to show that each direction of the quantified equation is valid at all worlds:

$$\forall \kappa : \mathbb{K}. \forall \varphi, \psi : \Omega. \rhd_\kappa(\varphi \Rightarrow \psi) \Rightarrow (\rhd_\kappa \varphi \Rightarrow \rhd_\kappa \psi) \tag{$\Rightarrow$}$$

$$\forall \kappa : \mathbb{K}. \forall \varphi, \psi : \Omega. (\rhd_\kappa \varphi \Rightarrow \rhd_\kappa \psi) \Rightarrow \rhd_\kappa(\varphi \Rightarrow \psi) \tag{$\Leftarrow$}$$

($\Rightarrow$) We will reason algebraically:

$$\begin{aligned} & \rhd_\kappa(\varphi \Rightarrow \psi) \Rightarrow (\rhd_\kappa \varphi \Rightarrow \rhd_\kappa \psi) \\ & \equiv \rhd_\kappa(\varphi \Rightarrow \psi) \wedge \rhd_\kappa \varphi \Rightarrow \rhd_\kappa \psi && (\wedge \dashv \Rightarrow) \\ & \equiv \rhd_\kappa((\varphi \Rightarrow \psi) \wedge \varphi) \Rightarrow \rhd_\kappa \psi && (\text{Theorem 29}) \end{aligned}$$

Now, assuming $\rhd_\kappa((\varphi \Rightarrow \psi) \wedge \varphi)$, we have to show $\rhd_\kappa \psi$. Observe that $((\varphi \Rightarrow \psi) \wedge \varphi) \Rightarrow \psi$; therefore, by monotonicity (Corollary 30) we have $\rhd_\kappa \psi$, which was our goal.

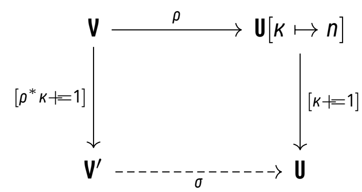
($\Leftarrow$) We will reason externally through Lemma 24; fixing a world $\mathbf{U}$ and elements $\kappa \in \mathbb{K}(\mathbf{U})$, $\varphi, \psi \in \Omega(\mathbf{U})$ such that $\mathbf{U} \Vdash \rhd_\kappa \varphi \Rightarrow \rhd_\kappa \psi$, we need to show that $\mathbf{U} \Vdash \rhd_\kappa(\varphi \Rightarrow \psi)$. Proceed by case on $\partial_U(\kappa)$:

Case $\partial_U(\kappa) \equiv 0$. Immediate.

Case $\partial_U(\kappa) \equiv n + 1$. Now we need to show:

$$\mathbf{U}[\kappa \mapsto n] \Vdash [\kappa \mathrel{+}= 1]^* \varphi \Rightarrow [\kappa \mathrel{+}= 1]^* \psi$$

Fix $\rho : \mathbf{V} \to \mathbf{U}[\kappa \mapsto n]$ such $\mathbf{V} \Vdash \rho^*[\kappa \mathrel{+}= 1]^* \varphi$ to show that $\mathbf{V} \Vdash \rho^*[\kappa \mathrel{+}= 1]^* \psi$. Writing $\mathbf{V}'$ for $\mathbf{V}[\rho^*\kappa \mapsto \partial_V(\rho^*\kappa) + 1]$, observe that we can form a map $\sigma : \mathbf{V}' \to \mathbf{U}$ such that the following diagram commutes:



As a map in $\mathbb{F}_+$, σ is the same as ρ; to see that it is a map in $\circleddash$, observe that $m_1 + 1 \leq m_2 + 1$ iff $m_1 \leq m_2$. Now, we have assumed $\mathbf{U} \Vdash \triangleright_\kappa \varphi \Rightarrow \triangleright_\kappa \psi$; instantiating this assumption at σ, we have the following external implication:

$$\mathbf{V}' \Vdash \triangleright_{\sigma^*\kappa} \sigma^* \varphi \Rightarrow \mathbf{V}' \Vdash \triangleright_{\sigma^*\kappa} \sigma^* \varphi$$

Observing that the action of σ on κ is the same as the action of ρ on κ (since $\mathbb{K}$ is oblivious to time assignments), we can unfold our implication further:

$$\mathbf{V} \Vdash [\rho^*\kappa \mathrel{+}= 1]^* \sigma^* \varphi \Rightarrow \mathbf{V} \Vdash [\rho^*\kappa \mathrel{+}= 1]^* \sigma^* \psi$$

By the diagram above, we calculate the composition of reindexings:

$$\mathbf{V} \Vdash \rho^*[\kappa \mathrel{+}= 1]^* \varphi \Rightarrow \mathbf{V} \Vdash \rho^*[\kappa \mathrel{+}= 1]^* \psi$$

But we have already assumed $\mathbf{V} \Vdash \rho^*[\kappa \mathrel{+}= 1]^* \varphi$, and $\mathbf{V} \Vdash \rho^*[\kappa \mathrel{+}= 1]^* \psi$ is what we were trying to prove.

□

Theorem 32 (Löb induction). *We have the following* Löb induction *principle for the later modality:*

$$\forall \kappa : \mathbb{K}. \forall \varphi : \Omega. (\triangleright_\kappa \varphi \Rightarrow \varphi) \Rightarrow \varphi$$

Proof. By Lemma 24, it suffices to show that for all $\mathbf{U}$ and $\kappa \in \mathbb{K}(\mathbf{U}, \kappa)$, we have the external proposition $P(\mathbf{U}, \kappa)$, defined as follows:

$$P(\mathbf{U}, \boldsymbol{\kappa}) \triangleq \forall \varphi \in \Omega(\mathbf{U}). (\mathbf{U} \Vdash \triangleright_\kappa \varphi \Rightarrow \varphi) \Rightarrow \mathbf{U} \Vdash \varphi$$

We proceed by induction on $\partial_U(\kappa)$; in what follows, we will write $\mathbf{U}_n$ for $\mathbf{U}[\kappa \mapsto n]$.

Case $\partial_U(\kappa) \equiv 0$. We need to establish $P(\mathbf{U}_0, \kappa)$. Fix $\varphi \in \Omega(\mathbf{U}_0)$ such that $\mathbf{U}_0 \Vdash \triangleright_\kappa \varphi \Rightarrow \varphi$, to show $\mathbf{U}_0 \Vdash \varphi$. Instantiating our assumption with the identity morphism, it suffices to show that $\mathbf{U}_0 \Vdash \triangleright_\kappa \varphi$; but this is trivial, since the value of κ is 0.

Case $\partial_U(\kappa) \equiv n+1$. Our induction hypothesis is $P(\mathbf{U}_n, \kappa)$, and we need to show $P(\mathbf{U}_{n+1}, \kappa)$. Fix $\varphi \in \Omega(\mathbf{U}_{n+1})$ such that $\mathbf{U}_{n+1} \Vdash \triangleright_\kappa\varphi \Rightarrow \varphi$, to show $\mathbf{U}_{n+1} \Vdash \varphi$. Instantiating this assumption with the identity morphism, it suffices to show $\mathbf{U}_{n+1} \Vdash \triangleright_\kappa\varphi$, which is the same as $\mathbf{U}_n \Vdash [\kappa \mathrel{+}= 1]^*\varphi$. To establish this, we instantiate our induction hypothesis with $[\kappa \mathrel{+}= 1]^*\varphi$, and it remains to show $\mathbf{U}_n \Vdash \triangleright_\kappa[\kappa \mathrel{+}= 1]^*\varphi \Rightarrow [\kappa \mathrel{+}= 1]^*\varphi$. We have assumed $\mathbf{U}_{n+1} \Vdash \triangleright_\kappa\varphi \Rightarrow \varphi$, so by reindexing we have $\mathbf{U}_n \Vdash \triangleright_{[\kappa \mathrel{+}= 1]^*\kappa}[\kappa \mathrel{+}= 1]^*\varphi \Rightarrow [\kappa \mathrel{+}= 1]^*\varphi$. This is the same as our goal, because $[\kappa \mathrel{+}= 1]^*\kappa \equiv \kappa$.

□

Definition 33 (Totality)**.** An object $X : \mathbf{S}_{\circledcirc}$ is called *total* if its action on all restriction maps $[\kappa \mathrel{+}= n]$ is a surjection.[9]

Definition 34 (Inhabitedness)**.** An object $X : \mathbf{S}_{\circledcirc}$ is called *inhabited* when the formula $\exists x : X.\ \top$ is valid in the internal logic of $\mathbf{S}_{\circledcirc}$.

The constant objects (such as $\mathbb{N}$) are all total; but note that an object may be *total* without being *constant*: for instance, the subobject classifier is total. In our development, we have only needed the fact that $\mathbb{N}$ is total.

Theorem 35. *Suppose that an object $Y : \mathbf{S}_{\circledcirc}$ is total and inhabited (Definitions 33,34). Then, if we* later *have an element of Y that satisfies φ, we can also* now *exhibit an element of Y that* later *satisfies φ.*

$$\forall \kappa : \mathbb{K}.\ \forall \varphi : \Omega^Y.\ \triangleright_\kappa(\exists y : Y.\ \varphi(y)) \Rightarrow \exists y : Y.\ \triangleright_\kappa\varphi(y)$$

Proof. Using Lemma 24, fix a world $\mathbf{U}$ and a predicate $\varphi \in \Omega^Y(\mathbf{U})$ such that $\mathbf{U} \Vdash \triangleright_\kappa(\exists y : Y.\ \varphi(y))$; we need to show $\mathbf{U} \Vdash \exists y : Y.\ \triangleright_\kappa\varphi(y)$. Proceed by case on $\partial_U(\kappa)$:

Case $\partial_U(\kappa) \equiv 0$. Then it suffices to exhibit an arbitrary element of Y at $\mathbf{U}$, since the predicate is trivial at this world. But we have already assumed Y to be inhabited, so we are done.

Case $\partial_U(\kappa) \equiv n+1$. In this case, our assumption amounts to the following external existential:

$$\mathbf{U}[\kappa \mapsto n] \Vdash \exists y : Y.\ [\kappa \mathrel{+}= 1]^*\varphi(y)$$

Unfolding the forcing clause for existential quantification, this means that we have an element $\alpha \in Y(\mathbf{U}[\kappa \mapsto n])$ such that the following holds:

$$\mathbf{U}[\kappa \mapsto n] \Vdash [\kappa \mathrel{+}= 1]^*\varphi(\alpha) \tag{H}$$

Our goal was to show that $\mathbf{U} \Vdash \exists y : Y.\ \triangleright_\kappa\varphi(y)$; because Y is total, from α we can get an element $\beta \in Y(\mathbf{U})$ such that $\alpha \equiv [\kappa \mathrel{+}= 1]^*\beta$.

Now it remains only to show that $\mathbf{U} \Vdash \triangleright_\kappa\varphi(\beta)$; at this world, this is the same as to say that $\mathbf{U}[\kappa \mapsto n] \Vdash \triangleright_\kappa[\kappa \mathrel{+}= 1]^*\varphi([\kappa \mathrel{+}= 1]^*\beta)$. Because $\alpha \equiv [\kappa \mathrel{+}= 1]^*\beta$, this is the same as (H).

□

[9] This is the analogous condition to the one described in Birkedal et al. [2011], generalized to the case of multiple clocks.

References

Andreas Abel, Andrea Vezzosi, and Theo Winterhalter. Normalization by evaluation for sized dependent types. *Proc. ACM Program. Lang.*, 1(ICFP):33:1–33:30, August 2017. ISSN 2475-1421.

S.F. Allen, M. Bickford, R.L. Constable, R. Eaton, C. Kreitz, L. Lorigo, and E. Moran. Innovations in computational type theory using nuprl. *Journal of Applied Logic*, 4(4): 428 – 469, 2006. ISSN 1570-8683. Towards Computer Aided Mathematics.

Stuart Frazier Allen. *A non-type-theoretic semantics for type-theoretic language*. PhD thesis, Cornell University, Ithaca, NY, USA, 1987.

Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. In Luke Ong, editor, *Foundations of Software Science and Computational Structures: 13th International Conference, FOSSACS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, pages 297–311, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-12032-9.

Abhishek Anand and Vincent Rahli. Towards a formally verified proof assistant. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving: 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 27–44, Cham, 2014. Springer International Publishing. ISBN 978-3-319-08970-6.

Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A very modal model of a modern, major, general type system. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, pages 109–122, New York, NY, USA, 2007. ACM. ISBN 1-59593-575-4.

Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 197–208, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2326-0.

P. Bahr, H. B. Grathwohl, and R. E. Møgelberg. The clocks are ticking: No more delays! In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2017.

Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Stovring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. In *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*, LICS '11, pages 55–64, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4412-0.

Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. Guarded Cubical Type Theory: Path Equality for Guarded Recursion.

In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-022-4.

Aleš Bizjak, Lars Birkedal, and Marino Miculan. A model of countable nondeterminism in guarded type theory. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi*, pages 108–123, Cham, 2014. Springer International Publishing. ISBN 978-3-319-08918-8.

Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus E. Møgelberg, and Lars Birkedal. Guarded dependent type theory with coinductive types. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures: 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*, pages 20–35, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-49630-5.

Aleš Bizjak and Rasmus Ejlers Møgelberg. A model of guarded recursion with clock synchronisation. *Electron. Notes Theor. Comput. Sci.*, 319(C):83–101, December 2015. ISSN 1571-0661.

Aleš Bizjak and Rasmus Ejlers Møgelberg. Denotational semantics for guarded dependent type theory. Draft, 2017.

Edwin Brady. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, 23(5):552–593, Sep 2013.

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

Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal Dependent Type Theory and Dependent Right Adjoints. https://arxiv.org/abs/1804.05236, 2018.

R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986. ISBN 0-13-451832-2.

Karl Crary. *Type–Theoretic Methodology for Practical Programming Languages*. PhD thesis, Cornell University, Ithaca, NY, August 1998.

R.L. Crole. *Categories for Types*. Cambridge Mathematical Textbooks. Cambridge University Press, New York, 1993. ISBN 9780521457019.

Brian A. Davey and Hilary A. Priestley. *Introduction to lattices and order*. Cambridge University Press, Cambridge, 1990.

Murdoch J. Gabbay and Martin Hofmann. Nominal renaming sets. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR '08, pages 158–173, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89438-4.

Douglas J. Howe. Equality in lazy computation systems. In *Proceedings of Fourth IEEE Symposium on Logic in Computer Science*, pages 198–203, New York, 1989. IEEE Computer Society.

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 637–650, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3300-9.

Saunders Mac Lane and Ieke Moerdijk. *Sheaves in geometry and logic : a first introduction to topos theory*. Universitext. Springer, New York, 1992. ISBN 0-387-97710-4.

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

Ieke Moerdijk and Erik Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104(1):189 – 218, 2000. ISSN 0168-0072.

H. Nakano. A modality for recursion. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*, pages 255–266, New York, 2000. IEEE Computer Society.

Ulf Norell. Dependently typed programming in agda. In *Proceedings of the 4th International Workshop on Types in Language Design and Implementation*, TLDI '09, pages 1–2, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-420-1.

Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. A model of PCF in Guarded Type Theory. *Electronic Notes in Theoretical Computer Science*, 319 (Supplement C):333 – 349, 2015. ISSN 1571-0661. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

Vincent Rahli, Mark Bickford, and Robert Constable. Bar induction: The good, the bad, and the ugly. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2017.

Sam Staton. Name-passing process calculi: operational models and structural operational semantics. Technical Report UCAM-CL-TR-688, University of Cambridge, Computer Laboratory, June 2007.

Jonathan Sterling and Robert Harper. `coq-guarded-type-theory`. `https://github.com/jonsterling/coq-guarded-type-theory`, 2018.

The Coq Development Team. The Coq Proof Assistant Reference Manual, 2016.

Andrea Vezzosi. *Guarded Recursive Types in Type Theory*. Institutionen för data- och informationsteknik, Datavetenskap (Chalmers), Chalmers tekniska högskola, 2015. 63.

Noam Zeilberger. *The logical basis of evaluation order and pattern-matching*. PhD thesis, Carnegie Mellon University, 2009.