# A stratified approach to Löb induction

**Daniel Gratzer**
Aarhus University

**Lars Birkedal**
Aarhus University

## Abstract

Guarded type theory extends type theory with a handful of modalities and constants to encode productive recursion. While these theories have seen widespread use, the metatheory of guarded type theories, particularly guarded *dependent* type theories remains underdeveloped. We show that integrating Löb induction is the key obstruction to unifying guarded recursion and dependence in a well-behaved type theory and prove a no-go theorem sharply bounding such type theories.

Based on these results, we introduce GTT: a stratified guarded type theory. GTT is properly two type theories, sGTT and dGTT. The former contains only propositional rules governing Löb induction but enjoys decidable type-checking while the latter extends the former with definitional equalities. Accordingly, dGTT does not have decidable type-checking. We prove, however, a novel *guarded canonicity* theorem for dGTT, showing that programs in dGTT can be run. These two type theories work in concert, with users writing programs in sGTT and running them in dGTT.

**2012 ACM Subject Classification** Theory of computation → Type theory; Theory of computation → Denotational semantics; Theory of computation → Modal and temporal logics

**Keywords and phrases** Dependent type theory, guarded recursion, modal type theory, canonicity, categorical gluing

**Digital Object Identifier** 10.4230/LIPIcs...

## 1 Introduction

It is well-known that a fixed-point combinator $\mathtt{fix} : (A \to A) \to A$ wreaks havoc within type theory by rendering the theory unsound and making type-checking undecidable. Instead of a unified concept of recursion therefore, type theory is extended with sound induction and coinduction principles with implementations using syntactic checks to ensure that recursively-defined functions can be elaborated into these tamer principles.

Unfortunately, these syntactic checks are brittle, often forcing an artificial restructuring of a program. In order to obtain a type-theoretic account of recursion, [20] introduced *guarded recursion*. Instead of ensuring *a posteriori* that recursive calls occur after progress has been made, guarded recursion changes the type of recursive calls to prevent non-productive use.

Concretely, guarded type theory extends type theory with a modality—a unary type constructor—$\blacktriangleright A$ (pronounced "later $A$"). This modality is cartesian and comes with a point $\mathsf{next} : A \to \blacktriangleright A$. It does not, however, have an operator $\blacktriangleright A \to A$; once data enters the $\blacktriangleright$ modality, it cannot be extracted again. The fixed-points are then restored by Löb induction:

$$\mathsf{lob} : (\blacktriangleright A \to A) \to A \qquad\qquad \mathsf{lob}(f) = f(\mathsf{next}(\mathsf{lob}(f)))$$

While Löb induction was introduced in the context of provability, the $\blacktriangleright$ modality has a temporal intuition: $\blacktriangleright A$ contains elements of $A$ available one step in the future. As it stands, however, lob prevents non-productive usage a little too well and there are no types with non-constant maps $\blacktriangleright A \to A$ which renders lob induction useless. In order to rectify this, guarded type theories introduce variants of common types instrumented with $\blacktriangleright$s. The perennial example of this is the guarded stream type:

$$\mathtt{GStr}_A = A \times \blacktriangleright \mathtt{GStr}_A \tag{1}$$

© Daniel Gratzer and Lars Birkedal;
licensed under Creative Commons License CC-BY 4.0

After isolating the tail of the stream under $\blacktriangleright$, stream processors are defined using Löb:

$$\mathtt{map}(f) = \mathsf{lob}(\lambda r.\ \lambda s.\ (f(\mathsf{pr}_1(s)), r \circledast \mathsf{pr}_2(s)))$$

Here $(\circledast) : \blacktriangleright(A \to B) \to \blacktriangleright A \to \blacktriangleright B$ is an operator which witnesses the fact that $\blacktriangleright$ is cartesian. A non-productive stream processor such as `filter` cannot be defined: any attempt to define `filter` leaves us with $\blacktriangleright \mathtt{Str}_A$ in a place where we require $\mathtt{Str}_A$.

On top of this substrate, many variants of guarded type theory have been proposed [1, 2, 4, 5, 7, 13, 19, 22]. These extend the theory with guarded recursive types, more intricate combinations of modalities, or more flexible notions of time. In particular, extending the theory with an idempotent comonad $\Box$ such that $\Box \blacktriangleright A \simeq \Box A$ allows guarded type theories to encode coinductive types through guarded recursive types [7].

Unfortunately, attempts to extend guarded type theory to dependent types have been met with mixed success. On one hand they escape the inconsistencies of an unguarded fixed-point combinator, but on the other this alone does not make type-checking decidable.

## 1.1 Modal dependent type theory

In prior work [4, 5, 19], the precise cause of this undecidability issue was muddied. Type theorists had only just begun to understand modalities in dependent type theory, and the inclusion of any modality was apt to break properties such as decidability of type-checking. Independent of guarded type theories, modal type theory have been the subject of intense focus and recently [11] has proposed MTT, a calculus which enjoys normalization, canonicity, and decidable type-checking [10] and which can support $\blacktriangleright$ and $\Box$ (but not lob).

MTT is a *Fitch-style* modal type theory [3] parameterized by a 2-category of modes, modalities, and coercions between them. In this theory, modalities (written $\langle \mu \mid A \rangle$) are equipped with an 'adjoint' action on contexts $\Gamma.\{\mu\}$. The introduction rules for modalities are given by tranposition along the 'adjunction' $-.\{\mu\} \dashv \langle \mu \mid - \rangle$:

$$\frac{\Gamma.\{\mu\} \vdash M : A}{\Gamma \vdash \mathsf{mod}_\mu(M) : \langle \mu \mid A \rangle}$$

MTT can be instantiated with two modalities $\ell$ and $e$ along with equations forcing $\ell$ to model $\blacktriangleright$ while $\langle e \mid - \rangle$ becomes its left adjoint. The left adjoint to $\blacktriangleright$, written $\blacktriangleleft$ by [4], can be used to *encode* the more familiar $\Box$ modality as the limit of $\blacktriangleleft$, $\blacktriangleleft^2$, etc., as in $\mathbf{PSh}(\omega)$.

Even with these advances, however, a comprehensive guarded dependent type theory has remained out of reach. Adding Löb induction directly to MTT disrupts canonicity and normalization, and other proposed guarded type theories either fail to support crucial reasoning principles [2] or lack a proof of canonicity or normalization [17].

In fact, Löb induction has proven to be a far more serious obstacle to a well-behaved guarded type theory than the modal apparatus. We will prove a 'no-go' result that shows that including a definitional equality allowing Löb induction to unfold nearly always results in undecidable type-checking. This effectively rules out a theory which naïvely includes lob.

## 1.2 Stratifying guarded type theory

Given the complications of including Löb induction into guarded type theory, we propose a novel approach to the problem with a type theory GTT stratified into two layers: static and dynamic. Both of the two layers of the theory exhibit one of the two key properties of a type theory (normalization and canonicity), but neither satisfies both.

The static layer of our system does not include any definitional equalities for Löb induction and only supplies the user with propositional equalities. This layer includes not only $\blacktriangleright$ and $\mathsf{lob}$, but an $\blacktriangleleft$ modality which enables the user to encode coinductive types. Traditionally, the inclusion of $\blacktriangleleft$ and $\blacktriangleright$ satisfying the correct properties has been a significant source of difficulty for type theorists, but the aforementioned advances in modal type theory ensure that this static system enjoys decidable type-checking. Moreover, even without definitional equalities for Löb induction we have shown the static layer to be highly usable.

By including these axioms without the corresponding definitional equalities, however, the static layer does not validate canonicity; there are closed terms of type $\mathsf{nat}$ which are not convertible to numerals. Our 'no-go' theorem implies that rectifying this problem would require a radical restructuring of $\mathsf{GTT}$. Accordingly, we introduce a second 'dynamic' layer to our theory. The dynamic system is an extension of the static theory by definitional equalities making $\mathsf{lob}$ compute and forcing the propositional equality governing Löb induction to collapse to reflexivity. The dynamic theory does not enjoy decidable type-checking, but the additional equalities cause previously stuck terms to compute which we have proven via a novel 'guarded' canonicity theorem.

In total, a user may write a term in the static theory and type-check it as expected and then extract it to the dynamic theory for computation. By a careful analysis of the models of these theories, we show that computation can be used as a valid reasoning principle when viewing the type theory as the internal language of a model such as the topos of trees.

## 1.3 Guarded canonicity

Typically, canonicity states that a closed term in type theory is convertible with a canonical form. In guarded type theory, this is complicated by two factors.

The first is the presence of modal types like $\blacktriangleright A$: what are the canonical forms of $\blacktriangleright A$? The answer depends on the precise formulation of modalities, but existing solutions from $\mathsf{MTT}$ [10] apply. Specifically, we prove a canonicity result not just for terms in context $\mathbf{1}$, but in contexts $\mathbf{1}.\{\mu\}$ for all $\mu$. This generalization makes it easy to state the canonical forms of $\langle \mu \mid A \rangle$ in context $\mathbf{1}.\{\nu\}$: they are terms $\mathsf{mod}_\mu(M)$ such that $M$ is canonical in $\mathbf{1}.\{\nu \circ \mu\}$.

More seriously, in the presence of Löb induction canonical elements of some types must be infinite. In order to tame this non-termination, we adopt an idea common in guarded recursion and allow only finite approximations of infinite canonical forms [13]. $\mathsf{dGTT}$ is extended with a special context $\mathbf{0}$ under which all judgments collapse. By indexing $\mathbf{0}$ with a modality and ensuring $\mathbf{0}[\mu].\{\mu\} \cong \mathbf{0}$, we can prove canonicity only to a certain 'depth', without counting steps or otherwise imposing any rewriting system on our type theory. We prove guarded canonicity through an adaptation of *multimodal synthetic Tait computability* [10, 27].

## 1.4 Contributions

We prove the first 'no-go' theorem bounding the possibilities for including Löb induction in a type theory with decidable type-checking. As an answer to this result, we contribute $\mathsf{GTT}$, a guarded type theory stratified into static and dynamic layers. We show that the static layer has decidable type-checking and that the dynamic layer satisfies a novel form of canonicity.

In Section 2 we motivate and prove the no-go theorem for Löb induction. In Section 3 we introduce $\mathsf{GTT}$ and prove that the static layer enjoys decidable type-checking. In Section 4 we state and prove the first guarded canonicity result for a guarded type theory. Finally, in Section 5 we show that even without certain definitional equalities the static layer is highly usable and formalize a model of idealized synchronous programming within $\mathsf{sGTT}$.

## 2 A no-go theorem for Löb induction

Consider the essential ingredients of a guarded type theory specified above: Martin-Löf type theory equipped with a pointed cartesian modality $(\blacktriangleright, \mathsf{next}, \circledast)$ and a constant $\mathsf{lob} : (\blacktriangleright A \to A) \to A$ satisfying the following definitional equality:

$$\mathsf{lob}(f) = f(\mathsf{next}(\mathsf{lob}(f)))$$

We show that under two minor additional assumptions, definitional equality—and therefore type-checking—is undecidable in this system. Our first assumption is the existence of a non-trivial guarded type. Specifically, a type $S$ equipped with a definitional isomorphism:

$$\mathtt{cons} : \mathsf{S} \cong \mathsf{nat} \times \blacktriangleright \mathsf{S} \tag{2}$$

In the presence of universes and a suitable *dependent* version of later $\hat{\blacktriangleright} : \blacktriangleright \mathsf{U} \to \mathsf{U}$, we can define $S = \mathsf{lob}(x.\ \mathsf{nat} \times \hat{\blacktriangleright} x)$. In order to make this counter-example as broad as possible, however, we have chosen to explicitly isolate $S$.

The second assumption is that $\mathsf{next}$ must be suitably injective with respect to definitional equality. In particular, $\mathbf{1} \vdash \mathsf{next}(M) = \mathsf{next}(N) : \blacktriangleright A$ must be equivalent to $\mathbf{1} \vdash M = N : A$. This second requirement is more onerous than the first, and it does not follow directly from the existence of a familiar structure like universes. It does hold, however, in the standard model of guarded recursion $\mathbf{PSh}(\omega)$ [4]. Additionally, in a guarded type theory extended with $\Box$ or $\blacktriangleleft$, this bi-implication is derivable. In fact, while initially surprising, this second assumption is satisfied by all guarded type theories in the literature.

With these requirements, we show that deciding conversion would entail deciding the extensional equality of functions $\mathsf{nat} \to \mathsf{nat}$.

▶ **Lemma 1.** *There exists a function* $\mathtt{tabulate} : (\mathsf{nat} \to \mathsf{nat}) \to \mathsf{S}$ *such that the nth element of* $\mathtt{tabulate}(f)$ *is* $\mathsf{next}^n(f(\bar{n})) : \blacktriangleright^n \mathsf{nat}$.

**Proof.** We define $\mathtt{tabulate}$ by means of a helper function using Löb induction:

$$\mathsf{go} : (\mathsf{nat} \to \mathsf{nat}) \to \mathsf{nat} \to \mathsf{S}$$
$$\mathsf{go}(f) = \mathsf{lob}(\lambda r, n.\ \mathtt{cons}(f(n), r \circledast \mathsf{next}(n+1)))$$
$$\mathtt{tabulate}(f) = \mathsf{go}(f, 0)$$

◀

▶ **Theorem 2.** *Given two closed functions* $f, g : \mathsf{nat} \to \mathsf{nat}$*, the streams* $\mathtt{tabulate}(f)$ *and* $\mathtt{tabulate}(g)$ *are convertible if and only if* $f(\bar{n}) = g(\bar{n})$ *for all* $n : \mathbb{N}$.

**Proof.** The definitional isomorphism $\mathtt{cons}$ together with the $\eta$ principle for dependent sums ensure that it is both necessary and sufficient to show that the two streams are pointwise equal, but Lemma 1 together with the injectivity of $\mathsf{next}$ on closed terms reduces this condition to the extensional equality of $f$ and $g$. ◀

▶ **Corollary 3.** *Conversion is undecidable in a guarded type theory satisfying the following requirements:*

- *there is a type* $S$ *equipped with a definitional isomorphism* $S \cong \mathsf{nat} \times \blacktriangleright S$.
- $\mathsf{next} : A \to \blacktriangleright A$ *is injective on closed terms,*
- *the* $\mathsf{lob}$ *operator unfolds.*

$$\ell : m \longrightarrow m \qquad\qquad e : m \longrightarrow m$$

$$\mathsf{id} = e \circ \ell \qquad\qquad \mathsf{id} \leq \ell \circ e \qquad\qquad \mathsf{id} \leq \ell$$

**Figure 1** $\mathcal{M}$: a mode theory for guarded recursion.

## 3 GTT: a stratified guarded type theory

Corollary 3 places firm bounds on how much a guarded type theory can hope to achieve while maintaining decidable type-checking. Including an unconditional unfolding rule for Löb induction makes this goal nearly untenable, but without the ability to unfold lob there is no hope for any canonicity or computational adequacy result. While some guarded type theories have approached this problem by attempting to restrict the behavior of Löb induction to compute only in certain situations [2, 17], GTT takes a more radical approach.

Instead of considering a single theory which is carefully crafted to enjoy both computational adequacy and decidable type-checking simultaneously, GTT is split into a pair of type theories: sGTT and dGTT. The first enjoys decidable type-checking and the second satisfies (guarded) canonicity, but neither type theory has both properties.

Both type theories are extensions of MTT [11] with a mode theory $\mathcal{M}$ for guarded recursion detailed in Figure 1. For compatibility with MTT, this mode theory is presented as a 2-category, but contains only one object and is poset-enriched, so that all 2-cells are uniquely determined by their boundaries. In fact, $\mathcal{M}$ could equivalently be described as a monoidal partial order with two generating modalities $\ell$ and $e$. The inequalities governing these modalities force $\ell$ (respectively $e$) to behave like $\blacktriangleright$ (resp. $\blacktriangleleft$) on $\mathbf{PSh}(\omega)$.

### 3.1 Posetal MTT

Given that both sGTT and dGTT are based around this instantiation of MTT, we briefly review some of the key rules of MTT specialized to this mode theory. We refer the reader to [11] for a more comprehensive introduction to MTT. As a Fitch-style type theory, each modality in MTT behaves like an adjoint—in fact, a dependent right adjoint [3]—with the left adjoint acting on contexts. For a given modality $\mu$, we write the action of this left adjoint $\Gamma.\{\mu\}$ and, following [3, 12], we refer to $-.\{\mu\}$ as a "lock". In fact, this action extends to a 2-functor from $\mathcal{M}^{\mathsf{coop}}$ to the category of contexts and substitutions,[1] complete with a functorial action $\gamma \mapsto \gamma.\{\mu\}$ on substitutions and definitional equalities $\Gamma.\{\nu \circ \mu\} = \Gamma.\{\nu\}.\{\mu\}$. Moreover, an inequality of modalities $\mu \leq \nu$ induces a natural transformation $-.\{\nu\} \longrightarrow -.\{\mu\}$.

▶ Remark 4. While formally we follow [10] and work with MTT, sGTT, and dGTT as generalized algebraic theories—complete with explicit substitutions, de Bruijn indices, etc.—for exposition's sake with work with a more familiar informal syntax in examples.

The introduction rule for the modality $\langle \mu \mid - \rangle$ is then inspired by transposition:

$$\frac{\Gamma.\{\mu\} \vdash M : A}{\Gamma \vdash \mathsf{mod}_\mu(M) : \langle \mu \mid A \rangle}$$

Here we already have taken advantage of the particulars of $\mathcal{M}$ to simplify the rules of MTT; in general, a mode theory might contain multiple modes (multiple objects in the

[1] $\mathcal{M}^{\mathsf{coop}}$ is a 2-category with the same objects as $\mathcal{M}$, but with the directions of 1 and 2-cells reversed.

2-category) which would require us to annotate all the judgments of the type theory by a mode. Given that we work with only one mode, however, we omit these annotations.

The elimination rule is more complex; adding the inverse direction of transposition would disrupt the substitution properties of the type theory. Instead, each entry in the context is annotated by a modality $\Gamma, x : (\mu \mid A)$. The rule for accessing variables is then tweaked to factor in both the annotation on a variable and the locks following it in the context:

$$\frac{\mu \leq \mathsf{mods}(\Gamma_1)}{\Gamma_0, x : (\mu \mid A), \Gamma_1 \vdash x : A}$$

Here $\mathsf{mods}(\Gamma_1)$ is the composition of all modalities $\{\nu\}$ occurring in $\Gamma_1$. Unlike full MTT, variables are not annotated by 2-cells. As $\mathcal{M}$ is merely enriched over posets, at most one 2-cell can exist for any given pair of modalities. Accordingly, we remove these annotations on variables and replace them with the premise $\mu \leq \nu$ in the variable rule. In fact, given an inequality $\mu \leq \nu$ we will silently coerce from terms and types in context $\Gamma.\{\mu\}$ to terms and types in context $\Gamma.\{\nu\}$. No ambiguity can arise from this because $\mathcal{M}$ is poset-enriched.

Dependent products are in turn generalized to allow for abstraction over elements of $A$ annotated with a modality other than $\mathsf{id}$:

$$\frac{\Gamma, x : (\mu \mid A) \vdash M : B}{\Gamma \vdash \lambda x.\ M : (\mu \mid x : A) \to B}$$

Finally, the addition of annotations for variables in the context introduces some redundancy in the system: what is the relationship between a variable of type $\langle \mu \mid A \rangle$ annotated with $\nu$ and a variable of type $A$ annotated with $\nu \circ \mu$. The role of the elimination rule for $\langle \mu \mid - \rangle$ is to address this mismatch and patch the difference between $\Gamma.(\nu \mid \langle \mu \mid A \rangle)$ and $\Gamma.(\nu \circ \mu \mid A)$:

$$\frac{\begin{array}{c}\Gamma.\{\nu \circ \mu\} \vdash A \qquad \Gamma, x : (\nu \mid \langle \mu \mid A \rangle) \vdash B \\ \Gamma.\{\nu\} \vdash M_0 : \langle \mu \mid A \rangle \qquad \Gamma, x : (\nu \circ \mu \mid A) \vdash M_1 : B[\mathsf{mod}_\mu(x)/x]\end{array}}{\Gamma \vdash \mathsf{let}_\nu\ \mathsf{mod}_\mu(x) \leftarrow M_0\ \mathsf{in}\ M_1 : B[M_0/x]}$$

These primitives combine to ensure that e.g. modal types become a "pseudofunctor" so that $\mathsf{comp}[\mu;\nu] : \langle \mu \circ \nu \mid A \rangle \simeq \langle \mu \mid \langle \nu \mid A \rangle \rangle$ and an inequality $\mu \leq \nu$ induces a coercion $\mathsf{coe}[\mu \leq \nu] : \langle \mu \mid A \rangle \to \langle \nu \mid A \rangle$. We introduce some shorthand for this instantiation of MTT:

$$\blacktriangleright A \triangleq \langle \ell \mid A \rangle \qquad\qquad \blacktriangleleft A \triangleq \langle e \mid A \rangle$$

The pseudofunctorial structure of modalities quickly yields the standard operations of a guarded type theory e.g. $\mathsf{now} : \blacktriangleleft \blacktriangleright A \simeq A$ and $\mathsf{next} : A \to \blacktriangleright A$. For instance,

$$\mathsf{next}_A = \lambda x.\ \mathsf{mod}_\ell(x)$$

## 3.2 Crisp identity induction principles

In MTT, the elimination principles for types like $\mathsf{Id}_A(M, N)$ or $\mathsf{bool}$ are *mode-local* and therefore can only be applied to a variable if it is annotated with $\mathsf{id}$. This restriction is vital for soundness; the intended model of GTT does not validate $\langle \ell \mid \mathsf{bool} \rangle \simeq \mathsf{bool}$ and allowing $\mathsf{if}\ x\ \mathsf{then}\ M_0\ \mathsf{else}\ M_1$ when $x$ is under an $\ell$ annotation is equivalent to such an identification.

The intended model does, however, validate such an equivalence for the identity type. Intuitively, the typical models of GTT are extensional and accordingly $\mathsf{Id}_A(M, N)$ is realized by an equalizer (a limit) which is therefore preserved by all modalities (right adjoints):

$$\mathsf{Id}_{\langle \mu \mid A \rangle}(\mathsf{mod}_\mu(M), \mathsf{mod}_\mu(N)) \simeq \langle \mu \mid \mathsf{Id}_A(M, N) \rangle \tag{3}$$

As it turns out, this equivalence is crucial in practice. While we will see specific examples in Section 5, informally any proof of equality between programs defined via Löb induction requires Equation (3). In order to prove e.g. that two stream processors are equal, we must show that they produce streams which have identical heads and tails. Löb induction gives a proof under a later that the two tails are equal, but without Equation (3) we cannot parlay this into an equality between next applied to the two tails.

In prior guarded variants of MTT, this was circumvented by adopting equality reflection, but GTT takes a more refined approach and adds a stronger variant of the elimination principle for identity types inspired by Shulman's crisp induction principles [25]:

$$\frac{\begin{array}{c}\Gamma, a_0 : (\mu \mid A), a_1 : (\mu \mid A), p : (\mu \mid \mathsf{Id}_A(a_0, a_1)) \vdash B \\ \Gamma, x : (\mu \mid A) \vdash M : B[x, x, \mathsf{refl}(a)/a_0, a_1, p] \\ \Gamma.\{\mu\} \vdash N_0, N_1 : A \qquad \Gamma.\{\mu\} \vdash P : \mathsf{Id}_A(N_0, N_1)\end{array}}{\Gamma \vdash \mathsf{J}^\mu(B, a.M, P) : B[N_0, N_1, P/a_0, a_1, p]} \qquad \mathsf{J}^\mu(B, x.M, \mathsf{refl}(N)) = M[N/x]$$

Importantly, [10] has shown that extending MTT with these rules preserves both canonicity and normalization. This, together with the fact that all intended models of GTT validate the equivalent Equation (3), ensures that adding these rules to guarded MTT has no downside.

## 3.3 sGTT: The static fragment of GTT

The 'static' fragment of the guarded calculus adds Löb induction to guarded MTT. It enjoys decidable type-checking but escapes Corollary 3 by not including any definitional equalities for the lob operator. Concretely, sGTT is the instantiation of MTT described above supplemented with the following pair of axioms:

$$\frac{\Gamma, x : (\ell \mid A) \vdash M : A}{\Gamma \vdash \mathsf{lob}(x.M) : A} \qquad \frac{\Gamma, x : (\ell \mid A) \vdash M : A}{\Gamma \vdash \mathsf{unfold}(M) : \mathsf{Id}_A(\mathsf{lob}(x.M), M[\mathsf{lob}(x.M)/x])}$$

A term in sGTT is then precisely a term in MTT in a context extended by these two constants, so we may immediately deduce the following results from [10]:

▶ **Theorem 5.** *Conversion and type-checking in sGTT are both decidable sGTT.*

**Proof.** Corollaries 8.8 and 8.10 of [10] show that conversion and type-checking are decidable if the equality of 1- and 2-cells $\mathcal{M}$ enjoy decidable conversion. This problem is easily seen to be equivalent to the decidability of the $\leq$ relation presented in Figure 1. We first observe that every 1-cell can be uniquely presented as $l^n \circ e^m$ for some pair $(n, m)$. Next, because $l^{n_0} \circ e^{m_0} \leq l^{n_1} \circ e^{m_1}$ if and only if $n_0 \leq n_1$ and $m_1 - n_1 \leq m_0 - n_0$, the equality of 1-cells and existence of 2-cells are both decidable as required. ◀

## 3.4 dGTT: the dynamic fragment of GTT

The dynamic half of GTT is an extension of sGTT, supplementing the static theory with a pair of definitional equalities:

$$\frac{\Gamma, x : (\ell \mid A) \vdash M : A}{\begin{array}{c}\Gamma \vdash \mathsf{lob}(x.M) = M[\mathsf{lob}(x.M)/x] : A \\ \Gamma \vdash \mathsf{unfold}(x.M) = \mathsf{refl}(\mathsf{lob}(x.M)) : \mathsf{Id}_A(\mathsf{lob}(M), M[\mathsf{lob}(x.M)/x])\end{array}}$$

Notice that Corollary 3 applies to dGTT and so type-checking dGTT is undecidable.

We also equip dGTT with a new form of context $\mathbf{0}[\mu]$, which will prove crucial to understanding its computational behavior. The context $\mathbf{0} = \mathbf{0}[\mathsf{id}]$ acts like an initial object in the category of contexts so that all judgments $\mathbf{0} \vdash \mathcal{J}$ hold, and the more general $\mathbf{0}[\mu]$ intuitively represents the application of the modality $\mu$ to this initial object. The rules for these contexts are presented in Figure 2, but we postpone further discussion of $\mathbf{0}$ until Section 4 where its role can be fully motivated.

Henceforth, it will become important to distinguish between which judgments hold in sGTT and which hold in dGTT. We write $\Gamma \vdash_{\mathbf{s}} M : A$ if $M : A$ is derivable with only the rules of sGTT and $\Gamma \vdash_{\mathbf{d}} M : A$ for the corresponding notion for dGTT. As dGTT is a strict extension of sGTT, the following is immediate:

▶ **Theorem 6.** *If a judgment is derivable in sGTT, the same judgment holds in dGTT.*

## 3.5 The semantics of GTT

A priori, of course, Theorem 6 could hold even if dGTT included all manner of extensions to sGTT. The fact that it only extends sGTT with two additional equations means that all models of dGTT are models of sGTT and all 'natural' semantic models of sGTT are also model of dGTT. Concretely, the category of models of sGTT is precisely the category of models of MTT [11] with mode theory $\mathcal{M}$ together with constants for lob and unfold. Unfolding these definitions, a model of sGTT is a category with families [8] equipped with functors interpreting $-.\{\mu\}$ and structure for interpreting modal types.

A model of dGTT is a model of sGTT satisfying a pair of additional equalities and with structure interpreting $\mathbf{0}[\mu]$. In practice these additional requirements are easily satisfied:

▶ **Theorem 7.** *A model of sGTT satisfying the following extends to a model of dGTT:*

1. *the category of contexts and substitutions has an initial object*
2. *the interpretation of* Id *validates equality reflection*
3. *the model is democractic; every context can be realized as a type*
4. *the modalities are realized by dependent right adjoints [3, Definition 2]*

In particular, the standard models of guarded recursion in sheaves over complete Heyting algebras with a well-founded basis such as $\mathbf{PSh}(\omega)$ are models of both sGTT and dGTT [4]. See Appendix A for a full definition of models of sGTT and dGTT.

# 4 Guarded canonicity

We have shown that sGTT enjoys decidable type-checking. In this section we establish that the twin theory dGTT is computationally effective, i.e. that programs in this theory can run. Typically, a canonicity theorem for type theory states that any closed boolean is convertible with either true or false. For dGTT, this result would be unsatisfactory because it would not give information about terms of ▶ bool and therefore could not shed light on executing guarded programs. We therefore characterize canonical forms in a class of contexts $\mathcal{C}$ other than the empty context. By ensuring that $\mathcal{C}$ is closed under $-.\{\mu\}$, canonicity applies to programs executing under a ▶, but this raises a further complication in the presence of Löb induction: programs like `tabulate`(id) (Lemma 1) with infinite canonical forms. Rather than allowing the canonical forms themselves to be infinite, we use a novel type-directed form of "fuel" in our statement of canonicity.

Prior to formulating this more refined notion, recall from Section 3.4 that dGTT has a distinguished context $\mathbf{0}$ which is initial in the category of contexts and substitutions (Figure 2).

$$\frac{}{\mathbf{0}\ \mathsf{cx}} \qquad \frac{}{\mathbf{0}[\mu]\ \mathsf{cx}} \qquad \frac{}{\mathbf{0} \vdash \lightning : A} \qquad \frac{}{\mathbf{0} \vdash \lightning} \qquad \frac{}{\mathbf{0}.\{\mu\} \vdash \lightning : \Gamma}$$

$$\frac{\Gamma \vdash \gamma : \mathbf{0}}{\Gamma \vdash M = N : A \qquad \Gamma \vdash A = B \qquad \Gamma \vdash \delta_0 = \delta_1 : \Delta} \qquad \frac{\Gamma \vdash \gamma : \mathbf{0}[\mu]}{\Gamma.\{\mu\} \vdash \gamma^{\leftarrow} : \mathbf{0}} \qquad \frac{\Gamma.\{\mu\} \vdash \gamma : \mathbf{0}}{\Gamma \vdash \gamma^{\rightarrow} : \mathbf{0}[\mu]}$$

$$\gamma^{\leftarrow\rightarrow} = \gamma \qquad \gamma^{\rightarrow\leftarrow} = \gamma \qquad \gamma^{\leftarrow} \circ \delta = (\gamma \circ \delta.\{\mu\})^{\leftarrow}$$

**Figure 2** The vacuous context.

For each modality $\mu$, there exists an additional context $\mathbf{0}[\mu]$—intuitively this is the context $\mathbf{0}$ under the modality $\mu$—whose behavior is fully captured by a natural isomorphism of hom-sets in the category of contexts: $\hom(\Gamma, \mathbf{0}[\mu]) \cong \hom(\Gamma.\{\mu\}, \mathbf{0})$. With this additional structure, we then characterize the canonical forms in programs in context $\mathbf{0}[\mu].\{\nu\}$ so that $(\mu, \nu)$ serves as an abstracted form of fuel:

▶ **Theorem 19** (Guarded canonicity). *Given a term $\mathbf{0}[\mu].\{\nu\} \vdash_{\mathbf{d}} M : A$, the following holds*

- *If $A = \mathsf{nat}$ then there exist a numeral $k$ such that $M = \bar{k}$*
- *If $A = \langle \xi \mid B \rangle$ then $M = \mathsf{mod}_{\xi}(N)$ for some $\mathbf{0}[\mu].\{\nu \circ \xi\} \vdash_{\mathbf{d}} N : B$.*

We pause for a moment to consider the consequences of Theorem 19 for various types and pairs $(\mu, \nu)$. First, if $\mu = \nu$ then there is a canonical substitution $\mathbf{0}[\mu].\{\nu\} \vdash \mathsf{id}^{\leftarrow} : \mathbf{0}$ and canonicity trivializes—all types are inhabited and all terms are equal in such a context.

Suppose instead, therefore, we have a term $\mathbf{0}[\ell \circ \ell].\{\mathsf{id}\} \vdash_{\mathbf{d}} M : \langle \ell \mid \langle \ell \mid A \rangle \rangle$. Guarded canonicity then ensures that $M = \mathsf{mod}_{\ell}(M_0)$ for some $\mathbf{0}[\ell \circ \ell].\{\ell\} \vdash_{\mathbf{d}} M_0 : \langle \ell \mid A \rangle$. The theorem then applies to $M_0$, yielding $M_0 = \mathsf{mod}_{\ell}(M_1)$ for some $\mathbf{0}[\ell \circ \ell].\{\ell \circ \ell\} \vdash_{\mathbf{d}} M_1 : A$ but the process stops here; the context is now isomorphic to $\mathbf{0}$ and so $M_1 = \lightning$.

Informally, in a context $\mathbf{0}[\mu].\{\nu\}$ the modality $\mu$ represents the initial fuel given to evaluate a program, while $\nu$ signifies the fuel expended thus far in the process. The analogy is imperfect—the presence of the earlier modality allows for 'negative' fuel to be spent—but it does make it clear how one might use Theorem 19: to evaluate a closed program $M : \blacktriangleright^n \mathsf{nat}$, one begin by weakening the program into the context $\mathbf{0}[\ell^{n+1}]$ and then repeatedly apply guarded canonicity to deduce that $M = \mathsf{mod}_{\ell}(\mathsf{mod}_{\ell}(\ldots(\bar{k})))$ for some numeral $k$. Crisply, in order to extract a numeral from $\langle \mu \mid \mathsf{nat} \rangle$ one must evaluate it with more than $\mu$ fuel.

Unlike other canonicity results based on fuel [13], our guarded canonicity result does not depend on a fixed operational semantics or presentation of the equality judgment. The fuel is a purely type-directed notion and throughout the statement and proof of guarded canonicity we manipulate only equivalence classes of terms up to definitional equality.

## 4.1 Proving guarded canonicity

In order to prove Theorem 19, we use a modified variant of *multimodal synthetic Tait computability* [10, 27], a refinement of gluing proofs. Rather than fixing a rewriting system presenting the equational theory of dGTT, we construct a model of dGTT lying over the syntactic model comprised essentially of terms equipped with a guarded canonical form. The initiality of syntax then guarantees that every term has a guarded canonical form.

Several caveats complicate this simple picture. In the first instance, it is frequently difficult to construct a model of dGTT lying over the syntactic model due to the number of strict equations involved. We therefore follow the approach taken by [10]: we begin by

defining a relaxed category of *Löb cosmoi* and show that there is an appropriate *syntactic Löb cosmos* which enjoys a suitable weakening of initiality. From there, we construct a glued Löb cosmos which supports a rich model of MTT complete with several new primitives. We call the resultant language *multimodal synthetic Tait computability* (MSTC) and use this language to to construct the interpretation of types and terms *internally*. In fact, because the model construction takes place within MSTC, once this language is in place the construction of the actual model is *mutatis mutandis* the one given in [10].

For the sake of space, therefore, we trace through the aspects of the proof which differ significantly from the normalization proof for MTT. In particular, we show how we construct the glued cosmos necessary to prove guarded canonicity and gloss the interpretation of Löb induction within MSTC. The remaining details—such as the interpretation of all the connectives of MTT—are deferred to Appendices B and C.

▶ **Definition 8.** *A Löb cosmos is a strict 2-functor $G : \mathcal{M} \longrightarrow \mathbf{Cat}$ such that $G(m)$ (which we abusively write $G$) is a locally Cartesian closed category with an initial object and each $G(\mu)$ is a right adjoint. We require the following additional structure:*

1. *A morphism $\tau_G : \dot{\mathcal{T}}_G \longrightarrow \mathcal{T}_G$ in $G$ representing the universe of types and closed under all the connectives of MTT e.g. for each $\mu$ a map $G(\mu)(\mathcal{T}_G) \longrightarrow \mathcal{T}_G$ encoding the formation rule for modal types.*
2. *An element* lob *with the appropriate type and necessary equation.*

*A morphism of Löb cosmoi is a 2-natural transformation of LCC functors satisfying Beck-Chevalley which preserves all structure.*

While in [10], syntax could be organized into a cosmos enjoying a property akin to initiality by taking presheaves on the category of contexts and substitutions $\mathsf{Cx}$, the same maneuver is not available here: the vacuous context $\mathbf{0}$ would no longer be initial when embedded into $\mathbf{PSh}(\mathsf{Cx})$. We therefore localize at $\mathbf{0}_{\mathbf{PSh}(\mathsf{Cx})} \longrightarrow \mathbf{y}(\mathbf{0})$ and consider *sheaves* over the category of contexts. Explicitly, we equip $\mathsf{Cx}$ with a Grothendieck topology $J$:

$$J(\Gamma) = \begin{cases} \{\hom(-,\Gamma), \emptyset\} & \text{if there exists a substitution } \Gamma \longrightarrow \mathbf{0} \\ \{\hom(-,\Gamma)\} & \text{otherwise} \end{cases}$$

The rules in Figure 2 ensure that this Grothendieck topology is subcanonical and that the presheaves of types and terms are in fact sheaves for this topology. It is also easily seen that the precomposition functors induced by $-.\{\mu\}$ restrict to right adjoints on sheaves and that $\mathbf{Sh}(\mathsf{Cx})$ is closed under finite limits and dependent products in $\mathbf{PSh}(\mathsf{Cx})$.

▶ **Theorem 9.** *There is a Löb cosmos $\mathcal{S}(m) = \mathbf{Sh}(\mathsf{Cx})$ where $\mathcal{T}$ (respectively $\dot{\mathcal{T}}$) are realized by the sheaves of types (respectively terms) and $\mathcal{S}(\mu)$ is given by precomposition with $-.\{\mu\}$.*

Passing to sheaves enables us to recover the *quasi-projectivity* property described by [10]:

▶ **Theorem 10.** *Given a Löb cosmos $G$ and a map $\pi : G \longrightarrow \mathcal{S}$, the following holds:*

1. *For every context $\Gamma$ $\mathsf{cx}_{\mathbf{d}}$, there exists an object $[\![\Gamma]\!] : G$ and $\alpha_\Gamma : \pi([\![\Gamma]\!]) \cong \mathbf{y}(\Gamma)$.*
2. *For every type $\Gamma \vdash_{\mathbf{d}} A$, there is a morphism $[\![A]\!] : [\![\Gamma]\!] \longrightarrow \mathcal{T}_G$ such that $\pi([\![A]\!]) \circ \alpha_\Gamma = \lfloor A \rfloor$.*
3. *For every $\Gamma \vdash_{\mathbf{d}} M : A$, there exists $[\![M]\!] : [\![\Gamma]\!] \longrightarrow \dot{\mathcal{T}}_G$ over $[\![A]\!]$ such that $\pi([\![M]\!]) \circ \alpha_\Gamma = \lfloor M \rfloor$.*

*Here $\lfloor - \rfloor$ is half of the isomorphism induced by the Yoneda lemma.*

We can now revise our original proof strategy for guarded canonicity: we will construct a particular Löb cosmos and use Theorem 10 to derive the theorem. We now turn to constructing this Löb cosmos by gluing the syntactic cosmos along a functor to a Grothendieck topos.

As in all gluing proofs, the choice of functor to glue along is crucial. For instance, when proving a standard canonicity result one glues along $\mathbf{PSh}(\mathsf{Cx}) \longrightarrow \mathbf{Set} = \mathbf{PSh}(\mathbf{1})$ given by precomposition with $\mathbf{1} \longrightarrow \mathsf{Cx}$. For normalization, one wishes to work with arbitrary contexts but normal forms are stable under a limited class of *renamings*. Accordingly, one glues along $\mathbf{PSh}(\mathsf{Cx}) \longrightarrow \mathbf{PSh}(\mathsf{Ren})$ given by precomposing with the inclusion $\mathsf{Ren} \longrightarrow \mathsf{Cx}$.

In our case, because we wish to prove a result about terms in context $\mathbf{0}[\mu].\{\nu\}$ we will take a category spanned by contexts of this form. Moreover, because guarded canonical forms are stable under the natural transformations $\mathbf{0}[\mu].\{\nu_0\} \longrightarrow \mathbf{0}[\mu].\{\nu_1\}$, we can recast this subcategory $\mathsf{Cx}$ spanned by contexts $\mathbf{0}[\mu].\{\nu\}$ as a partial order:

▶ **Definition 11.** *Define $(P, \leq)$ to be a partial order whose elements are pairs of modalities $(\mu, \nu)$ such that $(\mu_0, \nu_0) \leq (\mu_1, \nu_1)$ if $\mu_0 = \mu_1$ and $\nu_1 \leq \nu_0$. There is a functor $i : P \longrightarrow \mathsf{Cx}$ sending $(\mu, \nu)$ to $\mathbf{0}[\mu].\{\nu\}$*

Unlike in prior gluing proofs, we represent syntax with $\mathbf{Sh}(\mathsf{Cx})$ rather than $\mathbf{PSh}(\mathsf{Cx})$. Accordingly, we must impose a Grothendieck topology on $P$ so that the inclusion $P \longrightarrow \mathsf{Cx}$ induces a functor $\mathbf{Sh}(\mathsf{Cx}) \longrightarrow \mathbf{Sh}(P)$ and it is this functor that we will glue along.

▶ **Definition 12.** *Transporting the Grothendieck topology on $\mathsf{Cx}$ along the functor $i : P \longrightarrow \mathsf{Cx}$ yields a new topology on $P$ covering $(\mu, \nu)$ with the empty family if $\exists \xi.\ \mu \circ \xi \leq \nu$.*

▶ **Lemma 13.** *Precomposition by $(\mu, \nu) \mapsto (\mu, \nu \circ \xi)$ induces a right adjoint $R_\xi : \mathbf{Sh}(P) \longrightarrow \mathbf{Sh}(P)$.*

▶ **Lemma 14.** *Recalling $\mathcal{S}(\xi) = (-.\{\xi\})^*$ from Theorem 9, $i^* \circ \mathcal{S}(\xi) = R_\xi \circ i^*$.*

Observe that for any pair $(\mu, \nu)$, there exists $n$ such that $\exists \xi.\ \mu \circ \xi \leq \nu \circ \ell^n$. Accordingly, given $X : \mathbf{Sh}(P)$ and $(\mu, \nu) : P$ there exists $n$ such that $R^n_\ell(X)(\mu, \nu) = \{\star\}$. This eventual trivialization ensures that $\mathbf{Sh}(P)$ satisfies Löb induction:

▶ **Lemma 15.** *For any $X : \mathbf{Sh}(P)$ there is a morphism $\mathsf{lob}_X : X^{R_\ell(X)} \longrightarrow X$ satisfying the unfolding equation for Löb induction.*

▶ **Lemma 16.** *Precomposition with $i$ induces a right adjoint $i^* : \mathcal{S} = \mathbf{Sh}(\mathsf{Cx}) \longrightarrow \mathbf{Sh}(P)$.*

Gluing along $i^*$, we obtain a Grothendieck topos $\mathcal{G} = \mathbf{Gl}(i^*)$ whose objects are triples $\big(X : \mathbf{Sh}(\mathsf{Cx}), Y : \mathbf{Sh}(P), f : Y \longrightarrow i^* X\big)$. Intuitively, these are proof-relevant predicates on syntax so that constructing a Löb cosmos in $\mathcal{G}$ is akin to a proof-relevant logical relation.

▶ **Lemma 17.** *There is a strict 2-functor $\mathcal{G} : \mathcal{M} \longrightarrow \mathbf{Cat}$ sending $\mathcal{G}(m) = \mathbf{Gl}(i^*)$ and $\mathcal{G}(\mu)$ is determined component-wise by $\mathcal{S}(\mu)$ and $R_\mu$. Furthermore, $\pi : \mathcal{G} \longrightarrow \mathcal{S}$ is a 2-natural transformation spanned by LCC functors and satisfying Beck-Chevalley.*

**Proof.** This follows from Theorem 4.13 of [10] and Lemma 13 ◀

It remains only to equip $\mathcal{G}$ with the structure of a Löb cosmos i.e. a universe $\tau_{\mathcal{G}} : \dot{\mathcal{T}}_{\mathcal{G}} \longrightarrow \mathcal{T}_{\mathcal{G}}$ closed under various connectives. In fact, this process is remarkably routine. $\mathcal{G}$ is a Grothendieck topos and therefore models extensional Martin-Löf type theory with a hierarchy of cumulative universes [24] satisfying the internal realignment principle formulated by [21].[2] Moreover, $\mathcal{G}$ is a model of extensional $\mathsf{MTT}$ with a pair of complementary idempotent monads $\bigcirc$ and $\bullet$ presenting $\mathbf{Sh}(\mathsf{Cx})$ (respectively $\mathbf{Sh}(P)$) as an open (resp. closed) subtopos.

[2] Unlike the well-known construction of universes in presheaf topoi set forth by [16], [24] gives a construction which applies in *arbitrary* arbitrary Grothendieck topoi using a small-object argument. This construction requires the axiom of choice and it is currently open whether a constructively acceptable analog exists.

This combination of modalities shapes $\mathcal{G}$ into a model of *multimodal synthetic Tait computability*. While there are minor differences in the precise properties of multimodal synthetic Tait computability, this interpretation ensures that we can virtually replay the construction of a universe closed under the connectives of MTT given by [10]. Given this essential similarity, we gloss only Löb induction: the last novelty of this construction.

Interpreting lob hinges on the fact that $\mathbf{Sh}(P)$ and $R_\ell$ satisfy Löb induction. Lifting Lemma 15 this into the internal language of $\mathcal{G}$ gives us a variant of Löb induction:

$$\textstyle\prod_{A:\mathsf{U}}(\langle \ell \mid \bullet A\rangle \to \bullet A) \to \bullet A$$

This limited constant is sufficient to construct the proper interpretation of Löb induction, which in turn yields the following cousin of the fundamental lemma of logical relations:

▶ **Theorem 18.** *There is a Löb cosmos in $\mathcal{G}$ such that $\pi : \mathcal{G} \longrightarrow \mathcal{S}$ is a map of Löb cosmoi.*

Finally, from Theorems 10 and 18 combined with the definition of terms in $\mathcal{G}$ we conclude:

▶ **Theorem 19** (Guarded canonicity)**.** *Given a term $\mathbf{0}[\mu].\{\nu\} \vdash_{\mathbf{d}} M : A$, the following holds*

- *If $A = \mathsf{nat}$ then there exist a numeral $k$ such that $M = \bar{k}$*
- *If $A = \langle \xi \mid B\rangle$ then $M = \mathsf{mod}_\xi(N)$ for some $\mathbf{0}[\mu].\{\nu \circ \xi\} \vdash_{\mathbf{d}} N : B$.*

## 5 Proving and programming with guarded streams

While Theorems 5 and 19 ensures that each half of GTT satisfies one of the two theorems typically used to evaluate usability of a type theory, neither satisfies both. It unclear, therefore, whether or not sGTT can really be used to write guarded programs. This is an empirical question, but we argue the affirmative by showing an encoding of synchronous programming [6] in sGTT and capitalize on Theorems 6 and 19 to compute with the results.

### 5.1 A warmup: defining guarded streams

To begin with, we define the type of guarded streams $\mathtt{Str} : \mathsf{U} \to \mathsf{U}$ in sGTT:

$$\begin{aligned} &\mathtt{Str} : \mathsf{U} \to \mathsf{U} && \iota : (A : \mathsf{U}) \to \mathsf{Id}_{\mathsf{U}}(\mathtt{Str}(A), A \times \langle \ell \mid \mathtt{Str}(A)\rangle) \\ &\mathtt{Str}(A) = \mathsf{lob}(S.\ A \times \langle \ell \mid S\rangle) && \iota = \mathsf{unfold}(S.\ A \times \langle \ell \mid S\rangle) \end{aligned}$$

We can define the usual operations on streams using this equivalence. For instance, cons (respectively head and tail) is given by transporting along $\iota^{-1}$ (resp. $\iota$). The functoriality of transports then yields an identification $\mathsf{Id}_{\mathtt{Str}(A)}(\mathtt{cons}(\mathtt{head}(s), \mathtt{tail}(s)), s)$.

Fix types $A, B : \mathsf{U}$ and a map $f : A \to B$, we extend this map to a map of streams:

$$\mathtt{map}(f) = \mathsf{lob}(g.\ \lambda s.\ \mathtt{cons}(f(\mathtt{head}(s)), \mathsf{mod}_\ell(g) \circledast \mathtt{tail}(s)))$$

▶ **Lemma 20.** *Given $s : \mathtt{Str}(A)$, there is an identification $\mathsf{Id}_{\mathtt{Str}(A)}(\mathtt{map}(\mathsf{id}, s), s)$.*

**Proof.** We prove this by Löb induction so we are given an induction hypothesis:

$$r : \left(\ell \;\middle|\; (s : \mathtt{Str}(A)) \to \mathsf{Id}_{\mathtt{Str}(A)}(\mathtt{map}(\mathsf{id}, s), s)\right)$$

Now fix $s : \mathtt{Str}(A)$. We have an identification between $s$ and $\mathtt{cons}(\mathtt{head}(s), \mathtt{tail}(s))$. The left-hand side is identified with $\mathtt{cons}(\mathsf{id}(\mathtt{head}(s)), \mathsf{next}(\mathtt{map}(\mathsf{id})) \circledast \mathtt{tail}(s))$ via $\mathsf{unfold}(\ldots)$. It then suffices to construct an element of $\mathsf{Id}_{\langle \ell \mid \mathtt{Str}(A)\rangle}(\mathsf{next}(\mathtt{map}(\mathsf{id})) \circledast \mathtt{tail}(s), \mathtt{tail}(s))$. To

this end, we use the elimination rule for modal types: Applying this rule to $\mathtt{tail}(s)$, it suffices to construct the following identification for an arbitrary $x : (\ell \mid \mathtt{Str}(A))$:

$$\mathsf{Id}_{\langle \ell \mid \mathtt{Str}(A) \rangle}(\mathsf{mod}_\ell(\mathtt{map}(\mathsf{id}, x)), \mathsf{mod}_\ell(x))$$

The result is now an immediate consequence of $r(x)$ combined with Equation (3). ◀

▶ **Lemma 21.** *Given functions $f : A \to B$, $g : B \to C$, for each $s : \mathtt{Str}(A)$ there is an identification $\mathsf{Id}_{\mathtt{Str}(C)}(\mathtt{map}(g \circ f, s), \mathtt{map}(g, \mathtt{map}(f, s)))$.*

The proofs of the above lemmas (see the one for Lemma 20), show how several features of sGTT are crucial for programming. In particular, we have made heavy use of the *propositional* unfolding rules for Löb induction to manipulate a definition using lob, and we rely on crisp identity induction principles to use Löb induction to prove an equality.

## 5.2 A logical foundation for synchronous programming

In order to test the limits of sGTT, we revisit the abstract encoding of synchronous programming [15] (as implemented by e.g. Lustre [14]) in dependent type theory with coinductive streams given by [6]. We repeat their construction with the guarded streams defined in Section 5.1. Intuitively, we interpret a type $\tau$ of their calculus as guarded stream of values in GTT. By using guarded streams, however, we are able to simplify the construction of this model and eliminate the need to manually ensure well-formedness of various constructions. Moreover, we show that propositional Löb is sufficient for guarded programming.

We first define a type of clocks or warps which dictate the rate at which a stream is expected to produce values. As in [6], these are realized by streams of booleans $W = \mathtt{Str}(\mathsf{bool})$. The primitive objects of our encoding are specialized streams indexed by a warp:

$$\mathtt{WStr}(A) = \mathsf{lob}(B.\ \lambda w.\ \mathsf{let}\ \mathsf{mod}_\mu(u) \leftarrow \mathtt{tail}(w)\ \mathsf{in}\ \mathsf{if}\ \mathtt{head}(w)\ \mathsf{then}\ A \times \langle \ell \mid B(u) \rangle\ \mathsf{else}\ \langle \ell \mid B(u) \rangle)$$

So a WStr is indexed by a type $A$ and warp $w$, and contains values of $A$ only when $w$ is true. This is similar to the model of Lustre detailed by [6], but in contrast to *loc. cit.*, which uses coinductive streams, our use of guarded streams facilitates a number of conceptual simplifications while immediately guaranteeing some theorems.

Henceforth we will stop explicitly using lob to define guarded recursive functions, and adopt the more typical informal style so that the above definition could be written

$$\mathtt{WStr}(A, \mathtt{cons}(b, \mathsf{mod}_\ell(w))) = \mathsf{if}\ b\ \mathsf{then}\ A \times \langle \ell \mid \mathtt{WStr}(A, w) \rangle\ \mathsf{else}\ \langle \ell \mid \mathtt{WStr}(A, w) \rangle$$

We also adopt pattern-matching syntax for modal types and equivalences like cons as well as a **rewrite** based on the available operation in Agda. We further avail ourselves of cons, head, and tail for WStr. These have slightly unusual types to account for the presence of warps and the potential absence of data at a given step.

$$\begin{aligned}
&\mathtt{cons} : (b : \mathsf{bool})(\ell \mid w : W) \\
&\qquad \to (\mathsf{if}\ b\ \mathsf{then}\ A \times \langle \ell \mid \mathtt{WStr}(A, w) \rangle\ \mathsf{else}\ \langle \ell \mid \mathtt{WStr}(A, w) \rangle) \to \mathtt{WStr}(A, \mathtt{cons}(b, \mathsf{mod}_\ell(w))) \\
&\mathtt{head} : (\ell \mid w : W) \to \mathtt{WStr}(A, \mathtt{cons}(\mathsf{tt}, \mathsf{mod}_\ell(w))) \to A \\
&\mathtt{tail} : (b : \mathsf{bool})(\ell \mid w : W) \to \mathtt{WStr}(A, \mathtt{cons}(b, \mathsf{mod}_\ell(w))) \to \langle \ell \mid \mathtt{WStr}(A, w) \rangle
\end{aligned}$$

We use this new surface syntax to define the constant stream of values:

$$\mathtt{const} : A \to (w : W) \to \mathtt{WStr}(A, w)$$

$$\texttt{const}(x, \texttt{cons}(b, \mathsf{mod}_\ell(w))) = \mathsf{if}\ b\ \mathsf{then}\ \texttt{cons}(x, \mathsf{mod}_\ell(\texttt{const}(x, w)))\ \mathsf{else}\ \texttt{cons}(\mathsf{mod}_\ell(\texttt{const}(x, w)))$$

Next, we define a combinator allowing us to zip two streams and combine their values:

$$\begin{aligned}
&\texttt{zip} : (w : W) \to \texttt{WStr}(A \to B, w) \to \texttt{WStr}(A, w) \to \texttt{WStr}(B)\\
&\texttt{zip}(\texttt{cons}(b, \mathsf{mod}_\ell(w)), s_1, s_2) =\\
&\quad \mathsf{if}\ b\\
&\quad \mathsf{then}\ \texttt{cons}(\texttt{head}(s_1)(\texttt{head}(s_2)), \mathsf{mod}_\ell(\texttt{zip}(w)) \circledast \texttt{tail}(s_1) \circledast \texttt{tail}(s_2))\\
&\quad \mathsf{else}\ \texttt{cons}(\mathsf{mod}_\ell(\texttt{zip}(w)) \circledast \texttt{tail}(s_1) \circledast \texttt{tail}(s_2))
\end{aligned}$$

While we omit the definitions, we also can define a variant of Lustre's `fby` operator, which delays a stream by one and replaces the first element:

$$\texttt{fby} : (w : W) \to A \to \texttt{WStr}(A, \texttt{cons}(\mathsf{ff}, \mathsf{mod}_\ell(w))) \to \texttt{WStr}(A, \texttt{cons}(\mathsf{tt}, \mathsf{mod}_\ell(w)))$$

In fact, we have provided definitions of all the combinators presented by [6], including several which further exercise the dependent types of sGTT.

## 5.3 Running an example program

Thus far we have carefully worked in sGTT, so our programs can be type-checked. Unfortunately, the lack of canonicity in sGTT means that they cannot be run. The split nature of GTT, however, guarantees equivalent programs in dGTT which can be executed. To concretize this discussion, consider the following the stream of natural numbers in sGTT:

$$\begin{aligned}
&\texttt{nats} : \texttt{WStr}(\mathsf{nat}, \mathsf{lob}(\texttt{cons}(\mathsf{tt}, \mathsf{mod}_\ell(-))))\\
&\texttt{nats}\ \textbf{rewrite}\ \mathsf{unfold}(\ldots) = \texttt{fby}(0, \texttt{zip}(\texttt{cons}(\mathsf{next}(\texttt{const}(\lambda n.\ \mathsf{suc}(n)))), \texttt{cons}(\mathsf{mod}_\ell(\texttt{nats}))))
\end{aligned}$$

While this program is relatively simple, we can already detect something unsatisfactory: it is hard to verify that the second element of this stream is 1. Using Theorem 6, we interpret `nats` as a program in dGTT where the corresponding equality follows from computation:

$$\begin{aligned}
\mathsf{next}(\texttt{head}) \circledast \texttt{tail}(\texttt{nats}) &= \mathsf{next}(\texttt{head}) \circledast \mathsf{mod}_\ell(\texttt{zip}(\texttt{const}(\lambda n.\ \mathsf{suc}(n))), \texttt{nats})\\
&= \mathsf{mod}_\ell(\texttt{head}(\texttt{zip}(\texttt{const}(\lambda n.\ \mathsf{suc}(n))), \texttt{nats}))\\
&= \mathsf{mod}_\ell(1)
\end{aligned}$$

These equalities are definitional; in dGTT our stream combinators reduce on-the-nose. Moreover, Theorem 7 ensures that if we were to interpret the original sGTT program into e.g., the topos of trees, the second element would indeed be 1.

This example is indicative of a general phenomenon: we can always program in sGTT to type-check a program and switch to dGTT to execute it and obtain a concrete result. Theorem 19 guarantees that this always yields a numeral, and Theorem 7 ensures that we obtain the expected results in standard models of sGTT such as the topos of trees.

# 6 Related work

GTT is closely related to a number of modal and guarded type theories. Most directly, GTT refines guarded recursion in MTT and improves upon the proposed guarded MTT presented by [11] by avoiding equality reflection. By replacing this rule with propositional Löb induction as well as a crisp identity induction principle, the theory remains practicable. Moreover, by extending with only these well-behaved principles we are able to benefit from the metatheory of MTT and its prototype proof assistant, which implements sGTT [26].

### Clocked Type Theory

Clocked Type Theory (CloTT) [2, 17, 18] is a family of closely related guarded type theories which also rely on dependent right adjoints to structure the later modality. Like sGTT, CloTT escapes Corollary 3 by preventing Löb induction in most circumstances. Rather than having a separate type theory where Löb induction does compute, CloTT carefully allows lob to unfold only when at the 'top-level'. In particular, if we were to construct the gadget used by Corollary 3 in CloTT, only the outer application of Löb would unfold.

Unfortunately, the additional definitional equality provided CloTT cannot be used in most circumstances; whenever a user is manipulating terms in a non-trivial context, Löb induction unfolds only up to a propositional equality, just as with sGTT. Moreover, in the existing formulation of CloTT in Agda [28], Löb induction is added as an *axiom* without even the definitional equality unfolding it at the top-level. The fact that CloTT has been used to formalize substantial arguments under these circumstances serves as further evidence for the practicability of propositional Löb induction as available in sGTT.

Finally, the metatheory of GTT is more developed than CloTT; the decidability of type-checking of GTT is proven— with a prototype implementation [26]—and an appropriate canonicity theorem is proven for the full system. Both of these results are conjectured for CloTT, but neither have been proven for a complete version of the system.

### Synthetic Tait computability

We have proven Theorem 19 by adapting (multimodal) synthetic Tait computability (STC) [10, 27]. This is the first application of STC—multimodal or not—which takes full advantage of the fact that STC applies to arbitrary sheaf topoi rather than just presheaf topoi. Moreover, this result expands the reach of STC to yet another class of languages: guarded recursion. Theorem 19 therefore opens the door to applying STC to guarded programming languages.

## 7 Conclusions

We contribute GTT, a type theory for guarded recursion featuring a novel decomposition into 'static' and 'dynamic' fragments. We prove that the static fragment enjoys normalization and decidability of type-checking and formulate and prove a guarded canonicity result for the 'dynamic' fragment. In so doing, we escape a no-go theorem (Corollary 3) showing that the presence of Löb induction is all but certain to disrupt the decidability of conversion. Finally, we have shown that this dichotomy between static and dynamic results in a usable system by carrying out a case-study based on synchronous programming.

In the future, we hope to further develop a prototype implementation of GTT based on [26] and potentially extend GTT with a richer mode theory for guarded recursion to permit more flexible guarded programming.

## References

1 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. Association for Computing Machinery, 2013. doi:10.1145/2500365.2500597.

2 Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. The clocks are ticking: No more delays! In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*

*(LICS)*. IEEE, 2017. URL: http://www.itu.dk/people/mogel/papers/lics2017.pdf, doi:10.1109/LICS.2017.8005097.

3 Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118–138, 2020. arXiv:1804.05236, doi:10.1017/S0960129519000197.

4 Lars Birkedal, Rasmus Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4), 2012. doi:10.2168/LMCS-8(4:1)2012.

5 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*, pages 20–35. Springer Berlin Heidelberg, 2016.

6 Sylvain Boulmé and Grégoire Hamon. Certifying synchrony for free. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 495–506, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

7 Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. Programming and reasoning with guarded recursion for coinductive types. In Andrew Pitts, editor, *Foundations of Software Science and Computation Structures*, pages 407–421. Springer Berlin Heidelberg, 2015.

8 Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs*, pages 120–134, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. doi:10.1007/3-540-61780-9_66.

9 Daniel Gratzer. Crisp induction for intensional identity types, 2021. Available at https://jozefg.github.io/papers/crisp-induction-for-intensional-identity-types.pdf.

10 Daniel Gratzer. Normalization for multimodal type theory, 2021. arXiv:2106.01414.

11 Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal Dependent Type Theory. *Logical Methods in Computer Science*, Volume 17, Issue 3, July 2021. URL: https://lmcs.episciences.org/7713, doi:10.46298/lmcs-17(3:11)2021.

12 Daniel Gratzer, G.A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20. ACM, 2020. doi:10.1145/3373718.3394736.

13 Adrien Guatto. A generalized modality for recursion. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18. ACM, 2018. doi:10.1145/3209108.3209148.

14 N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991. doi:10.1109/5.97300.

15 Nicolas Halbwachs. Synchronous programming of reactive systems. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification*, pages 1–16, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

16 Martin Hofmann and Thomas Streicher. Lifting Grothendieck universes. Unpublished note, 1997. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf.

17 Magnus Baunsgaard Kristensen, Rasmus Ejlers Møgelberg, and Andrea Vezzosi. Greatest hits: Higher inductive types in coinductive definitions via induction under clocks. 2021. URL: https://arxiv.org/abs/2102.01969, arXiv:2102.01969.

18 Bassel Mannaa, Rasmus Ejlers Møgelberg, and Niccolò Veltri. Ticking clocks as dependent right adjoints: Denotational semantics for clocked type theory. *Logical Methods in Computer Science*, Volume 16, Issue 4, December 2020. doi:10.23638/LMCS-16(4:17)2020.

19 Rasmus Ejlers Møgelberg. A type theory for productive coprogramming via guarded recursion. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, 2014. doi:10.1145/2603088.2603132.

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

21 Ian Orton and Andrew M. Pitts. Axioms for Modelling Cubical Type Theory in a Topos. *Logical Methods in Computer Science*, 14(4), 2018. arXiv:1712.04864, doi:10.23638/LMCS-14(4:23)2018.

22 François Pottier. A typed store-passing translation for general references. In *ACM Symposium on Principles of Programming Languages (POPL)*, Austin, Texas, January 2011. URL: http://cambium.inria.fr/~fpottier/publis/fpottier-fork.pdf, doi:10.1145/1925844.1926403.

23 Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, 16(1), 2020. arXiv:1706.07526.

24 Michael Shulman. The univalence axiom for elegant reedy presheaves. 17:81–106.

25 Michael Shulman. Univalence for inverse diagrams and homotopy canonicity. *Mathematical Structures in Computer Science*, 25(5):1203–1277, 2015. arXiv:1203.3253, doi:10.1017/S0960129514000565.

26 Philipp Stassen, Daniel Gratzer, and Lars Birkedal. A flexible multimodal proof assistant, 2022. To appear in Workshop on the Implementation of Type Systems 2022.

27 Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. PhD thesis, 2021. CMU technical report CMU-CS-21-142. doi:10.5281/zenodo.5709838.

28 Niccolò Veltri and Andrea Vezzosi. Formalizing $\pi$-calculus in guarded cubical agda. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 270–283, 2020.


## A Models of GTT

A model of sGTT extends a model of MTT instantiated with $\mathcal{M}$ with two additional constants for lob and unfold.

▶ **Definition 22.** *A model of MTT with mode $\mathcal{M}$ consists of a 2-functor $F : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ such that $F(m)$ has a terminal object along with the following structure:*

- *A* modal *representable natural transformation $\tau : \dot{\mathcal{T}} \longrightarrow \mathcal{T}$ [11].*
- *A choice of codes closing $\tau$ under dependent sums, modal dependent products, natural numbers, modalities, etc.*
- *A choice of code closing $\tau$ under identity types and left-lifting structure further equipping those types with a crisp induction principle.*

▶ **Definition 23.** *A model of sGTT consists of a model of MTT further supplemented with two elements of $\dot{\mathcal{T}}$ witnessing* lob *and* unfold.

▶ **Definition 24.** *A model of dGTT consists of a model of sGTT satisfying the following conditions:*

- lob *satisfies the expected definitional equality*
- unfold *is precisely interpreted by reflexivity*
- *There is a chosen initial object $\mathbf{0}$ in $F(m)$ which is preserved by each $F(\mu)$*
- $\dot{\mathcal{T}}(\mathbf{0}) = \mathcal{T}(\mathbf{0}) = \{\star\}$
- *A choice of objects representing the functors* $\hom(-.\{\mu\}, \mathbf{0})$ *for each $\mu$.*

## B Löb cosmoi

Here we give a more precise definition of Löb cosmoi. We presuppose knowledge with Section 3 of [10] which introduces the notion of *MTT cosmoi*.

▶ **Definition 25.** *A Löb cosmos is a strict 2-functor $G : \mathcal{M} \longrightarrow \mathbf{Cat}$ such that $G(m)$ (which we abusively write $G$) is a locally Cartesian closed category with an initial object and each $G(\mu)$ is a right adjoint. We require the following additional structure:*

1. *A morphism $\tau_G : \dot{\mathcal{T}}_G \longrightarrow \mathcal{T}_G$ in $G$*
2. *A choice of codes ensuring that $\tau_G$ is closed under dependent sums, weak natural numbers, identity types with a crisp induction principle and a weak Tarski universe. The last is itself closed under all connectives except the universe.*
3. *A choice of code making $\tau_G$ closed under modal dependent products: a code for $\mathbf{P}_{G(\mu)\tau_G}(\tau_G)$ for each $\mu : m \longrightarrow m$.*
4. *An element* lob *with the appropriate type and necessary equation.*
5. *For each $\mu$, there exists a chosen commuting square*

$$\begin{tikzcd} G(\mu)(\dot{\mathcal{T}}_n) \arrow[r] \arrow[d] & \dot{\mathcal{T}}_m \arrow[d] \\ G(\mu)(\mathcal{T}_n) \arrow[r] & \mathcal{T}_m \end{tikzcd} \tag{4}$$

6. *For each $\mu : m \longrightarrow m$ and $\nu : m \longrightarrow m$, there is a chosen lifting structure $G(\mu)(m) \pitchfork G(\mu \circ \nu)(\mathcal{T}_G) \times \tau_G$, where $m : G(\nu)(\dot{\mathcal{T}}_o) \longrightarrow G(\nu)(\mathcal{T}_G) \times_{\mathcal{T}_G} \dot{\mathcal{T}}_G$ is the comparison map induced by Diagram 4.*

▶ **Definition 26.** *A morphism of Löb cosmoi is a 2-natural transformation whose components are LCC functors preserving the initial object up to isomorphism and all structure strictly.*

We can form a Löb cosmos in $\mathbf{Sh}(\mathsf{Cx})$: sheaves over contexts and substitutions localizing the map $\mathbf{0}_{\mathbf{PSh}(\mathsf{Cx})} \longrightarrow \mathbf{y}(\mathbf{0})$.

▶ **Theorem 27.** *This topology is subcanonical.*

The universe of types and terms is taken is given by the standard representable universe (it is a map of sheaves by the $\eta$-laws governing $\mathbf{0}$) which we write $\tau_{\mathbf{Sh}(\mathsf{Cx})}$:

$$\mathcal{T}_{\mathbf{Sh}(\mathsf{Cx})}(\Gamma) = \{A \mid \Gamma \vdash_{\mathbf{d}} A\}$$
$$\dot{\mathcal{T}}_{\mathbf{Sh}(\mathsf{Cx})}(\Gamma) = \{(A, M) \mid \Gamma \vdash_{\mathbf{d}} M : A\}$$

We must show that $\tau$ is closed under the various required connectives. These proofs are based on the following observation.

▶ **Lemma 28.** *Viewed as a subcategory of $\mathbf{PSh}(\mathsf{Cx})$, $\mathbf{Sh}(\mathsf{Cx})$ is closed under limits and dependent products.*

▶ Remark 29. A type-theoretic proof of this the second half of this lemma follows from Lemma 1.26 of [23].

**Proof.** First consider a diagram $J : \mathcal{I} \longrightarrow \mathbf{Sh}(\mathsf{Cx})$, we wish to show that $\lim \mathbf{i} \circ J$ is a sheaf. As $\mathbf{i}$ is a right adjoint, it preserves limits and so $\lim \mathbf{i} \circ J = \mathbf{i}(\lim J)$ which is clearly a sheaf.

Next consider a pair of morphisms of sheaves $f : F \longrightarrow G$ and $g : G \longrightarrow H$. We wish to show that $\mathbf{i}(g)_*(\mathbf{i}(f)) = \mathbf{i}(g_* f)$. It suffices to show that they satisfy the same universal property, so fix $h : P \longrightarrow \mathbf{i}(H)$. Write $h'$ for the unique induced map $\mathbf{a}(P) \longrightarrow H$ extending $h$ and compute:

$$\begin{aligned}
\hom_{\mathbf{PSh}(\mathsf{Cx})/\mathbf{i}(H)}(h, \mathbf{i}(g_* f)) &\cong \hom_{\mathbf{Sh}(\mathsf{Cx})/H}(h', g_* f) \\
&\cong \hom_{\mathbf{Sh}(\mathsf{Cx})/G}(h' \times g, f) \\
&\cong \hom_{\mathbf{PSh}(\mathsf{Cx})/\mathbf{i}(G)}(\mathbf{i}(h') \times \mathbf{i}(g), \mathbf{i}(f)) \\
&\cong \hom_{\mathbf{PSh}(\mathsf{Cx})/\mathbf{i}(G)}(h \times \mathbf{i}(g), \mathbf{i}(f)) \\
&\cong \hom_{\mathbf{PSh}(\mathsf{Cx})/\mathbf{i}(H)}(h, \mathbf{i}(g)_* \mathbf{i}(f))
\end{aligned}$$

◀

We already know that $\tau$ is closed under dependent sums and products in $\mathbf{PSh}(\mathsf{Cx})$ so that there is a Cartesian square

$$\begin{tikzcd}
\sum_{A:\mathbf{i}(\mathcal{T}_{\mathbf{Sh}(\mathsf{Cx})})} \mathbf{i}(\dot{\mathcal{T}}_{\mathbf{Sh}(\mathsf{Cx})})^{\mathbf{i}(\tau_{\mathbf{Sh}(\mathsf{Cx})}[A])} \arrow[r] \arrow[d] & \mathbf{i}(\dot{\mathcal{T}}_{\mathbf{Sh}(\mathsf{Cx})}) \arrow[d] \\
\sum_{A:\mathbf{i}(\mathcal{T}_{\mathbf{Sh}(\mathsf{Cx})})} \mathbf{i}(\mathcal{T}_{\mathbf{Sh}(\mathsf{Cx})})^{\mathbf{i}(\tau_{\mathbf{Sh}(\mathsf{Cx})}[A])} \arrow[r] & \mathbf{i}(\mathcal{T}_{\mathbf{Sh}(\mathsf{Cx})})
\end{tikzcd}$$

However, sheaves are closed under the formation of finite limits and dependent products, so this is already a Cartesian square in $\mathbf{Sh}(\mathsf{Cx})$. A similar argument holds for dependent sums and natural numbers.

It remains to describe the interpretations of modalities on this category. Given a modality $\mu$ we define $F_\mu = (-.\{\mu\})^*$. We first must show that this sends sheaves to sheaves: there is an isomorphism $\mathbf{0}.\{\mu\} \cong \mathbf{0}$ so if $\Gamma$ is covered by the empty sieve so too is $\Gamma.\{\mu\}$. This functor also preserves all limits (they are computed pointwise in sheaves) so it is a right adjoint. We must show that it is weakly representable, but we already have such a representation in presheaves and it again descends.

▶ **Theorem 30.** *There is a Löb cosmos $\mathcal{S}$ sending $m$ to $\mathbf{Sh}(\mathsf{Cx})$ and each modality $\mu$ to $F_\mu$. The universe of types and terms is interpreted by $\tau$ and all the connectives descend from $\mathbf{PSh}(\mathsf{Cx})$ in the manner described above.*

## C The canonicity cosmos

In this appendix we highlight the novel constructions of the canonicity cosmos. As with Appendix B, we again presuppose some familiarity with [10]. Specifically, we make use of the internal presentation of cosomoi discussed in Section 6, the language of multimodal synthetic Tait computability introduced in Section 6, and the construction of the normalization cosmos from Section 7.

As with the construction of the normalization cosmos of [10], we now construct a Löb cosmos in $\mathcal{G}$ which lies strictly over the syntactic Löb cosmos $\mathcal{S}$. We proceed internally to $\mathcal{G}$, using the language of multimodal synthetic Tait computability to define a sequence of constants.

▶ **Definition 31.** *The language of multimodal synthetic Tait computability consists of extensional MTT together with a universe of strict propositions, a distinguished proposition* $\mathbf{syn} : \Omega$ *inducing a pair of complementary lex idempotent monads* $\bigcirc = \mathbf{syn} \to -$ *and* $\bullet = \mathbf{syn} \vee -$.

*Furthermore, each universe of MTT satisfies the internal realignment axiom of [21] and both lex monads commute with $\langle \mu \mid - \rangle$.*

A word of caution is required as—unlike in [10]—$\langle \mu \mid \mathbf{syn} \rangle \neq \mathbf{syn}$. This divergence stems essentially from the fact that $\langle \ell \mid - \rangle$ does not preserve $\mathbf{0}$. However, this is immaterial as $\langle \mu \mid - \rangle$ does commute with both $\bigcirc$ and $\bullet$.

When interpreted into $\mathcal{G}$, the $\bigcirc$-modal correspond to objects in $\mathcal{S}$. Accordingly, the syntactic model is manifested as a series of $\bigcirc$-modal types ($\mathsf{Ty}$, $\mathsf{Tm}$, etc.) and constants as in [10]. We then must construct a series of types and constants in $\mathsf{MTT}$ which collapse to the syntactic model under $\mathbf{syn}$.

▶ Remark 32. We write $\bigcirc_z A(z)$ as shorthand for the dependent open modality $\prod_{z:\mathbf{syn}} A(z)$.

▶ Remark 33. We write $\{A \mid z : \phi \mapsto a(z)\}$ for the type of elements of $A$ which are equal to $a(z)$ under the assumption $z : \phi$. We treat the coercion from $\{A \mid z : \phi \mapsto a(z)\}$ to $A$ as silent.

▶ Remark 34. As a convention, we will write $\texttt{foo}^*$ for the constant lying over the element $\texttt{foo}$ of the syntactic model.

We begin by defining the following type and using realignment to ensure that it lies strictly over $\mathsf{Ty}$:

$$\begin{aligned} &\textbf{record } \mathsf{Ty}^* : \{\mathsf{U}_2 \mid z : \mathbf{syn} \mapsto \mathsf{Ty}(z)\} \textbf{ where} \\ &\quad \mathsf{code} : \bigcirc_z \mathsf{Ty}(z) \\ &\quad \mathsf{pred} : \{\mathsf{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}(z, \mathsf{code})\} \end{aligned} \tag{5}$$

We further define $\mathsf{Tm}^* : \{\mathsf{Ty}^* \to \mathsf{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}(z)\}$ to send $A$ to $\mathsf{pred}(A)$.

▶ **Lemma 35.** *The gluing model supports dependent products and sums.*

**Proof.** This follows more-or-less immediately by choosing the computability structures of a dependent product (respectively, sum) to be the dependent product (resp. sum) of the component computability structures. ◀

▶ **Lemma 36.** *The gluing model supports weak natural numbers.*

**Proof.** We begin by defining $\mathsf{Nat}^*$:

$$\begin{aligned} &\mathsf{code}(\mathsf{Nat}^*) = \mathsf{Nat} \\ &\mathsf{pred}(\mathsf{Nat}^*) = \textstyle\sum_{M:\bigcirc_z \mathsf{Nat}(z)} \bullet \left[\sum_{n:\mathsf{nat}} \bar{n} = M\right] \end{aligned}$$

The two introduction forms deviate only slightly from the standard definitions:

$$\begin{aligned} &\mathsf{zero}^* = (\mathsf{zero}, \eta(0, \star)) \\ &\mathsf{succ}^* = \lambda M.\ (\mathsf{succ}(M), (n, \star) \leftarrow \pi_1(M); \eta(n+1, \star)) \end{aligned}$$

We finally define $\mathsf{rec}^*$:

$$\mathsf{rec}^*(A, M_z, M_s, N) = \begin{cases} \mathsf{rec}^*(z, M_z, M_s, N) & \pi_1(N) = \iota_0(z) \\ M_s^n(M_z) & \pi_1(N) = \iota_1(n, \star) \end{cases}$$ ◀

▶ **Lemma 37.** *The gluing model supports a weak Tarski universe.*

**Proof.** The construction of the universe follows the construction given in [10] without variation. As to be expected, $\mathsf{code}(\mathsf{Uni}^*) = \mathsf{Uni}$. The predicate is only slightly more complext:

$$\mathsf{pred}(\mathsf{Uni}^*) = \textstyle\sum_{A:\bigcirc_z \mathsf{Tm}(z,\mathsf{Uni})} \{\mathsf{U}_0 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}(z, \mathsf{El}(z, A(z)))\}$$

We must also define the type family $\mathsf{El}^*$:

$$\mathsf{code}(\mathsf{El}^*(M)) = \lambda z.\ \pi_0(M)$$

It remains to associate the computability data:

$$\mathsf{pred}(\mathsf{El}^*(M)) = \pi_1(M)$$

The construction of various elements in this universe is routine. ◀

▶ **Lemma 38.** *The gluing model supports identity types with crisp induction principles.*

**Proof.** We define $\mathsf{Id}^*(A, a_0, a_1)$:

$$\mathsf{code}(\mathsf{Id}^*(A, a_0, a_1)) = \lambda z.\ \mathsf{Id}(z, A, a_0, a_1)$$
$$\mathsf{pred}(\mathsf{Id}^*(A, a_0, a_1)) = \textstyle\sum_{p:\bigcirc_z \mathsf{Id}(z,A,a_0,a_1)} \bullet[(a_0 = a_1) \times (p = \lambda z.\ \mathsf{refl}(z, a_0))]$$

It remains to define the introduction and elimination forms for identity types. First, introduction:

$$\mathsf{refl}^*(a) = (\lambda z.\ \mathsf{refl}(z, a), (\star, \star))$$

The elimination rule is complicated by the necessity to account for crisp induction. Therefore, we follow the definition given by [9], not the simpler version presented by [10]. In particular, we must define a constant of the following type:

$$\begin{aligned}
\mathsf{J}^*_\mu : {} & (\mu \mid A : \mathsf{Ty}^*)(B : (\mu \mid a_0, a_1 : \mathsf{Tm}^*(A))(\mu \mid p : \mathsf{Tm}^*(\mathsf{Id}^*(A, a_0, a_1))) \to \mathsf{Ty}^* m) \to \\
& (b : (\mu \mid a : \mathsf{Tm}^*(A)) \to \mathsf{Tm}^*(B(a, a, \mathsf{refl}^*(a)))) \to \\
& (\mu \mid a_0, a_1 : \mathsf{Tm}^*(A))(\mu \mid p : \mathsf{Tm}^*(\mathsf{Id}(A, a_0, a_1))) \to \\
& \{\mathsf{Tm}^*(B(a_0, a_1, p)) \mid z : \mathbf{syn} \mapsto \mathsf{J}_\mu(A, B, b, p)\}
\end{aligned}$$

To this end, assume that we are given $B$, $b$, $a_0$, $a_1$, and $p$ with the appropriate types. We can commute $\langle \mu \mid - \rangle$ past part of the definition of $\mathsf{Tm}^*(A, a_0, a_1)$ and assume that $p = (l, r)$ belongs to the following type:

$$\textstyle\sum_{p:\bigcirc_z \langle \mu \mid \mathsf{Id}(z,A,a_0,a_1)\rangle} \bullet[(a_0 = a_1) \times (p = \lambda z.\ \mathsf{mod}_\mu(\mathsf{refl}(z, a_0)))]$$

We therefore proceed by case analysis on $r$:

$$\begin{cases} \mathsf{J}(z, B, b, a_0, a_1, p) & q = \mathsf{in}_0(z) \\ b(a_0) & q = \mathsf{in}_1(\star, \star) \end{cases}$$ ◀

▶ **Lemma 39.** *The gluing model supports modal types.*

**Proof.** We begin by defining the $\mathsf{Mod}^*[\mu](A)$:

$$\mathsf{code}(\mathsf{Mod}^*[\mu](A)) = \lambda z.\ \mathsf{Mod}[\mu](z, A)$$

$$\mathsf{pred}(\mathsf{Mod}^*[\mu](A)) = \textstyle\sum_{M:\bigcirc_z \mathsf{Mod}[\mu](z,A)} \bullet\left[\sum_{N:\langle\mu|\mathsf{Tm}^*(A)\rangle} \mathsf{m}[\mu](N) = M\right]$$

The introduction form is defined as follows:

$$\mathsf{m}^*[\mu](m) = (\lambda z.\ \mathsf{m}[\mu](z,m), \eta(m,\star))$$

Most involved is modal elimination:

$$\mathsf{letmod}^*[\mu;\nu](B,b,m) = \begin{cases} \mathsf{letmod}[\mu;\nu](z,B,b,m) & \pi_1(m) = \iota_0(z) \\ b(a) & \pi_1(m) = \iota_1(a,\star) \end{cases}$$

That this satisfies the required $\beta$ rule and alignment conditions follows immediately from calculation. ◀

▶ **Lemma 40.** *There is a constant* $\mathsf{lob}^* : (\langle\ell \mid \mathsf{Tm}^*(A)\rangle \to \mathsf{Tm}^*(A)) \to \mathsf{Tm}^*(A)$ *lying strictly over* $\mathsf{lob}$ *and satisfying the unrolling rule.*

**Proof.** Let us fix $f : \langle\ell \mid \mathsf{Tm}^*(A)\rangle \to \mathsf{Tm}^*(A)$. We must construct $\mathsf{lob}^*(f)$. We now use the fracture theorem:

$$A \cong \bigcirc A \times_{\bullet\bigcirc A} \bullet A$$

We have the left component of this pullback already: $\lambda z.\ \mathsf{lob}(z,f)$. It remains to construct an element of $\bullet\mathsf{Tm}^*(A)$ which coheres appropriately with $\mathsf{lob}$. Here we use $\mathsf{lob}_\bullet$ induction with the target $\bullet\mathsf{Tm}^*(A)$. We must produce a function $\langle\ell \mid \bullet\mathsf{Tm}^*(A)\rangle \to \bullet\mathsf{Tm}^*(A)$. We have $g = \bullet f : \langle\ell \mid \bullet A\rangle \longrightarrow \bullet A$. Therefore $\mathsf{lob}_\bullet(g) : \bullet\mathsf{Tm}^*(A)$. We must show the following:

$$(\bullet\eta_\bigcirc)(\mathsf{lob}_\bullet(g)) = \eta_\bullet(\lambda z.\ \mathsf{lob}(z,f)) \tag{6}$$

Let us prove this through Löb induction, available because equality of terms of $\bullet$-modal type is a $\bullet$-modal type. We then assume Equation (6) under $\langle\ell \mid -\rangle$. Taking advantage of $\langle\ell \mid -\rangle$ as a fully-fledged dependent right adjoint, we rephrase this assumption as the equality

$$\mathsf{next}(\bullet\eta_\bigcirc(\mathsf{lob}_\bullet(g))) = \mathsf{next}(\eta_\bullet(\lambda z.\ \mathsf{lob}(z,f)))$$

We may simplify this by taking advantage of our ability to commute MTT modalities past those induced by gluing:

$$\bullet(\eta_\bigcirc \circ \mathsf{next})(\mathsf{lob}_\bullet(g)) = \eta_\bullet(\lambda z.\ \mathsf{next}(\mathsf{lob}(z,f))) : \bullet\bigcirc\langle\ell \mid \mathsf{Tm}^*(A)\rangle$$

Returning now to our goal, after applying both computation rules for the different forms of Löb induction, we are left with the following:

$$\bullet\eta_\bigcirc(g((\bullet\mathsf{next})(\mathsf{lob}_\bullet(g)))) = \eta_\bullet(\lambda z.\ f(\mathsf{next}(\mathsf{lob}(z,f))))$$

Rewriting, we obtain

$$\bullet(\bigcirc f \circ \eta_\bigcirc \circ \mathsf{next})(\mathsf{lob}_\bullet(g)) = (\bullet\bigcirc f)(\eta_\bullet \lambda z.\ \mathsf{next}(\mathsf{lob}(z,f)))$$

The result now follows from our induction hypothesis. ◀

▶ **Theorem 41.** *$\mathcal{G}$ supports a Löb cosmos equipped with a projection onto $\mathcal{S}$.*

We shown the existence of a model in $\mathcal{G}$ lying over the syntactic Löb cosmos in $\mathcal{S}$. From this we aim to extract canonicity.

▶ **Theorem 19** (Guarded canonicity). *Given a term* $\mathbf{0}[\mu].\{\nu\} \vdash_{\mathbf{d}} M : A$*, the following holds*

- *If* $A = \mathsf{nat}$ *then there exist a numeral* $k$ *such that* $M = \bar{k}$
- *If* $A = \langle \xi \mid B \rangle$ *then* $M = \mathsf{mod}_{\xi}(N)$ *for some* $\mathbf{0}[\mu].\{\nu \circ \xi\} \vdash_{\mathbf{d}} N : B$*.*

**Proof.** Fix a term $\mathbf{0}[\mu].\{\nu\} \vdash_{\mathbf{d}} M : A$. By Theorem 10 we obtain an element of $M^* : A^*$ which lies over $M$ up to isomorphism of contexts. We have the following commutative square in $\mathbf{Sh}(P)$

$$\begin{tikzcd}
(F_\mu)_!(F_\nu(\mathbf{0}_{\mathbf{Sh}(P)})) \arrow[r] \arrow[d] & (\tau_{\mathcal{G}}[A^*])_0 \arrow[d, "\tau_{\mathcal{G}}[A^*]"] \\
i^*\mathbf{y}(\mathbf{0}[\mu].\{\nu\}) \arrow[r, "i^*\mathbf{y}(M)"'] & i^*(\tau_{\mathcal{S}}[A])
\end{tikzcd} \tag{7}$$

We instantiate this diagram of sheaves at $(\mu, \nu)$. Let us construct an $(\mu, \nu)$-point of $(F_\nu)_!(F_\mu(\mathbf{0}_{\mathbf{Sh}(P)}))$. By functoriality, it suffices to construct a $(\mu, \mathsf{id})$ point of $F_\mu(\mathbf{0})$. Transposing, it is sufficient to construct a $(\mu, \mu)$-point of $\mathbf{0}$, which exists uniquely (it is a map between initial objects). Calculating, this lies over $\mathsf{id} : i^*\mathbf{y}(\mathbf{0}[\mu].\{\nu\})(\mu, \nu)$. This, together with the construction of the computability data for $\mathsf{nat}$ and $\langle \xi \mid - \rangle$, yields the desired result. ◀