Normalisation for Multimodal Type Theory

Daniel Gratzer
gratzer@cs.au.dk
Aarhus University
Aarhus, Denmark

Abstract

We prove normalisation for MTT, a general multimodal dependent type theory capable of expressing modal type theories for guarded recursion, internalized parametricity, and various other prototypical modal situations. We prove that deciding type checking and conversion in MTT can be reduced to deciding the equality of modalities in the underlying modal situation, immediately yielding a type checking algorithm for all instantiations of MTT in the literature. This proof follows from a generalization of synthetic Tait computability—an abstract approach to gluing proofs—to account for modalities. This extension is based on MTT itself, so that this proof also constitutes a significant case study of MTT.

CCS Concepts:
• Theory of computation → Type theory; Modal and temporal logics: Categorical semantics.

Keywords: type theory, modal type theory, normalization, modalities, categorical semantics, Artin gluing

ACM Reference Format:

1 Introduction

If type theory is classically the study of objects invariant under change of context, modal type theory is the study of adding non-invariant connectives—modalities—to type theory. Given that many natural features of particular models of type theory are not invariant under substitution, modal type theories have sparked considerable interest. By nature, however, modal type theories must thread the needle of presenting modalities in such a way that the classical substitution theorems of type theory still hold.

Typically, modal type theories require modifications to the apparatus of contexts and substitutions. Unfortunately, these tweaks are often more art than science, with expert attention required even to make the most trivial modification to the modal structure of a type theory. In order to address this complexity, general modal type theories have been introduced [14, 25]. These theories can be instantiated by a description of a modal situation to produce a system enjoying the theorems usually laboriously proved by experts.

1.1 Multimodal type theory

We focus on one such general modal type theory: MTT [14]. MTT can be instantiated with an arbitrary collection of modalities and transformations between them to yield a highly usable syntax. The modalities in MTT behave like (weak) dependent right adjoints (DRAs) [8] so that MTT can be used to internalize nearly any right adjoint. This flexibility allows MTT to encode calculi for guarded recursion, internalized parametricity, and other handcrafted calculi.

More precisely, MTT can be instantiated by a mode theory, a strict 2-category describing modes, modalities, and natural transformations between these modalities. This 2-categorical structure is then reflected into the structure of substitutions in MTT, ensuring that e.g., a transformation between two modalities μ and ν gives rise to a function (μ | A) → (ν | A).

While this flexibility allows MTT to accommodate many interesting calculi, it becomes proportionally more challenging to prove metatheoretic results about MTT. In particular, the rich substitution structure inherited from the mode theory can introduce subtle equations between terms. The proof that the crisp induction principles can be reconstructed in MTT [15, Theorem 10.4], for instance, exemplifies this and hinges on many such calculations. In fact, the metatheoretic results established by Gratzer et al. [14] (soundness and canonicity) are results on closed terms in MTT, allowing their proofs to avoid the majority of the substitution apparatus.

Crucially, it remained open whether MTT admitted a normalization algorithm and, consequently, whether type checking was decidable. Even in the presence of a normalization algorithm MTT cannot admit an unconditional type checking algorithm: it is not only necessary to have a decision procedure for terms in the language, but also for modalities and 2-cells as both appear in terms for MTT.

In this paper we show the best possible result holds: MTT admits an unconditional normalization algorithm and conversion of normal forms is decidable if and only if conversion...
is decidable in the mode theory. As corollaries, we show that type constructors in MTT are always injective and that type checking is decidable when the mode theory is decidable.\footnote{This requirement is potentially nontrivial e.g., the word problem for groups is known to be undecidable and is subsumed by the problem for 2-categories.}

## 1.2 Normalization for type theories

A normalization algorithm must begin by defining normal forms. Their precise formulation varies depending on the situation but they always satisfy two crucial properties. First, the equality of normal forms \( u = v \) is clearly decidable—often no more than structural equality—and there is a function \( \text{dec}(u) \) decoding a normal form to a term of the same type.

Relative to a notion of normal form, a normalization algorithm sends a term \( \Gamma \vdash M : A \) to a normal form \( \text{nf}_{\Gamma}(M, A) \) such that \( (\text{nf}_{\Gamma}(-, A), \text{dec}(-)) \) lifts to an isomorphism between equivalence classes of terms of \( A \) and normal forms [1].

Proving normalization is an involved affair. Traditionally, one begins by fixing a strongly normalizing confluent rewriting system presenting the equational theory of the type theory. The normal forms are then exactly the terms of the theory which cannot be further reduced. This approach does not scale, however, to type theories with type-directed equations such as the unicity principles of dependent sums and the unit type. These equations defy attempts to present them in a rewriting system and require type-directed algorithms.

The preeminent type-directed technique for normalization is normalization-by-evaluation (NbE) [1]. Proving that an NbE algorithm works, however, is an extremely intricate affair involving a variety of complex constructions. Recent work by Gratzer et al. [17] extended NbE to a type theory with an idempotent comonad but even in this minimal case the correctness proof occupied a 90 page technical report [18].

### 1.2.1 Normalization-by-gluing

These difficulties are not unique to modal type theories, and a long line of research focuses on taming the complexity of NbE through gluing [3, 4, 12, 13, 34, 35]. This line of work recasts normalization algorithms as the construction of models of type theory in categories defined by Artin gluing. Objects in this model are proof-relevant logical relations: proof-relevant predicates on syntax taken up to definitional equality.

To prove normalization, types in the gluing model are supplemented with reify and reflect maps reminiscent of those from classical NbE which e.g., send a witness of a predicate on some term to a normal form of that term. This structure, together with the initiality of syntax, allows us to extract a normalization algorithm from this model.

Normalization-by-gluing proofs force a number of design decisions which are not inherently categorical, but vital to simplifying the proof. Working with syntax up to definitional equality, for instance, may appear a minor point but it is essential: connectives in type theory only have universal properties when considered up to definitional equality. Only when working with equivalences classes therefore, can we use these universal properties and benefit from existing results. For instance, the local cartesian closure of the gluing category can be used to automatically close the model under dependent products. In a similar vein, proof-relevance is essential for a natural interpretation of the universe, where witnesses encode the data of a type in the model. In a proof-irrelevant setting, one must laboriously work around the inability to encode this data indirectly [2].

### 1.2.2 Synthetic Tait computability

Using gluing to prove normalization is certainly an improvement over ‘free-hand’ proofs of normalization-by-evaluation, but the picture is not as rosy at may first appear. Models of type theory are subject to a variety of strict equations which often force external constructions, where naturality obligations can be prohibitive. Worse, the passage between between mathematics internal to the gluing category and external constructions is difficult, and the boundary frequently raises subtle mismatches.

We follow Sterling and Harper [36] and adopt a synthetic approach to gluing. By ensuring that the gluing category is a (presheaf) topos we obtain a rich internal language for manipulating logical relations synthetically. By further extending this language with a pair of lex monads and the strictification axiom stated by Orton and Pitts [28], we can construct the normalization model completely internally.

Sterling and collaborators have termed this approach synthetic Tait computability (STC) and shown that working internally simplifies constructions involved in the gluing model, making it practical to prove normalization for even extremely complex type theories like cubical type theory [34, 35].

### 1.2.3 Synthetic Tait computability for MTT

Unlike Martin-Löf type theory or cubical type theory, a model of MTT is not a single category equipped with additional structure. Rather, a model is a network of categories, each supporting their own individual model of type theory which are then connected by various adjoints and natural transformations. The internal language of any of these categories is insufficient to construct the gluing model, so it is necessary to generalize from working in the extensional type theory of a topos to working in all topoi simultaneously using extensional MTT. Each topos then comes equipped with the structure of STC: a pair of lex monads and a strictification axiom. We prove that this mode-local structure is respected by the MTT modalities between topoi and call the resulting language multimodal synthetic Tait computability.

With this machinery, we are able to give a concise and conceptual construction of the gluing model and extract the first normalization algorithm for multimodal type theory.

## 1.3 Contributions

We contribute a normalization algorithm for MTT equipped with the full suite of connectives: dependent sums, products, booleans, intensional identity types, a universe, and modal
We collect the key ideas of MTT we formally view tensional identity types, booleans—are all incorporated in of type theory—dependent sums, dependent products, in-

In order to prove this result, we advance modern gluing techniques to apply to modal type theories and demonstrate that extensional MTT itself is a suitable metalanguage for carrying out the proof of normalization-by-gluing. We further argue that these techniques scale by extending the proof to a version of MTT supplemented with crisp induction principles and deduce e.g. normalization continues to hold.

Section 2 gives a brief tutorial on MTT and introduces normal forms for this type theory. In Section 3, we discuss the models of MTT and relax the definition of a model of MTT to obtain MTT cosmoi. We prove that the syntactic cosmos enjoys a privileged position among MTT cosmoi (Theorem 3.8). Section 4 introduces multimodal synthetic Tait computability and shows that gluing together a network of topoi results in a model of extensional MTT equipped with STC structure in each mode (Theorem 4.13). Finally, in Section 5 we construct the normalization cosmos (Theorem 5.7) and extract the normalization function in Section 6 (Theorem 6.3). Section 7 extends this proof to support crisp induction.

**Metatheory.** We work in a constructive metatheory such as IZF equipped with a hierarchy of Grothendieck universes.

## A primer on MTT

We collect the key ideas of MTT [15]. First, as mentioned in Section 1, MTT is parametrized by a mode theory: a strict 2-category $\mathcal{M}$ whose objects are modes, morphisms are modalities, and 2-cells are maps between modalities. Henceforth, we will work with MTT over a fixed mode theory $\mathcal{M}$. Second, we formally view MTT as a particular generalized algebraic theory (GAT). Accordingly, binding is handled by de Bruijn indices and the theory uses explicit substitutions [26]. We refer the reader to Gratzer et al. [15, Section 4] for a discussion of the motivations and consequences behind this choice.

### Mode-local connectives in MTT

Each mode in MTT constitutes its own separate type theory. In fact, each mode $m$ is equipped with its own copy of the judgments of type theory e.g., $\Gamma \vdash A @ m$, $\Gamma \vdash M : A @ m$. Much of the theory of MTT is *mode-local* and only mentions a single copy of these judgments at a time. For these connectives the rules are precisely the standard rules from MLTT, replicated for each mode. The connectives of type theory—dependent sums, dependent products, intensional identity types, booleans—are all incorporated in this fashion. Each mode also contains a *weak universe* à la Tarski. Explicitly, this means that there are separate codes and an El($\_\_$) operation decoding a code to a type, but the decoding operation only commutes with connectives up to isomorphism. While the restriction to weak universes is not fundamental, it simplifies the proof and recent implementations have shown them to be practical [29].

### 2.2 Modalities in MTT

MTT draws inspiration from Fitch-style type theories [8,11] with each modality defined together with an adjoint action on contexts. Accordingly, each $\mu : n \rightarrow m$ defines a context former sending contexts in mode $m$ to contexts in mode $n$ and this is then used to define modal types $(\mu \mid A)$:

\[
\begin{align*}
\Gamma \vdash A @ n & \\
\Gamma \vdash \mu @ A n & \\
\Gamma \vdash (\mu \mid A) @ m & \\
\end{align*}
\]

These context operations assemble into a 2-functor $m \mapsto \text{C.vx}_m$ from $\mathcal{M}_{\text{coop}}^2$ to the category of contexts. Concretely, a substitution $\Delta \vdash \gamma : \Gamma @ m$ lifts to a substitution $\Delta, \{\mu\} \vdash \gamma : \Gamma, \{\mu\} @ n$ and each 2-cell $\alpha : n \rightarrow m$ induces a substitution $\Gamma, \{\mu\} \vdash \{\alpha\} : \Gamma, \{\alpha \circ \xi\} \text{cx} @ o$.

Two basic questions remain: what is the elimination principle for $(\mu \mid A)$ and which terms can be constructed in the context $\Gamma, \{\mu\}$? Both of these problems are addressed through the same idea, the final component of MTT. We generalize the context extension $\Gamma.A$ from MLTT to annotate each variable with a modality:

\[
\begin{align*}
\Gamma \vdash \text{cx} @ m & \\
\Gamma, \{\mu\} \vdash A @ n & \\
\Gamma, (\mu \mid A) \text{cx} @ m & \\
\end{align*}
\]

Intuitively, $\Gamma, (\mu \mid A)$ plays the same role as $\Gamma, (\mu \mid A)$ and comes equipped with a similar universal property: a substitution $\Delta \vdash \gamma : \Gamma, (\mu \mid A) @ m$ is precisely determined by a substitution $\Delta \vdash \gamma' : \Gamma @ m$ and a term $\Delta, \{\mu\} \vdash M : A[\gamma', \{\mu\}] @ n$. Depsite this similarity, they occupy different positions in the theory. The variable rule of MTT is adjusted to take into account modal annotations and require that the modalities in the context must ‘cancel’ a variable’s annotation:

\[
\begin{align*}
\Gamma \vdash \text{cx} @ m & \\
\Gamma, \{\mu\} \vdash A @ n & \\
\Gamma, (\mu \mid A), \{\mu\} \vdash A[\uparrow, \{\mu\}] @ n & \\
\end{align*}
\]

As in Martin-Löf type theory, it is necessary to apply a weakening substitution $\uparrow$ to $A$ when describing the type of $v$. The normal context extension rule and normal variable rules are special cases of this principle, obtained by setting $\mu = \text{id}$.

**Remark 1.** From the view of Fitch-style type theories, where $\text{\_}_\mu$ is a left adjoint to the modal type, this rule plays the role of the count; it allows us to pass from $L(R(A))$ to $A$.

\[\text{Given a 2-category } \mathcal{C}, \text{ recall that } \mathcal{C}^{\text{coop}} \text{ is a 2-category with the same objects as } \mathcal{C} \text{ but with 1- and 2-cells reversed.}\]
The addition of modal annotations creates a redundancy in our system: we may hypothesize \( v \mid A \) with annotation \( \mu \) or directly hypothesize over \( A \) with annotation \( \mu \circ v \). There is a substitution navigating in one direction, but not the other:

\[
\Gamma \vdash \mu \circ v \mid A \rightarrow \top \mod(v) \circ \Gamma : \Gamma.\mu \mid (v \mid A) \circ o
\]

This mismatch is addressed through elimination for \( (v \mid -) \). Informally, this rule ensures that these two contexts are isomorphic ‘from the perspective of a type’:

\[
v : o \rightarrow n \quad \mu : n \rightarrow m
\]

\[
\Gamma \vdash c x @ m \quad \Gamma.\mu, \{v\} \vdash A \circ o
\]

\[
\Gamma.\mu \mid (v \mid A) \circ B @ m \quad \Gamma.\mu \circ M_0 : \langle v \mid A \rangle \circ @ n
\]

\[
\Gamma.\mu \circ v \mid A \circ \Gamma \vdash M_1 : B[\top \mod(v, v_0)] \circ @ m
\]

\[
\Gamma \vdash \lambda(M) : (\mu \mid A) \rightarrow B @ m
\]

\[
\Gamma \vdash M : (\mu \mid A) \rightarrow B @ m \quad \Gamma.\mu \vdash N : A @ n
\]

\[
\Gamma \vdash M(N) : B[\top \mod(v_0)] @ m
\]

This feature will be a useful convenience when using MTT as a metalanguage, but for space reasons we will only prove normalization for non-modal dependent products.

### 2.3 Normal and neutral forms in MTT

As mentioned in Section 1.2, the starting point for normalizations is the definition of normal form. In MTT—as in other type theories—normal forms are presented together with a class of neutral forms. Intuitively, normal forms capture terms in \( \beta \)-normal and \( \eta \)-long form while neutrals are chains of eliminations applied to a variable.

We define normal and neutral forms as separate syntactic classes, equipped with their own family of typing judgments and decoding functions sending them to terms. Dependency complicates this definition as various typing rules require substitution in the types of premises or the conclusion. Unfortunately, it is just as hard to define substitution on normal forms as it is to define normalization in general [40]. Accordingly, a normal form (resp. neutral, normal type) is typed by the judgment \( \Gamma \vdash \mu \circ u \mid A \circ @ m \) (resp. \( \Gamma \vdash e : A \circ @ m, \Gamma \vdash \tau \circ @ m \) where \( A \) is not required to be any sort of normal form. Furthermore, these judgments are defined inductively-recursively with decoding functions \([u] \) (resp. \([e], [\tau]\) which send a normal form (resp. neutral, normal type) to its corresponding piece of syntax. Normal and neutral forms for mode-local connectives are unchanged from their standard presentation in type theory:

\[
\begin{align*}
\text{(Normals)} & \quad u \coloneqq \lambda(u) \mid up(e) \mid \text{mod}_{\mu}(u) \mid \ldots \\
\text{(Neutral)} & \quad e \coloneqq v_1^\epsilon \mid e(u) \mid \text{letmod}(\mu; v; \tau; e; u) \mid \ldots \\
\text{(Normal types)} & \quad \tau \coloneqq \tau \rightarrow \sigma \mid \langle \mu \mid \tau \rangle \mid \text{El}(u) \mid \ldots
\end{align*}
\]

We defer a more complete presentation of the judgments and decoding function to Fig. 3, but remark that the neutral form for variables is annotated with a 2-cell and index, decoding to \( v_0 \) together with a combination of weakening and 2-cell substitutions \( \top \circ \) and \( \{\alpha\} \).

To ensure that normal forms are \( \eta \circ \)-long, neutrals can only be ‘injected’ into normals by up(−) for types without \( \eta \) law e.g., at modal types but not at dependent products. Finally, we emphasize that normal forms are freely generated, so their equality is decidable if and only if equality of modalities and 2-cells is decidable.

**Renamings.** While normal and neutral forms are not stable under substitution, they are stable under the restricted class of renamings. While we properly define renamings through their own judgment and decoding function in Fig. 4, they are intuitively the smallest class of substitutions closed under weakening, composition, identity, modal substitutions (−, \( \{\mu\}, \{\alpha\} \)), and extension by variables \( v_1^\tau \).

Renamings are easily seen to act on normal forms, neutral forms, and normal types. Unlike normals and neutrals, however, renamings are taken up to a definitional equality which ensures that e.g., composition is associative and that modal substitutions organize into a 2-functor. This poses no issue as the action of renamings on normals and neutrals send definitionally equal renamings to identical normals and neutrals, ensuring that the action lifts to equivalences classes.

A nontrivial definitional equality on renamings is essential, however, as it ensures that the class of contexts of mode \( m \) and renamings between them organizes into a category \( \text{Ren}_m \) and that the assignments \( m \mapsto \text{Ren}_m, \mu \mapsto -, \{\mu\} \), and \( \alpha \mapsto \{\alpha\} \) define a 2-functor \( \text{M}^{\text{coop}} \rightarrow \text{Cat} \).

**Lemma 2.1.** The decoding of renamings to substitutions gives a 2-natural transformation \( [\cdot] \colon \text{Ren}_- \rightarrow \text{Ctx}_- \).

### 3 Models and cosmoi

Gratzer et al. [15] introduced MTT as a generalized algebraic theory so that MTT is automatically equipped with a category of models. A standard result of GATs ensures that the syntax of MTT organizes into an initial model which opens the possibility of semantic methods for proving results about syntax. Gratzer et al. [15, Section 5] then repackage the definition of models in the language of natural models [6].
3.1 Natural models of MTT

We begin by recalling the presentation of a model of MTT given by Grätzer et al. [15]. Recall that a natural model of type theory [6] is a pair of a category $C$—representing a category of contexts—together with a representable natural transformation $\tau : T^* \to T$.

**Definition 3.1.** A natural transformation $f : X \to Y : \text{PSh}(C)$ is representable when each fiber of $f$ over a representable point of $Y$ is itself representable i.e., $y(C) \times_Y X$ is representable for each $y(C) \to Y$.

Intuitively, $\tau$ displays pairs of terms with their types over types. These two objects organize into presheaves through substitution on terms and types. With this in mind, the representability condition encodes context extension. In order to adapt this to MTT, we can no longer consider just a category of contexts. The existence of multiple modes mandates that we consider a 2-functor of contexts $F : \mathcal{M}^\text{coop} \to \text{Cat}$. The action of modalities $F(\mu) : F(m) \to F(n)$ gives the semantic equivalent of $\cdot \cdot \cdot \{\mu\}$, while the 2-cell component $F(\alpha)$ interprets $\{\alpha\}$.

Each mode $m : M$ is equipped with a morphism $\tau_m : T^*_m \to T_m : \text{PSh}(F(m))$ representing the terms and types of mode $m$ and each modality $\mu : n \to m$ induces a functor which acts by precomposition $F(\mu)^*$.

**Definition 3.2.** A model of MTT without any type constructors is a strict 2-functor $F : \mathcal{M}^\text{coop} \to \text{Cat}$ together with a collection of morphisms $\tau_m : T^*_m \to T_m : \text{PSh}(F(m))$ such that $F(\mu)^*(\tau_n)$ is representable for each $\mu : n \to m$.

Connectives are individually specified on top of this structure. For instance, the following pullback square in $\text{PSh}(F(m))$ for each mode $m$ ensures closure under dependent products:

$$\Sigma A : T_m \Sigma B \tau_m[A] : \tau_m \prod_{a : \tau_m(A)} \tau_m[B(a)] \to T^*_m \quad (1)$$

Diagram 1 takes advantage of the model of extensional MLTT in a presheaf topos [19]. We will freely take advantage of this model and use our assumption of a hierarchy of Grothendieck universes to equip it with an infinite hierarchy of cumulative universes [21]. We refer to a family of presheaves as small if it is classified by a universe.

Given $\mu : n \to m$, we can specify the formation and introduction rules of $\mu : \vdash \cdot$ with another commuting square:

$$F(\mu)^* T^*_n \to T^*_m \quad (2)$$

Unlike dependent products, modal types do not have a universal property—an $\eta$ law—so they cannot be encoded by a single pullback. Instead we must describe the elimination principle separately. However, this principle is rather complex and, since we will soon be in a position to present a much simpler account of modal elimination, we refer the reader to Grätzer et al. [15, Section 5] for the full definition.

As models of a particular GAT, models of MTT assemble into a category. A morphism between models $F$ and $G$ is given by a 2-natural transformation $F \to G$ along with natural assignments of terms and types of $F$ to the terms and types of $G$. All of these operations are required to strictly preserve term, type, and context formers. We refer the reader to Grätzer et al. [15, Section 5] for a precise description.

Finally, a standard result of GATs is that the syntactic model occupies a distinguished place in the category of models:

**Theorem 3.3.** Syntax is the initial model of MTT.

3.2 MTT cosmoi

As mentioned in Section 1, normalization is proven through the construction of a model of MTT together with a map from this model to syntax. Models of MTT and morphisms between them are difficult to construct, however, because of the extreme strictness of morphisms and the requirement that each $\tau_m$ be a representable natural transformation. Prior to normalization, therefore, we introduce a weakened notion of model: an MTT cosmos. An MTT cosmos is an axiomatization of a natural model of MTT, but rather than working in presheaf topos and requiring that $\tau_m$ is a representable natural transformation a cosmos requires only that $\tau_m$ be a morphism in a locally cartesian closed category equipped with structure such as Diagrams 1 and 2.

**Definition 3.4.** A cosmos is a pseudofunctor $F : M \to \text{Cat}$ such that each $F(m)$ is a locally cartesian closed category and each $F(\mu)$ has a left adjoint $F(\mu) \dashv F(\mu)^*$.

**Example 3.** A model of MTT $F$ assembles into a cosmos $G$ by taking $G(m) = \text{PSh}(F(m))$ and $G(\mu) = F(\mu)^*$. In particular, we write $S : M \to \text{Cat}$ for the cosmos induced by the initial model of MTT specified by Theorem 3.3.

The additional requirements imposed by natural models of MTT to encode various connectives can be transferred mutatis mutandis to a cosmos; they are all stated within the language of locally cartesian closed categories. For instance, we model the formation and introduction rules of modal type by requiring the following diagram:

$$F(\mu)(T^*_n) \to T^*_m \quad (3)$$

**Definition 3.5.** An MTT cosmos $F$ is a cosmos equipped with the choice of morphism $\tau_m : T^*_m \to T_m : F(m)$ for each $m : M$ along with choices of diagrams à la Diagram 3 to
close each $\tau_m$ under dependent products, sums, booleans, identity types, a weak universe, and modal types.

**Definition 3.6.** A morphism of cosmoi $\alpha : F \to G$ is a 2-natural transformation such that each naturality square satisfies Beck-Chevalley and each $\alpha_m$ is an LCC functor.

**Definition 3.7.** A morphism of MTT cosmoi $\alpha : F \to G$ is a morphism of cosmoi such that $\alpha_m$ strictly preserves $\tau_m$ and all connectives.

A morphism of MTT cosmoi is both more and less restrictive than a morphism of MTT models. While a morphism of models need not induce an LCC functor between the relevant presheaf categories, a morphism of cosmoi is not required to strictly preserve context extension or the choice of terminal context. It so happens that the only map of consequence in this proof is locally cartesian closed, so the additional structure of morphisms of cosmoi poses no issue. Not requiring the strict preservation of context extension and dropping the representability requirements from MTT cosmoi, however, ensures that cosmoi are far easier to construct.

Merely defining a normalization cosmos $G$ and projection $\pi : G \to S$, however, is not enough to prove normalization; we also need a section to $\pi$. In the category of models, this section would exist as a consequence of initiality, but $S$ is not initial in the category of MTT cosmoi. Accordingly, we cannot easily obtain a section of a map into $S$ and in fact sections rarely exist. Any such map, however, is surjective on definable terms and this ‘quasi-projectivity’ is sufficient:

**Theorem 3.8.** Fix an MTT cosmos $G$ and projection $\pi : G \to S$.

1. For $\Gamma \to \alpha m$, there exists $\Gamma] : G(m)$ and a canonical isomorphism $\alpha : \pi(\Gamma]) = y(\alpha)$.
2. For every $\Gamma \vdash A @ m$, there exists $\Gamma] : A$ such that $\pi([\Gamma]) \circ \alpha = [A]$.
3. For every $\Gamma \vdash M : A @ m$, there exists $[\Gamma] : M$ with $\pi([\Gamma]) \circ \alpha = [M]$.

Here $[-]$ is the isomorphism induced by the Yoneda lemma.

**Remark 4.** Both Theorem 3.3 and Theorem 3.8 are categorical abstractions of rule induction. Indeed, Theorem 3.3 is used to prove Theorem 3.8—via the construction of an appropriate displayed model [23]—and the latter takes the place of rule induction in the proof of normalization (see Theorem 6.3).

### 3.3 Presheaf cosmoi

Example 3 shows that each model of MTT induces an MTT cosmos. In fact, such cosmoi are particularly well-behaved as they are comprised over presheaf topos connected by adjoint triples. These cosmoi enjoy a privileged role in our proof and we observe some of their unique behavior.

$\text{Prod} : (A : Ty_m)(B : Tm_m(A) \to Ty_m) \to Ty_m$

$\alpha_{\text{Prod}} : (A : Ty_m)(B : Tm_m(A) \to Ty_m)$

$\to Tm_m(\text{Prod}(A, B)) \cong \left[\Pi_{a : Tm_m(A)} Tm_m(B(a))\right]$

$\text{Mod}_\mu : (\mu | Ty_n) \to Ty_m$

$\mu : (\mu | A : Ty_n) \to Tm_m(\text{Mod}_\mu(A))$

$\text{letmod}_{\mu,v} : (v \circ \mu | A : Ty_n) \to Tm_m(\text{Mod}_\mu(A))$

$\text{letmod}_{\mu,v} : (v \circ \mu | x : Tm_n) \to Tm_n(B(m_\mu(A, x)))$

$\text{letmod}_{\mu,v} : (v | a : Tm_m(\text{Mod}_\mu(A)))$

$\to Tm_n(B(a))$

$\text{Mod} / \beta_{\mu,v} : (\cdots) \to \text{letmod}_{\mu,v}(A, B, b, \mu,(A, a)) = b(a)$

**Figure 1.** Dependent products and modal types, internally

**Definition 3.9.** A presheaf cosmos $F$ is a cosmos where each $F(m)$ is a presheaf topos and each right adjoint $F(\mu)$ sends small families to small families.

What distinguishes presheaf cosmoi from other cosmoi is the rich internal language they offer. Gratzer et al. [15] have proven that such a cosmos $F$ supports a model of extensional MTT with the same mode theory where $\mu$ is interpreted by $F(\mu)$. We will now use extensional MTT as a multimodal metalanguage to specify the structure of an MTT cosmos as a sequence of constants, thereby reducing its construction to a series of programming exercises.

Some caution is required here, as a presheaf cosmos will frequently host more than one interpretation of MTT. A presheaf cosmos is always equipped with this modal metalanguage (extensional MTT) which can then be used to specify a model of (intensional) MTT. This is comparable to Diagram 1, where type theory is used to describe a model of type theory.

Within this internal language, the universe $\tau_m : Tm^\ast \to Tm$ is encoded by a pair of small types $\to Ty_m : U_0 @ m$ and $A : Ty_m \to Tm_m(A) : U_0 @ m$. Translating Diagram 3 into this language yields the following constants:

$\text{Mod}_\mu : (\mu | Ty_n) \to Ty_m$

$\mu : (\mu | A : Ty_n) \to Tm_m(\text{Mod}_\mu(A))$

$\text{Mod} / \beta_{\mu,v} : (\cdots) \to \text{letmod}_{\mu,v}(A, B, b, \mu,(A, a)) = b(a)$

**Figure 1** contains the complete specification of modal types—including modal elimination—and dependent products. The remaining connectives are detailed in Appendix C.

### 4 Multimodal synthetic Tait computability

In light of Section 3, we revise the proof outlined in Section 1: instead of constructing a glued model of MTT, we will construct a glued MTT cosmos. In fact, we will construct a glued presheaf cosmos, and take advantage of the internal language discussed in Section 3.3 to upgrade it to an MTT cosmos with a projection onto $S$. Prior to this, however, we must show

---

2-monad theory [16, 24] yields an initial cosmos $I$ but we work with $S$ because—unlike $I$—it is known to adequately represent syntax.
that (1) a pair of cosmoi can be glued together and (2) that each mode of the internal language of the resulting cosmos can be extended with synthetic Tait computability primitives compatible with the already-present MTT modalities.

### 4.1 Synthetic Tait computability

For this subsection, fix two presheaf topoi \( E \) and \( F \) along with a continuous functor \( \rho : E \to F \).

**Definition 4.1.** The Artin gluing \( \text{Gl}(\rho) \) is a category whose objects are triples \( (E : E, F : F, \rho : \rho(E) \to \rho(F)) \) while morphisms are commuting squares:

\[
\begin{array}{ccc}
E_0 & \xrightarrow{f_0} & E_1 \\
\downarrow & & \downarrow \\
\rho(E_0) & \xrightarrow{\rho(f_1)} & \rho(E_1)
\end{array}
\]

Projection induces functors \( \pi_0 : \text{Gl}(\rho) \to E \) and \( \pi_1 : \text{Gl}(\rho) \to F \).

**Example 5.** Intuitively \( \text{Gl}(\rho) \) is a category of proof-relevant \( F \)-predicates on \( \rho \)-elements of \( E \). To cultivate this intuition, consider \( F = \text{Set} \) and \( \rho = [1, -] \). An object of \( \text{Gl}(\{1, -\}) \) is a triple of \( (S, E, f) \) which induces a proof-relevant predicate \( \Phi(e) = f^{-1}(e) \) on the global points of \( E \). Following Tait [38], we refer to elements in the image of \( f \) as computable elements. Morphisms are then morphisms of \( E \) equipped with additional structure ensuring that computable elements are sent to computable elements.

We now reap the first reward from considering proof-relevant predicates: \( \text{Gl}(\rho) \) is extremely well-behaved.

**Theorem 4.2** (Artin et al. [5], Carboni and Johnstone [10]). \( \text{Gl}(\rho) \) is a presheaf topos and \( \pi_0 \) is a LCC functor with left and right adjoints.

As a presheaf topos, \( \text{Gl}(\rho) \) enjoys a model of extensional type theory with a strictly cumulative hierarchy of universes and a universe of propositions \( \Omega \). We can use this language to synthetically build logical relations models [36]. In order to effectively construct such models, however, we must supplement type theory with primitives specific to \( \text{Gl}(\rho) \). The most fundamental of these is a proposition:

**Definition 4.3.** The syntactic proposition \( \text{syn} : \Omega \) is interpreted in \( \text{Gl}(\rho) \) as the subterminal object \( (1_E, 0_F, !) \).

Recalling the correspondence between objects of \( \text{Gl}(\rho) \) and predicates, \( \text{syn} \) is the predicate on \( 1_E \) with no computable elements. What makes this proposition useful is its ability to wipe out the obligation to track computable elements. A morphism \( f : \text{syn} \times A \to B \) must contain a morphism \( \rho_0(f) : \rho_0(\text{syn} \times A) \cong \rho_0(A) \to \rho_0(B) \), but there are no computable elements of \( \text{syn} \times A \) so \( \rho_0(f) \) entirely determines \( f \); there is a bijection \( [\text{syn} \times A, B]_{\text{Gl}(\rho)} \cong [\rho_0(A), \pi_0(B)]_E \).

Internally, hypothesizing \( \text{syn} \) collapses the category to \( E \):

\[
\text{syn} \times A \xrightarrow{\sim} A \\
\downarrow \quad \quad \downarrow \\
\text{syn} \quad \quad \bullet A
\]

**Lemma 4.4.** There is an equivalence \( E \cong \text{Gl}(\rho)/\text{syn} \).

In topos-theoretic terms, \( E \) is an open subtopos of \( \text{Gl}(\rho) \). As an open subtopos, we can present \( E \) internally to \( \text{Gl}(\rho) \) through a lex idempotent monad \( \circ A = \text{syn} \to A \). This modality has a strictly disjoint lex idempotent modality, \( \bullet A \) [30, Section 3.4]. While we could work with \( \bullet \) entirely through this characterization, it is helpful to fix a definition:

\[
\begin{array}{ccc}
\text{syn} \times A & \to & A \\
\downarrow & & \downarrow \\
\text{syn} & \to & \bullet A
\end{array}
\]

Intuitively, \( \bullet A \) is the portion of \( A \) with a trivial \( E \)-component. This is even clearer if one calculates the behavior of \( \bullet \) on a closed type \( A = (E, F, f) \) as \( \bullet A = (1, F, !) \). Just as hypothesizing \( \text{syn} \) i.e., working under \( \circ \) recovers \( E \) internally to \( \text{Gl}(\rho) \), working under \( \bullet \) recovers \( F \). Phrased in topos-theoretic terms, \( F \) is a closed subtopos of \( \text{Gl}(\rho) \).

The final ingredient we must add to our type theory is the **realignment axiom** [7, 28, 36], stating that the following canonical map has an inverse \( \text{re} \) for any \( B : U \):

\[
\left( \sum_{A : U} [A \equiv B] \right) \to \left( \sum_{A : \text{syn} \to U} \prod_{z : \text{syn} A(z) \equiv B} B \right)
\]

Unfolding these conditions yields the following:

**Definition 4.5.** Fix \( B : U, A : \circ U \), and \( \alpha : \prod_{z : \text{syn} A(z) \equiv B} B \). The realignment \( \text{re}(B, A, \alpha) \) of \( B \) along \( \alpha \) is a term of type \( \sum_{A : U} A^* \equiv B \) satisfying the following condition:

\[
\prod_{z : \text{syn} A(z)} \text{re}(B, A, \alpha) = (A(z), \alpha(z))
\]

More intuitively, realignment states that a predicate lying over an object in \( E \) can be shifted to lie over an isomorphic object. A proper motivation of realignment is deferred to its use in Section 5, but broadly realignment will be used to satisfy the strict equalities demanded by **Definition 3.6** where a priori two constants might agree only up to isomorphism.

Orton and Pitts [28, Theorem 8.4] show that a Hofmann–Streicher universe satisfies realignment for levelwise decidable propositions. Using the presentation of \( \text{Gl}(\rho) \) as a presheaf topos [10], \( \text{syn} \) is clearly levelwise decidable and so realignment at \( \text{syn} \) is constructively valid.

**Definition 4.6.** The language of synthetic Tait computability is extensional type theory with a cumulative hierarchy of universes and a universe of propositions equipped with a distinguished proposition \( \text{syn} : \Omega \) such that each universe satisfies the realignment axiom for \( \text{syn} \).

This subsection is summarized by the following result, which might be termed the ’fundamental lemma’ of STC:

**Theorem 4.7.** \( \text{Gl}(\rho) \) is a model of STC.
4.2 Gluing together cosmoi

While a model in \( \text{Gl}(\rho) \) for a carefully chosen \( \mathcal{E}, \mathcal{F}, \rho \) is sufficient to prove many results of MTT [12] the situation for MTT is more complex. Rather than gluing along a single functor, it is necessary to glue along an entire 2-natural transformation of continuous functors between 2-functors of presheaf topoi. We begin by considering a pair of presheaf cosmoi for the mode theory \( \{ \mu : n \to m \} \) and a 2-natural transformation of continuous functors between them:

\[
\begin{align*}
\mathcal{E}_n & \xrightarrow{\rho_n} \mathcal{F}_n \\
\mathcal{E}_m & \xrightarrow{\rho_m} \mathcal{F}_m \quad (6)
\end{align*}
\]

Let us further assume that \( f \) and \( g \) preserve finite colimits.

Gluing ‘horizontally’, we obtain a pair of categories \( \text{Gl}(\rho_n) \) and \( \text{Gl}(\rho_m) \) and by Theorems 4.2 and 4.7 both are presheaf topoi and models of STC. Artin glue is functorial, and Diagram 6 induce a functor \( \text{Gl}(f, g) : \text{Gl}(\rho_n) \to \text{Gl}(\rho_m) \) sending \( (E_n, F_n, x) \) to \( (f(E_n), g(F_n), g(x)) \).

**Lemma 4.8.** \( \text{Gl}(f, g) : \text{Gl}(\rho_n) \to \text{Gl}(\rho_m) \) is a right adjoint.

**Proof.** While this follows classically from the special adjoint functor theorem, an explicit construction is useful. There is a comparison \( \beta : g \circ \rho_m \to \rho_n \circ f \) induced by transposition and the unit of the \( f \). The left adjoint \( \text{Gl}(f, g)_1 \) sends \( f : F \to \rho_m(E) \) to \( \beta \circ g(f) : g(F) \to \rho_n(f(E)) \).

**Lemma 4.9.** The adjunction \( \text{Gl}(f, g)_1 : \text{Gl}(f, g) \) induces a dependent right adjoint.

As a consequence of Lemma 4.9, we obtain a model of MTT with the mode theory \( \{ \mu : n \to m \} \) which interprets \( n, m, \) and \( \mu \) as \( \text{Gl}(\rho_n), \text{Gl}(\rho_m), \) and \( \text{Gl}(f, g) \) respectively. This model of MTT is particularly well-behaved: equality is extensional and \( \text{Gl}(f, g) \) validates the strong transposition-style elimination rules specified by [8].

**Lemma 4.10.** In this model of MTT, \( \langle \mu | \text{syn}_n \rangle \cong \text{syn}_m \)

**Proof.** Externally, \( \text{syn}_n = (1, 0, !) \) but \( g \) preserves 0 while \( f \) preserves 1, so \( \text{Gl}(f, g)(\text{syn}_n) \cong (1, 0, !) = \text{syn}_m \).

In fact, in this model \( \circ (\mu | A) \cong (\mu | O) \) and \( \bullet (\mu | A) \cong (\mu | \bullet A) \). These isomorphisms are not automatic consequences of Lemma 4.10 but both follow from similar calculations.

**Remark 6.** Technically, \( \text{syn}, \circ, \) and \( \bullet \) should be always annotated with a mode. In light of these results, however, we shall omit this annotation and systematically identify \( \text{syn}_m \) and \( \langle \mu | \text{syn}_n \rangle \). As both are subterminal, there are no coherence issues in this identification.

**Definition 4.11.** The language of multimodal STC (MSTC) is extensional MTT with a cumulative hierarchy of universes and a universe of propositions such that

- Each mode is equipped with a proposition \( \text{syn} \).
- Each universe satisfies the realignment axiom for \( \text{syn} \).
- MTT modalities commute with \( \text{syn}, \circ, \) and \( \bullet \).

Summarizing the preceding discussion:

**Theorem 4.12.** \( \text{Gl}(\rho_n), \text{Gl}(\rho_m), \) and \( \text{Gl}(f, g) \) assemble into a presheaf cosmos and a model of MSTC.

In fact, it is only a small step from this result to the full fundamental lemma of multimodal STC:

**Theorem 4.13.** Given a pair of cosmoi \( F, G : M \to \text{Cat} \) and a 2-natural transformation \( \rho : F \to G \) such that each \( F(\mu), G(\mu) \) preserves finite colimits and each \( \rho_m \) is continuous, \( \text{Gl}(\rho) : M \to \text{Cat} \) both a presheaf cosmos and a model of MSTC. Furthermore \( \pi_0 : \text{Gl}(\rho) \to F \) is a morphism of cosmoi.

5 The normalization cosmos

Recall from Section 2.3 the 2-functor of categories of renamings \( \text{Ren} \). By an identical construction to Example 3, we obtain the cosmos of renamings \( \mathcal{R}(-) = \text{PSh}(\text{Ren}) \) and the 2-natural transformation \( i[-] : \text{Ren} \to \text{Cat} \) acts by precomposition to yield a 2-natural transformation \( i[-] : \mathcal{S} \to \mathcal{R} \).

**Theorem 4.13 then yields the following:**

**Definition 5.1.** The normalization cosmos \( \mathcal{G} \) is a presheaf cosmos and model of MSTC where \( \mathcal{G}(m) = \text{Gl}(i[m]) \).

As a further consequence of Theorem 4.13, the projection map \( \pi_0 : \mathcal{G} \to \mathcal{S} \) is a morphism of cosmoi. In this section, we equip \( \mathcal{G} \) with the structure of an MTT cosmos and show that \( \pi_0 \) extends to a morphism of MTT cosmoi.

5.1 Prerequisites for the normalization cosmos

Before we extend \( \mathcal{G} \) to an MTT cosmos, we import features of \( \mathcal{G} \) into the language of MSTC to specialize the latter to this situation. In this section, we begin using the interpretation of MTT to work internally to \( \mathcal{G} \) and explicitly record the extensions to MSTC required for the normalization proof.

**Notation 5.2** (Dependent open modality). As \( \circ A = \text{syn} \to A \), we will write \( \circ A(z) = (z : \text{syn}) \to A(z) \) for the dependent version of the open modality.

**Notation 5.3** (Extension types). Given a type \( A \), a proposition \( \phi \), and an element \( a : \phi \to A \), we write \( \{ A | x : \phi \mapsto a(x) \} \) for subtype of \( A \) of elements equal to \( a \) under \( \phi \). Formally:

\[ \{ A | x : \phi \mapsto a(x) \} = \sum_{a : A} (x : \phi) \to a' = a(x) \]

We treat the coercion \( \{ A | x : \phi \mapsto a(x) \} \to A \) as silent and refer to the equation \( a' = a(x) \) as a boundary condition.

Recall from Example 3 that \( \mathcal{S} \) already contains the structure of an MTT cosmos. As a presheaf cosmos, this manifests through a series of constants in the internal language of \( \mathcal{S} \). Using Lemma 4.4 we import these constants into \( \mathcal{G} \).
Extension 1. For each \( m \in \mathcal{M} \), there is a pair of constants \( z : \text{syn} \vdash \mathcal{T}_m(z) : U_0 \circ m \) and \( z : \text{syn} \vdash \mathcal{T}_m(z) : \mathcal{T}_m(z, A) \). These constants are further equipped with operations à la Fig. 1 closing them under dependent sums, dependent products, modal types, etc.

Next, observe that normals, neutrals, and normal types are equipped with an action by renamings, so that they can be structured as presheaves over \( \text{Ren} \). The decoding operations further organize them into proof-relevant predicates over terms and types e.g., the presheaf of normal types as an object of \( G \) lying over the presheaf of types from \( S(m) \). In fact, because renamings map variables to variables, the collection of variables of a given type organizes into a presheaf over \( \text{Ren} \) and part of an object in \( G \). We import these objects into the internal language as additional constants:

**Extension 2.** Given \( m \in \mathcal{M} \) and \( A : \mathcal{O}_z \mathcal{T}_m(z) \), we have constants \( \text{Nf}_m(A), \text{Ne}_m(A), \mathcal{V}_m(A) : \{ U_0 | z : \text{syn} \mapsto \mathcal{T}_m(z, A(z)) \} \) and \( \text{NfTy}_m : \{ U_0 | z : \text{syn} \mapsto \mathcal{T}_m(z) \} \).

We treat the coercion from \( \mathcal{V}_m(A) \) to \( \text{Ne}_m(A) \) as silent.

**Notation 5.4.** We frequently omit explicitly passing \( z : \text{syn} \) as an argument to \( \mathcal{O}_z \). For instance, given \( A, B : \mathcal{O}_z \mathcal{T}_m(z) \) we write \( \text{Nf}_m(\mathcal{P}_m(A, B)) \) not \( \text{Nf}_m(\lambda z. \mathcal{P}_m(z, A(z))) \).

The normals and neutrals themselves lift to constants of type \( \text{Nf}_m(A), \text{Ne}_m(A), \) and \( \text{NfTy}_m \) using a form of higher-order abstract syntax [20]. For example, the constants for dependent products and modal types are presented in Fig. 2. These operations collapse to the corresponding syntactic constants specified by Extension 1 under \( z : \text{syn} \)—recall from Extension 2 that here e.g. \( \text{Nf}_m(A) = \mathcal{T}_m(z, A) \). The full collection of constants is specified in Appendix D.

**Extension 3.** There are constants internalizing normals, neutrals, and normal types.

Finally, inspecting Definition 5.1 reveals that modalities are interpreted by functors which are both left and right adjoints. As a result, modalities preserve coproducts:

**Extension 4.** \( \langle \mu \mid A + B \rangle \cong \langle \mu \mid A \rangle + \langle \mu \mid B \rangle \).

5.2 The MTT cosmos
We now extend \( \mathcal{G} \) to an MTT cosmos. To ensure that \( \pi_0 \) induces a morphism of MTT cosmoi, it suffices to ensure that each constant we add to \( \mathcal{G} \) is equal to the corresponding piece of \( S \) as internalized by Extension 1 under \( z : \text{syn} \).

The universe of computable types and terms. We begin with the definition of types and terms in this cosmos. Concretely, we require the following for each \( m \in \mathcal{M} \):

\[
\mathcal{T}_m^* : \{ U_2 | z : \text{syn} \mapsto \mathcal{T}_m(z) \}
\]

\[
A : \mathcal{T}_m^* + \mathcal{T}_m^*(A) : \{ U_1 | z : \text{syn} \mapsto \mathcal{T}_m(z, A) \}
\]

**Prod:** \( \langle A : \text{NfTy}_m \rangle (B : \mathcal{V}_m(A)) \rightarrow \text{NfTy}_m \rightarrow \text{NfTy}_m \)

**Mod:** \( \langle \mu | \text{NfTy}_m \rangle \rightarrow \text{NfTy}_m \)

\[
\text{lam} : \langle \mu | \mathcal{O}_z \mathcal{T}_m(z) \rangle (B : \langle \mu | \mathcal{O}_z \mathcal{T}_m(A) \rangle) \rightarrow \mathcal{O}_z \mathcal{T}_m(z)
\]

\[
\text{app} : \langle \mu | \mathcal{O}_z \mathcal{T}_m(z) \rangle (B : \langle \mu | \mathcal{O}_z \mathcal{T}_m(A) \rangle) \rightarrow \mathcal{O}_z \mathcal{T}_m(z)
\]

**Figure 2.** Neutral and normal forms, internally

We start with the following putative definition of types:

**record** \( T : U_2 \) where

\[
\text{code} : \text{NfTy}_m
\]

\[
\text{pred} : \{ U_1 | z : \text{syn} \mapsto \mathcal{T}_m(z, \text{code}) \}
\]

\[
\text{reflect} : \{ \text{Ne}_m(\text{code}) \rightarrow \text{pred} \mid \text{syn} \mapsto \text{id} \}
\]

\[
\text{reify} : \{ \text{pred} \rightarrow \text{Nf}(\text{code}) \mid \text{syn} \mapsto \text{id} \}
\]

In prose, \( A : T \) contains the code of a normal type \( A.\text{code} \) as well as a proof-relevant predicate on the elements of \( A.\text{code} \). The last two fields ensure that (1) all elements tracked by this predicate can be assigned normal forms, and (2) all neutrals lie within the predicate. We write \( \downarrow_A \) and \( \uparrow_A \) for \( A.\text{reify} \) and \( A.\text{reflect} \). Of the two, the reify is the crucial operation needed for the normalization algorithm: it ensures that computable elements can be given normal forms. Tait [38], however, has shown that the pair of operations is necessary to close all type formers under just reify.

We cannot simply define \( \mathcal{T}_m^* = T \), as \( T \) does not satisfy the equation \( \langle z : \text{syn} \rangle + T = \mathcal{T}_m(z) \). It does, however, satisfy this condition up to isomorphism: under \( z : \text{syn} \), the types of pred, reflect, and reify collapse to singletons, while the type of code collapses to \( \mathcal{T}_m(z) \) by Extension 2:

\[
\alpha(z, A) = A.\text{code} : \prod x \mathcal{S}(z) \cong \mathcal{T}_m(z)
\]

Observe \( (\mathcal{T}_m, \alpha) : \sum \mathcal{O} \cup \prod x \mathcal{S}(z) \cong T \), so the re-alignment axiom of Definition 4.5 applies and we can define

\[
(\mathcal{T}_m, \alpha) : \text{re}(T, \mathcal{T}_m, \alpha) \cong \mathcal{T}_m(z)
\]

The equation \( z : \text{syn} + \mathcal{T}_m = \mathcal{T}_m(z) \) follows immediately from the second half of Definition 4.5. On elements \( A : \mathcal{T}_m^* \), this implies \( z : \text{syn} + A = \alpha(A) \). For readability, we continue to use record notation to manipulate \( \mathcal{T}_m^* \).

Given \( A : \mathcal{T}_m^* \), we define \( \mathcal{T}_m^*(A) \):

\[
\mathcal{T}_m^*(A) = A.\text{pred} : \{ U_1 | z : \text{syn} \mapsto \mathcal{T}_m(z, A) \}
\]
To see that this is well-typed, we must show $\text{Tm}_m^\ast(A) = \text{Tm}_m(z, A)$ given $z : \text{syn}$. The type of $A$ code in Construction 7 ensures $\text{Tm}_m^\ast(A) = \text{Tm}_m(z, A).$ We have observed that $A = A$.code under $z : \text{syn}$ so $\text{Tm}_m(A) = \text{Tm}_m(z, A)$.

Type connectives. It remains only to close $(\text{Tm}_m^\ast, \text{Tm}_m^\ast)$ under all connectives. For mode-local connectives, these constructions are identical to those given in Sterling [34]. Accordingly, we detail only dependent products as a representative mode-local connective along with modal types. The remaining connectives are discussed in Appendix E.

Lemma 5.5. $(\text{Tm}_m^\ast, \text{Tm}_m^\ast)$ is closed under dependent products.

Proof. We must define two constants:

$$\text{Prod}^\ast : (A : \text{Ty}_m^\ast)(B : \text{Tm}_m^\ast(A) \rightarrow \text{Ty}_m^\ast) \rightarrow \text{Ty}_m^\ast$$

$$\alpha_{\text{Prod}^\ast} : (A : \text{Ty}_m^\ast)(B : \text{Tm}_m^\ast(A) \rightarrow \text{Ty}_m^\ast) \rightarrow \text{Tm}_m^\ast(\text{Prod}^\ast(A, B)) \equiv \left[ \prod_{z : \text{Tm}_m^\ast(A)} \text{Tm}_m^\ast(B)(a) \right]$$

Additionally, we must show that if $z : \text{syn}$ then $\text{Prod}^\ast = \text{Prod}(z)$ and $\alpha_{\text{Prod}^\ast} = \alpha_{\text{Prod}}(z)$.

We begin by fixing $A : \text{Tm}_m^\ast$ and $B : \text{Tm}_m^\ast(A) \rightarrow \text{Ty}_m^\ast$. Define $\Phi = \prod_{z : \text{Tm}_m^\ast(A)} \text{Tm}_m^\ast(B)(a)$ and observe under $z : \text{syn}$ that $\Phi = \prod_{z : \text{Tm}_m^\ast(A, B)} \text{Tm}_m^\ast(B(z, a))$ and therefore

$$\alpha_{\text{Prod}^\ast}(z : \text{Tm}_m(z, A, B)) \equiv \Phi$$

We now realign $\Phi$ along this isomorphism to obtain a type $\Psi$ and isomorphism $\beta : \Psi \equiv \Phi$. Under $z : \text{syn}$ these restrict to $\text{Tm}_m(z, \text{Prod}(z, A, B))$ and $\alpha_{\text{Prod}^\ast}(z)$ respectively. With these to hand we define $\text{Prod}^\ast$ and $\alpha_{\text{Prod}^\ast}$:

$$\text{Prod}^\ast(A, B).\text{code} = \text{Prod}(A.\text{code}, \lambda v. B(\downarrow_A v).\text{code})$$

$$\text{Prod}^\ast(A, B).\text{pred} = \Psi$$

$$\text{Prod}^\ast(A, B).\text{reflect} = \lambda e. \beta^{-1}(\lambda a. \text{app}(e, \lambda a. B(\downarrow_A a)))$$

$$\text{Prod}^\ast(A, B).\text{reify} = \beta f. \lambda m. \lambda v. \text{app}(e, \downarrow_A v)(\beta(f)(\downarrow_A v))$$

$$\alpha_{\text{Prod}^\ast} = \beta$$

It remains to check a variety of boundary conditions under $z : \text{syn}$. In particular, we must show that $\text{Prod}^\ast(A, B) = \text{Prod}(z, A, B)$ and that reflect and reify become the identity. These follow directly from assumptions about $A, B$, and the boundaries of various constructors. For instance

$$\text{Prod}^\ast(A, B) = \text{Prod}^\ast(A, B).\text{code}$$

$$= \text{Prod}(A.\text{code}, \lambda v. B(\downarrow_A v).\text{code})$$

$$= \text{Prod}(z, A.\text{code}, \lambda v. B(\downarrow_A v))$$

$$= \text{Prod}(z, A, B(\downarrow_A v))$$

$$= \text{Prod}(z, A, B) \square$$

Lemma 5.6. $(\text{Tm}_m^\ast, \text{Tm}_m^\ast)$ is closed under modal types.

Proof. Fix a modality $\mu : n \rightarrow m$. In this case we define the four constants specified by Fig. 1, subject to the expected boundary conditions. Fix a variable $A : \text{Ty}_m^\ast$ under the modal annotation $\mu$ i.e., $(\mu \mid A : \text{Ty}_m^\ast)$. We define the unaligned predicate as follows:

$$\text{record } \Phi \triangleq U_1\text{ where }$$

$$\text{tm : } \text{Nf}_m(\text{Mod}_\mu(A))$$

$$\text{prf : } \bullet \left( \sum_{e : N_f_m(\text{Mod}_\mu(A))} \text{tm} = \text{up}(e) \right.$$

$$\left. + \sum_{a : (\mu \mid A.\text{pred})} \text{tm} = \text{mod}_\mu(\downarrow_A a) \right)$$

For the first time, we have used the closed modality $\bullet o$ to explicitly tweak the proof-relevant predicate. Intuitively, $\Phi$ is a predicate on $\text{Tm}_m(z, \text{Mod}_\mu(z, A))$ and $\text{tm}$ ensures that this predicate tracks elements with normal forms. The second field, moreover, ensures that these normal are either neutral or $\mu$ where $a$ is computable. Without the closed modality shielding the second field of $\Phi$, however, this could never have the correct extent along $z : \text{syn}$. Using $\bigcirc o X = 1$ and the boundary of $\text{Nf}_m(\text{Mod}_\mu(A))$, we can now define the following isomorphism:

$$\alpha_{\circ}(z, p) = p.\text{tm} : \prod_{z \text{syn}} \Phi \equiv \text{Tm}_m(z, \text{Mod}_\mu(z, A))$$

We now define $\text{Mod}_\mu^\ast$:

$$\text{Mod}_\mu^\ast(A).\text{code} = \text{Mod}_\mu(A.\text{code})$$

$$\text{Mod}_\mu^\ast(A).\text{pred} = \Psi$$

$$\text{Mod}_\mu^\ast(A).\text{reflect} = \lambda e. \alpha_{\circ}(\uparrow(e), \eta_1(e, *))$$

$$\text{Mod}_\mu^\ast(A).\text{reify} = \lambda m. \alpha(m).\text{tm}$$

Unlike Lemma 5.5, the introduction and elimination principles are not automatically obtained from $\alpha$ and they must be constructed separately:

$$\text{m}_\mu^\ast(A, a) = \alpha_{\circ}(\downarrow_A a, \eta_2(a, *))$$

It remains to define the elimination principle $\text{letmod}_\mu^\ast$. This is an involved affair and we describe it step-by-step. Begin by fixing $v : m \rightarrow o$ along with the following:

$$B : (v \mid \text{Tm}_m(\text{Mod}_\mu(A))) \rightarrow \text{Ty}_o$$

$$b : (v \circ \mu \mid x : \text{Tm}_m^\ast(A)) \rightarrow \text{Tm}_m^\ast(B(m_\mu^\ast(A, x)))$$

$$(v \mid m : \text{Tm}_m^\ast(\text{Mod}_\mu(A)))$$

We must construct an element of $\text{Tm}_m^\ast(B(a))$. We begin by inspecting $m$. As MTT modalities in extensional MTT commute with dependent sums, equality, $\bullet o$, and by Extension 4—with finite coproducts, $m$ can be decomposed into the following:

$$(v \mid \text{tm} : \text{Nf}_m(\text{Mod}_\mu(A)))$$

$$\text{prf : } \bullet \left( \sum_{e : \langle v \mid \text{Nm}_m(\text{Mod}_\mu(A)) \rangle} \text{tm} = \text{up}(e) \right.$$

$$\left. + \sum_{a : \langle \mu \mid A.\text{pred} \rangle} \text{tm} = \text{mod}_\mu(\downarrow_A a) \otimes a \right)$$

Recall from Diagram 4 that $\bullet o X$ is a pushout of syn and $X$. To define a map out of $\bullet o X$, therefore, it suffices to define
a map out of $X$ which is constant assuming $z : \text{syn}$. We conclude by scrutinizing prf:

$$
\begin{align*}
\{ t_1 \text{ mod } v(e) \_ \_ \_ & \mapsto \uparrow \text{letmod}_{\mu,v}(A, \lambda w. B(\uparrow v) . \text{code}, \lambda x. \downarrow b(\uparrow x), e) \\
\{ t_2 \text{ mod } v(a) \_ \_ \_ & \mapsto b(a)
\end{align*}
$$

Given $z : \text{syn}$, both branches collapse to letmod$_{\mu,v}(z, A, B, b, a)$ so this yields a well-defined map. We omit the routine computations of boundary conditions for space reasons. $\square$

Theorem 5.7. $G$ supports an MTT cosmos built around $(Ty_m, \text{Tr}_m)$ and $\pi_0 : G \rightarrow S$ is a map of MTT cosmos.

6 Normalization

After Theorem 5.7, it remains only to parlay the existence of the normalization cosmos into a normalization function.

6.1 The normalization function

At this point, it becomes necessary to shift from working purely internally to $G$ to inspecting some constructions externally. Accordingly, we will have use for the total spaces of terms and normal forms e.g. $\text{Tr}_m = \sum_{A : \text{Ty}_m} \text{Tr}_m^*(A)$. We write $\text{Tr}_m$ and $\text{Tr}_m^*$ for the presheaves of types and terms in $S(m)$ to disambiguate them from $\text{Ty}_m$ and $\text{Tr}_m^*$.

Lemma 6.1. There is a morphism $\downarrow : \text{Tr}_m \rightarrow \text{Nf}_m$ which restricts to id under $\text{syn}$.

Proof. Working internally, $\downarrow(A, M) = (A, \downarrow_A M)$. $\square$

Fix a term $\Gamma \vdash M : A @ m$. Theorems 3.8 and 5.7 define a map $\llbracket M \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \text{Tr}_m^*$ in $G(m)$ along with an isomorphism $\alpha : \pi_0(\llbracket \Gamma \rrbracket) \cong y(\Gamma)$ such that $\pi_0(\llbracket M \rrbracket) = [M] \circ \alpha$.

We would like to obtain a normal form for $M$ from $\llbracket M \rrbracket$. To this end, we can unfold $\llbracket M \rrbracket$ along with $\downarrow$ from Lemma 6.1 to obtain a commuting diagram:

$$
\begin{array}{cccc}
\pi_1(\llbracket \Gamma \rrbracket) & \rightarrow & \pi_1(\text{Tr}_m^*) & \rightarrow & \pi_1(\text{Nf}_m) \\
\alpha \circ \llbracket \Gamma \rrbracket & \downarrow & i[m]^*(y(\Gamma)) & \downarrow & i[m]^*(\llbracket T_m \rrbracket) \\
i[m]^*(\llbracket M \rrbracket) & \rightarrow & i[m]^*(\llbracket \text{Nf}_m \rrbracket)
\end{array}
$$

To normalize $M$, it suffice to construct atoms$_G : \pi_1(\llbracket \Gamma \rrbracket)_G$ such that $\alpha(\llbracket \Gamma \rrbracket(\text{atoms}_G)) = \text{id} : i[m]^*(y(\Gamma))_G$: pushing atoms$_G$ along the top of the diagram would yield a normal form (an element of $\pi_1(\text{Nf}_m)$) which decodes to $M$ by Yoneda.

Lemma 6.2. For any $\Gamma \vdash X @ m$ there exists atoms$_G : \pi_1(\llbracket \Gamma \rrbracket)_G$ lying over id : $i[m]^*(y(\Gamma))$.

Proof. This proof proceeds by induction on $\Gamma$.

Case. $\Gamma = 1$

Here $\llbracket \Gamma \rrbracket$ is terminal, so atoms$_1$ is its unique element.

Case. $\Gamma = \Delta(\mu : A)$

In this case $\llbracket \Gamma \rrbracket = \llbracket \Delta \rrbracket \times_{G(\mu)(\text{Tr}_m^*)} G(\mu)(\text{Tr}_m^*)$. First, we reindex atoms$_A$ by $\Gamma \vdash \uparrow : \Delta @ m$ to obtain $\delta \in \llbracket A \rrbracket$. Next, using the element $v_0 \in G(\mu)(\text{Ne}_n(A))_G$ we define atoms$_G = (\delta, \uparrow v_0)$.

Case. $\Gamma = \Delta(\mu : A)$

We define atoms$G = G(\mu)(\text{atoms}_A)$. $\square$

Remark 7. atoms$_G$ is analogous to the initial environment used in classical NbE proofs to kick off normalization. Abel [1], for instance, denotes the environment $\Gamma^\downarrow$.

Combining Lemma 6.2 with the argument above, we conclude that for term $\Gamma \vdash M : A @ m$, there exists $\Gamma \rightarrow u : A @ m$ such that $|u| = M$. Moreover, because we have consistently worked with equivalences class of terms, this function automatically respects definitional equality. Summarizing:

Theorem 6.3. There is a function $\text{nft}_G(\_ , A)$ sending terms of type $\Gamma \vdash A @ m$ to normal forms such that

1. If $\Gamma \vdash M : A @ m$ then $\Gamma \vdash \text{nft}_G(\Gamma, A) = M : A @ m$.
2. If $\Gamma \vdash M = N : A @ m$ then $\text{nft}_G(\Gamma, A) = \text{nft}_G(\Gamma, N, A)$.

We can repeat this process to normalize types instead of terms. Given $\Gamma \vdash A @ m$, we obtain $\llbracket A \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \text{Nf}_m$ which unfolds to an analogous diagram with only a small change: rather than using $\uparrow$ to pass from atoms$_G$ to normal forms, we use code to shift from $\text{Tr}_m^*$ to normal types:

$$
\begin{array}{cccc}
\pi_1(\llbracket \Gamma \rrbracket) & \rightarrow & \pi_1(\text{Tr}_m^*) & \rightarrow & \pi_1(\text{Nf}_m) \\
\alpha \circ \llbracket \Gamma \rrbracket & \downarrow & i[m]^*(y(\Gamma)) & \downarrow & i[m]^*(\llbracket T_m \rrbracket) \\
i[m]^*(\llbracket M \rrbracket) & \rightarrow & i[m]^*(\llbracket \text{Nf}_m \rrbracket)
\end{array}
$$

By again pushing atoms$_G$ along the top of this diagram, we obtain a normalization function for types.

Theorem 6.4. There is a function $\text{nf}_G(\_ , A)$ sending types to normal types such that

1. If $\Gamma \vdash A @ m$ then $\Gamma \vdash \text{nf}_G(\Gamma, A) = A @ m$.
2. If $\Gamma \vdash A = B @ m$ then $\text{nf}_G(\Gamma, A) = \text{nf}_G(\Gamma, B)$.

6.2 Corollaries of normalization

A number of important theorems follow as corollaries of Theorems 6.3 and 6.4. For instance, we can reduce the decidability of conversion to the decidability of the mode theory.

Corollary 6.5 (Decidability of conversion).

1. $\Gamma \vdash M = N : A @ m$ iff $\text{nf}_G(\Gamma, M, A) = \text{nf}_G(\Gamma, N, A)$.
2. $\Gamma \vdash A = B @ m$ iff $\text{nf}_G(\Gamma, A) = \text{nf}_G(\Gamma, B)$.

Proof. We show only the proof for this first claim. The ‘only if’ direction is established by the second point of Theorem 6.3. Suppose instead $\text{nf}_G(\Gamma, M, A) = \text{nf}_G(\Gamma, N, A)$, so $\text{nf}_G(\Gamma, M, A) = \text{nf}_G(\Gamma, N, A)$. By the first point of Theorem 6.3, $\text{nf}_G(\Gamma, M, A) = M$ and $\text{nf}_G(\Gamma, N, A) = N$, so the conclusion follows. $\square$
Equality of normal forms and normal types is evidently decidable if equality in \( M \) is decidable, so this proves the promised sharp bound on the decidability of conversion in MTT. While we have not developed a bidirectional syntax for MTT, the fully annotated presentation of its syntax is decidable precisely when conversion is decidable:

**Corollary 6.6.** If \( M \) is decidable, type checking is decidable.

A priori, however, a given term could have multiple normal forms which complicates further analysis. We therefore strengthen Theorem 6.3 with the following:

**Theorem 6.7 (Tightness).**

1. If \( \Gamma \not\vdash u : A @ m \), then \( \text{nft}_\Gamma (|u|, A) = u \).
2. If \( \Gamma \not\vdash \tau @ m \), then \( \text{nft}_\Gamma (|\tau|) = \tau \).

This result is proven by a straightforward inductive argument over neutrals, normal forms, and normal types and by inspecting the relevant portions of the normalization cosmos in each case.

**Corollary 6.8.** Normalization is an isomorphism between equivalence classes of terms (resp. types) and normal forms (resp. normal types).

**Proof.** Corollary 6.5 already shows that normalization is injective and Theorem 6.7 provides a section. ☐

These results imply the injectivity of type constructors, an essential property for implementation.

**Corollary 6.9.** If \( \Gamma \vdash A_0 \rightarrow B_0 = A_1 \rightarrow B_1 @ m \) then \( \Gamma \vdash A_0 = A_1 @ m \) and \( \Gamma . (\text{id} \mid A_0) + B_0 = B_1 @ m \).

**Proof.** Set \( \tau_i = \text{nft}_\Gamma (A_i) \) and \( \sigma_i = \text{nft}_\Gamma ((\text{id} \mid A_i)(B_i)) \). Unfolding definitions shows that \( |\tau_i| \rightarrow |\sigma_i| = |A_i| \rightarrow |B_i| \).

By Corollary 6.8, \( \text{nft}_\Gamma (A_i \rightarrow B_i) = \tau_i \rightarrow \sigma_i \).

Next, we recall that \( \Gamma \vdash A_0 \rightarrow B_0 = A_1 \rightarrow B_1 @ m \) by assumption, so \( \tau_0 \rightarrow \sigma_0 = \tau_1 \rightarrow \sigma_1 \). As an operation on normal forms, however, \( \rightarrow \rightarrow \) is clearly injective, so \( \tau_0 = \tau_1 \) and \( \sigma_0 = \sigma_1 \). The result now follows from Corollary 6.5. ☐

Finally, Gratzer et al. [14] show canonicity for MTT extended with the equality \( 1.\{\mu\} = 1 \). Normalization provides a (heavy-handed) proof of canonicity without this equation:

**Corollary 6.10.** If \( 1.\{\mu\} \vdash M : \text{bool} @ m \) then \( M \in \{\text{tt}, \text{ff}\} \).

7 **Crisp identity induction principles**

In Shulman [32] and Gratzer et al. [15], **crisp** induction principles are a variation of the induction principles for types such as bool or \( \text{Id}_4(a_0, a_1) \) which allow the scrutinee of the induction to occur beneath a modality. Crisp induction principles are derivable in MTT if the modality has an internal right adjoint [15], but they are justified in other situations. In particular, crisp induction for identity types is validated if and only if the canonical map \( \text{Id}_{\{\mu\}(A)} \) is an equivalence:

\[
\text{Id}_{\{\mu\}(A)}(\text{mod}_\mu (M_0), \text{mod}_\mu (M_1)) \rightarrow (\mu | \text{Id}_4(M_0, M_1)) \quad (10)
\]

This is true if \( \text{Id}_4(a_0, a_1) \) supports equality reflection.

This equivalence is often indispensable when reasoning about modal operations in MTT; it plays essentially the same role as function extensionality. Accordingly, it is frequently useful to add the following strengthened version of the induction principle for identity types which implies Eq. (10):

\[
\begin{align*}
\Gamma . (\mu | A). (\mu | A[\tau\downarrow]). (\mu | \text{Id}_4(\tau(v_0), v_0)) + B @ m \\
\Gamma . (\mu | A) + M : B[\tau(v_0), \text{refl}(v_0)] @ m \\
\Gamma . (\mu) + N_0, N_1 : A @ n \quad \Gamma . (\mu) + P : \text{Id}_4(N_0, N_1) @ n \\
\Gamma + J^\mu(B, M, P) : B[\text{id}.N_0, N_1, P] @ m
\end{align*}
\]

**Gluing for modal type theory.** Gratzer et al. [17] gave a classical normalization-by-evaluation proof for a Fitch-style type theory. The complexity of this proof, however, makes it intractable to extend to a general modal type theory like MTT. Unfortunately, extending gluing techniques to modal type theories has proven challenging. In particular, Gratzer et al. [14] used gluing to prove canonicity for MTT, but they were forced to add an additional equality to MTT \( 1.\{\mu\} = 1 \) to tame the construction of the gluing model. The challenge lies in fitting the glued category of contexts into a CwF-style model of type theory; the natural definition of glued types and terms fails to admit modalities. While there have been some attempts to systematize the construction of glued CwF's [22], they do not apply to MTT.

**Synthetic Tait computability.** The introduction of representable map categories [39] and LCCCs [16] for modeling the syntax of (non-modal) type theory offered an alternative approach. Crucially, they show that syntax can be given a
universal property among structured categories with better behavior than CwFs. Sterling and collaborators [34–36] have built on this idea and introduced synthetic Tait computability to prove syntactic metatheorems via gluing together LCCCs rather than CwFs. Unlike other approaches to gluing, STC generalizes well to a multimodal setting and by extending STC to MSTC normalization for MTT becomes tractable.

MTT as a metalanguage. In a parallel line of work, Bocioquet et al. [9] have also used MTT as a metalanguage in the construction of models of type theory. They, however, do not work with a modal object type theory and instead use MTT to internalize a functor \( F \) rather than working internally to \( \text{Gl}(F) \). As a result, while both proofs use MTT modalities, the modalities used by op. cit. are encoded in our proof by fibered lex monads \( (\bigcirc, \bullet) \) which prove easier to manipulate.

9 Conclusions and future work

We prove normalization for MTT (Theorem 6.3) and thereby reduce the decidability of conversion and type checking to the decidability of equality of the underlying mode theory (Corollaries 6.5 and 6.6). In addition, we deduce a number of corollaries from normalization itself, including the injectivity of type constructors and canonicity (Corollaries 6.9 and 6.10).

By working constructively, we have obtained an effective procedure for normalization. This, along with our results on type checking, open the door to a theoretically-sound implementation of MTT generic in the mode theory. In the future, we intend to develop a bidirectional syntax for MTT and implement it. Stassen et al. [33] have made promising initial steps in this direction for poset-enriched mode theories.

Acknowledgments

We are thankful for discussions with Carlo Angiuli, Martin Bidlingmaier, Lars Birkedal, Thierry Coquand, Alex Kavvos, Christian Sattler, and Jonathan Sterling. The author was supported in part by a Villum Investigator grant (no. 25804), Center for Basic Research in Program Verification (CPV), and the VILLUM Foundation.

References


Normalization for Multimodal Type Theory

LICS '22, August 2–5, 2022, Haifa, Israel

Figure 3. Definition of selected normals, neutrals, and normal types

Figure 4. Complete definition of renamings
A Definition of an MTT cosmos

Definition A.1 (Definition 18 [6]). An internal lifting structure \( s : i \vdash \tau \) between a pair of morphisms \( i : A \to B \) and \( \tau : X \to Y \) is a section of canonical map \( X^B \to Y^B \times_{Y^A} X^A \).

Definition A.2. The 2-category \( \Cat_g \) consists of small categories, functors between them, and natural isomorphisms.

Definition A.3. The category of cosmoi \( \cV \) is the locally full 2-subcategory of \( \{ M, \Cat_g \} \) spanned by pseudofunctors \( F \) such that:

- For each \( m : M, F(m) \) is an LCCC
- For each \( \mu : n \to m, F(\mu) \) is a right adjoint.

Note that we do not require that \( F(\mu) \) preserves the LCC of \( F(m) \). We write \( F(\mu)_\gamma \) for the left adjoint of \( F(\mu) \).

Definition A.4. An object \( F : \cV \) of an MTT cosmos when equipped with the following structure:

1. In \( F(m) \), there is a universe \( \tau_m : \mathcal{T}_m^\bullet \to \mathcal{T}_m \) with a choice of codes witnessing its closure under dependent sums and products, identity types, and boolean e.g.

\[ \sum_{A : \mathcal{T}_m} \sum_{B : \tau_m[A]} \sum_{\alpha : \tau_m[A]} \tau_m[B(\alpha)] \xrightarrow{\text{lam}} \mathcal{T}_m^\bullet \]

2. For each \( \mu : n \to m \), there exists a chosen commuting square

\[ F(\mu)(\mathcal{T}_n^\bullet) \xrightarrow{\text{Mod}} \mathcal{T}_m^\bullet \]

\[ F(\mu)(\mathcal{T}_n) \xrightarrow{\text{Prod}} \mathcal{T}_m \quad (11) \]

3. For each \( \mu : n \to m \) and \( \nu : o \to n \), there is a chosen lifting structure \( F(\mu)(m) \vdash F(\mu \circ \nu)(\mathcal{T}_o^\bullet) \times_{\tau_m} \tau_n \), where \( m : F(\nu)(\mathcal{T}_o^\bullet) \to F(\nu)(\mathcal{T}_o) \times_{\tau_m} \mathcal{T}_n^\bullet \) is the comparison map induced by Diagram 11.

4. \( \tau_m \) contains a subuniverse also closed under all these connectives.

Definition A.5. A morphism between MTT-structured cosmoi \( a : F \to G \) is a 2-natural transformation \( \alpha \) such that \( a_m(\alpha) \) is an LCCC functor and preserves all connectives strictly. Furthermore, we require that there is a natural isomorphism \( \beta_\mu : a_m \circ F(\mu) \cong G(\mu) \circ a_m \) commuting with transposition. Precisely, if \( a : X \to F(\mu)(Y) : F(m) \) the transposition of \( a_\mu \circ a_m(\alpha) \) is \( a_\mu(a(\bar{\alpha})) \circ \beta_\mu^{-1}(\alpha) \).

B Proof of quasi-projectivity

We briefly describe the proof of Theorem 3.8.

Theorem 3.8. Fix an MTT cosmos \( G \) and \( \pi : G \to \mathcal{S} \).

1. For \( \Gamma \vdash \mu \circ \eta \), there exists \( \mathbb{I}([\mu]) : G(m) \) and a canonical isomorphism \( \alpha_\eta : \pi([\mu]) \cong \eta(\mathbb{I}) \).
2. For every \( \Gamma \vdash \eta : \mu \), there exists \( \mathbb{I}([\mu]) : [\Gamma] \to \mathcal{T}_m \) such that \( \pi([\mu]) \circ \alpha_\eta = [\mu] \).
3. For every \( \Gamma \vdash M : \mu \), there exists \( \mathbb{I}([M]) : [\Gamma] \to \mathcal{T}_m^* \) lying over \( \mathbb{I}([\mu]) \) such that \( \pi([M]) \circ \alpha_\eta = [M] \).

Here \([-\cdot] \) is the isomorphism induced by the Yoneda lemma.

Remark 8. While we have proven this result at quite generally, we will apply it only in the special case where \( \pi \) is a 2-natural transformation between strict 2-functors and required isomorphisms of left adjoints are likewise identities.

Once this model is constructed, the result is a direct consequence of Theorem 3.3. The construction of contexts, substitutions, terms, and types is relatively routine as \( \pi \) is a 2-natural transformation which preserves finite limits, and commutes with all connectives. We show two representative cases.

The action of a modality on a context. Given a triple \( (\Gamma, \alpha, \mu) \) at mode \( n \) and a modality \( \mu : n \to m \), we define the 'locked' context to be the following:

\[ \beta : \pi(G(\mu),X) \equiv \mathbb{I}([\mu]), \pi(X) \quad \gamma : \mu \equiv \mathbb{I}([\mu]) \]

Here \( \beta : \pi(G(\mu),X) \equiv \mathbb{I}([\mu]), \pi(X) \equiv \mathbb{I}([\mu]) \) are the canonical isomorphisms.

Modal types. Suppose we are given a context \( (\Gamma, \alpha, \mu) \) and a type \( (A', A) \) in the context \( (G(\mu), [\mu](\mu)(X)), \Gamma \vdash \mu \). We form the modal type as

\[ (\text{Mod}_{\mu}(A'), \langle \mu | A \rangle) \]

It remains to check that these types are coherent i.e.:
By assumption, $\pi(A^*) = [A] \circ \gamma \circ [\bar{\mu}]^*(\alpha) \circ \beta$. By our assumption that $\pi$ satisfies Beck-Chevalley $\pi(A^*) = [A] \circ \gamma \circ \alpha$. The result follows from the fact that $\pi$ preserves $\text{Mod}_\mu$. □

C Internal definition of an MTT cosmos

We present the full sequence of constants which specify an MTT cosmos internally.

**Term and type sorts**

\[
\begin{align*}
Ty_m &: U \\
Tm_m &: Ty_m \rightarrow U
\end{align*}
\]

**Dependent sums**

\[
\begin{align*}
\text{Sig} &: (A : Ty_m) \rightarrow Tm_m(A) \rightarrow Ty_m \rightarrow Ty_m \\
\text{as}_{\text{Sig}} &: (A : Ty_m)(B : Tm_m(A)) \rightarrow Ty_m \\
& \quad \rightarrow \left[ Tm_m(\text{Sig}(A, B)) \equiv \sum_{a : Tm_m(A)} Tm_m(B(a)) \right]
\end{align*}
\]

**Dependent products**

\[
\begin{align*}
\text{Prod} &: (A : Ty_m)(B : (\mu | Tm_m(A))) \rightarrow Ty_m \\
\text{as}_{\text{Prod}} &: (A : Ty_m)(B : Tm_m(A)) \rightarrow Ty_m \\
& \quad \rightarrow \left[ Tm_m(\text{Prod}(A, B)) \equiv \prod_{a : Tm_m(A)} Tm_m(B(a)) \right]
\end{align*}
\]

**Booleans**

\[
\begin{align*}
\text{Bool} &: Ty_m \\
\text{true}, \text{false} &: Tm_m(\text{Bool}) \\
\text{if} &: (A : Tm_m(\text{Bool})) \rightarrow Ty_m \\
& \quad \rightarrow Tm_m(A(\text{true})) \rightarrow Tm_m(A(\text{false})) \\
\text{let} &: (A : Tm_m(\text{Bool})) \rightarrow Ty_m \\
& \quad \rightarrow (t : Tm_m(A(\text{true}))) \\
& \quad \rightarrow (f : Tm_m(A(\text{false}))) \\
& \quad \rightarrow (\text{if}(A, t, f, \text{true}) = t) \times (\text{if}(A, t, f, \text{false}) = f)
\end{align*}
\]

**Identity types**

\[
\begin{align*}
\text{Id} &: (A : Ty_m)(a_0, a_1 : Tm_m(A)) \rightarrow Ty_m \\
\text{refl} &: (A : Ty_m)(a : Tm_m(A)) \rightarrow Tm_m(\text{Id}(A, a, a)) \\
\text{J} &: (A : Ty_m) \\
& \quad (B : (a_0, a_1 : Tm_m(A)) \rightarrow Ty_m) \\
& \quad \rightarrow ((a_0 : Tm_m(A)) \rightarrow Tm_m(B(a, a, \text{refl}(a)))) \\
& \quad \rightarrow ((a_1 : Tm_m(A)) \rightarrow Tm_m(\text{Id}(A, a_0 a_1))) \\
& \quad \rightarrow Tm_m(B(a_0, a_1, \text{refl}(a))) \\
\text{let} &: (A : Ty_m) \\
& \quad (B : (a_0, a_1 : Tm_m(A)) \rightarrow Ty_m) \\
& \quad \rightarrow (b : (a : Tm_m(A)) \rightarrow Tm_m(B(a, a, \text{refl}(a)))) \\
& \quad \rightarrow (a : Tm_m(A)) \rightarrow J(A, B, b, a, a, \text{refl}(a)) = b(a)
\end{align*}
\]

**Modal types**

\[
\begin{align*}
\text{Mod}_\mu &: (\mu | Ty_m) \rightarrow Ty_m \\
\text{m}_\mu &: (\mu | A : Ty_m) \rightarrow Tm_m(\text{Mod}_\mu(A)) \\
\text{letmod}_\mu &: (\nu \circ \mu | A : Ty_m) \\
& \quad (B : (\nu \circ \mu | x : Tm_m(A))) \rightarrow Ty_m \\
& \quad \rightarrow Tm_m(B(a)) \\
& \quad \rightarrow Tm_m(\text{letmod}_\mu(A, B, b, m_\mu(A, a)) = b(a)
\end{align*}
\]

**Universe à la Tarski**

\[
\begin{align*}
\text{Uni} &: Ty_m \\
\text{El} &: Tm_m(\text{Uni}) \rightarrow Ty_m \\
\text{Sig} &: (A : Tm_m(\text{Uni})) \rightarrow (Tm_m(\text{El}(A))) \rightarrow Tm_m(\text{Uni}) \\
\text{Prod} &: (A : Tm_m(\text{Uni})) \rightarrow (Tm_m(\text{El}(A))) \rightarrow Tm_m(\text{Uni}) \\
\text{Bool} &: Tm_m(\text{Uni}) \\
\text{Mod} &: (\mu | Tm_m(\text{Uni})) \rightarrow Tm_m(\text{Uni}) \\
\text{dec}_{\text{Sig}} &: (A : Tm_m(\text{Uni}))(B : Tm_m(\text{El}(A))) \rightarrow Tm_m(\text{Uni}) \\
& \quad \rightarrow Tm_m(\text{El}(\text{Sig}(A, B))) \equiv Tm_m(\text{Sig}(\text{El}(A), \text{El} \circ B)) \\
\text{dec}_{\text{Prod}} &: (A : Tm_m(\text{Uni}))(B : Tm_m(\text{El}(A))) \rightarrow Tm_m(\text{Uni}) \\
& \quad \rightarrow Tm_m(\text{El}(\text{Prod}(A, B))) \equiv Tm_m(\text{Prod}(\text{El}(A), \text{El} \circ B)) \\
\text{dec}_{\text{Bool}} &: Tm_m(\text{El}(\text{Bool})) \equiv Tm_m(\text{Bool}) \\
\text{dec}_{\text{Mod}} &: (\mu | A : Tm_m(\text{Uni})) \rightarrow Tm_m(\text{El}(\text{Mod}(A))) \equiv Tm_m(\text{Mod}_\mu(\text{El}(A)))
\end{align*}
\]

**D Normal and neutral forms, internally**

We present a representative selection of the internalization of various normal and neutral forms. We require that all normal and neutral forms become equal to their counterparts in the syntactic internal MTT cosmos under the assumption $z : \text{syn}$ but avoid repeatedly stating this.

**Normal types**

\[
\begin{align*}
\text{Prod} &: (A : NfTy_m)(B : V_m(A) \rightarrow NfTy_m) \rightarrow NfTy_m \\
\text{Sum} &: (A : NfTy_m)(B : V_m(A) \rightarrow NfTy_m) \rightarrow NfTy_m
\end{align*}
\]
\[ \text{Booleans} \]
\[ \begin{align*}
& \text{up : } \text{Ne}_m(\text{Bool}) \rightarrow \text{Nf}_m(\text{Bool}) \\
& \text{tt, ff : } \text{Nf}_m(\text{Bool}) \\
& \text{if : } (A : \text{V}_m(\text{Bool}) \rightarrow \text{Nf}_m(\text{Bool})) \\
& \quad \rightarrow (b : \text{Ne}_m(\text{Bool})) \rightarrow \text{Ne}_m(A(b)) \\
\end{align*} \]

\[ \text{Identity types} \]
\[ \begin{align*}
& \text{up : } (A : \text{O}_m(\text{Ty}))((a_0, a_1 : \text{O}_m(\text{Ty}(A)) \\
& \quad \rightarrow \text{Ne}_m(\text{Id}(A, a_0, a_1))) \\
& \quad \rightarrow \text{Nf}_m(\text{Id}(A, a_0, a_1)) \\
& \text{refl : } (A : \text{O}_m(\text{Ty}(z)))(a : \text{O}_m(\text{Ty}(z, A(z)))) \rightarrow \text{Nf}_m(\text{Id}(A, a, a)) \\
& \quad \rightarrow (b : \text{Ne}_m(\text{Id}(A, a_0, a_1))) \\
& \quad \rightarrow \text{Ne}_m(B(a_0, a_1, \eta(p))) \\
\end{align*} \]

\[ \text{Modal types} \]
\[ \begin{align*}
& \text{up : } (\mu : A : \text{Ty}_n) \rightarrow \text{Ne}_m(\text{Mod}_\mu(A)) \rightarrow \text{Nf}_m(\text{Mod}_\mu(A)) \\
& \text{mod}_\mu : (\mu : A : \text{Ty}_n) \rightarrow \text{Nf}_m(\lambda z. \text{Mod}_\mu(z, A(z))) \\
& \text{letmod}_\mu : (v \circ \mu : A : \text{Ty}_n) \quad (A : \text{Ty}_n) \\
& \quad \rightarrow (B : (v \circ \mu : A : \text{Ty}_n)) \rightarrow \text{Nf}_m(\text{Mod}_\mu(A)) \\
& \quad \rightarrow ((v \circ \mu : A : \text{Ty}_n)) \rightarrow \text{Ne}_m(B(m_\mu(a)))) \\
& \quad \rightarrow (v \circ a : \text{Ne}_m(\text{Mod}_\mu(A))) \rightarrow \text{Ne}_n(B(a)) \\
\end{align*} \]

\[ \text{Universe à la Tarski} \]
\[ \begin{align*}
& \text{Uni : } \text{Nf}_m(\text{Ty}_m) \\
& \text{El : } \text{Ne}_m(\text{Uni}) \rightarrow \text{Nf}_m(\text{Uni}) \\
& \text{up : } \text{Ne}_m(\text{Uni}) \rightarrow \text{Nf}_m(\text{Uni}) \\
& \text{mod}_\mu : (\mu : A : \text{Nf}_n(\text{Uni})) \rightarrow \text{Nf}_m(\text{Uni}) \\
& \text{dec}^{\text{mod}_\mu} : (\mu : A : \text{Nf}_n(\text{Uni})) \\
& \quad \rightarrow \text{Nf}_m(\text{Mod}_\mu(A)) \rightarrow \text{Ne}_m(\text{Mod}_\mu(A)) \\
& \text{dec}^{\text{mod}_\mu} : (\mu : A : \text{Nf}_n(\text{Uni})) \\
& \quad \rightarrow \text{Ne}_m(\text{El}(\text{Mod}_\mu(A))) \rightarrow \text{Ne}_m(\text{Mod}_\mu(A)) \\
\end{align*} \]

\[ \text{E Normalization structures} \]

We present the normalization structures missing from Section 5.2.

**Lemma E.1.** \( (\text{Ty}_m, \text{Tm}_m^*) \) is closed under dependent sums.

**Proof.** Fixing \( A : \text{Ty}_m \) and \( B : \text{Tm}_m^*(A) \rightarrow \text{Ty}_m^* \). We must construct the following pair of constants:

\[ \text{Sig}^*(A, B) : \text{Ty}_m^* \]
\[ a_{\text{Sig}}^* : \text{Tm}_m^*(\text{Sig}^*(A, B)) \equiv \sum_{a : \text{Tm}_m(A)} \text{Tm}_m^*(B(a)) \]

Such that they lie over \( \text{Sig} \) and \( a_{\text{Sig}} \) respectively.

We begin by applying realignment to the following:

\[ \left( \sum_{a : A\text{-pred}} B(a).\text{pred}, a_{\text{Sig}(z)} \right) \]

This produces \( \Psi : U_1 \) and \( a_{\text{Sig}}^* : \Psi \equiv \sum_{a : A\text{-pred}} B(a).\text{pred} \) such that under the assumption \( z : \text{syn} \) the following holds:

- \( \Psi = \text{Sig}(z, A, B) \)
- \( a_{\text{Sig}}^* = a_{\text{Sig}}(z) \).

We now define \( \text{Sig}^*(A, B) \) as follows:

\[ \text{Sig}^*(A, B).\text{code} = \text{Sum}(\text{code}A, \lambda v. B.\text{code}(\uparrow_A v)) \]
\[ \text{Sig}^*(A, B).\text{pred} = \Psi \]
\[ \text{Sig}^*(A, B).\text{reflect} = \lambda e. a_{\text{Sig}}^{-1}((A_{\text{proj}_0}(e)), (B_{\text{proj}_1}(e))) \]

The fact that \( \downarrow \) and \( \uparrow \) lie over the identity follows directly from the \( \beta \) and \( \eta \) laws of dependent sums in MTT. We show the calculations for \( \uparrow \). Fix \( z : \text{syn} \):

\[ \uparrow_{\text{Sig}(A, B)}(e) = a_{\text{Sig}}^{-1}((A_{\text{proj}_0}(e)), (B_{\text{proj}_1}(e))) \]
\[ = a_{\text{Sig}}^{-1}((\text{proj}_0(e)), (\text{proj}_1(e))) \]
\[ = a_{\text{Sig}}(a_{\text{Sig}(A, B)}(e)) \]
\[ = e \]

\[ \text{Lemma E.2.} \quad (\text{Ty}_m^*, \text{Tm}_m^*) \text{ is closed under booleans.} \]

**Proof.** We must implement the following constants:

\[ \text{Bool}^* : \{ \text{Ty}_m^* | z : \text{syn} \mapsto \text{Bool}(z) \} \]
\[ \text{true}^* : \{ \text{Tm}_m^*(\text{Bool}) | z : \text{syn} \mapsto \text{true} \} \]
\[ \text{false}^* : \{ \text{Tm}_m^*(\text{Bool}) | z : \text{syn} \mapsto \text{false} \} \]
\[ \text{if}^* : (A : \text{Tm}_m^*(\text{Bool}(z)) \rightarrow \text{Ty}_m^* \rightarrow \text{Tm}_m^*(A(\text{true}^*)) \rightarrow \text{Tm}_m^*(A(\text{false}^*))) \]
Normalization for Multimodal Type Theory

LICS ‘22, August 2–5, 2022, Haifa, Israel

if

\[ \Phi (A, t, f, b) \]

First, we define \( \Phi \) by realignment:

**record** \( \Phi : \{ U_1 \mid z : \text{syn} \mapsto \text{Tm}_m(z, \text{Bool}) \} \)

where

\[ \text{tm} : \text{Nf}_m(\text{Bool}) \]

\[ \text{prf} : ( \sum_{e : \text{Nem}(\text{Bool})} \text{tm} = \text{up}(e) ) \]

We may now define \( \text{Bool}^* \):

\[ \text{Bool}^* . \text{code} = \text{Bool} \]

\[ \text{Bool}^* . \text{pred} = \Phi \]

\[ \text{Bool}^* . \text{reflect} = \lambda e . \langle \text{up}(e), \eta(\iota_1(e, *)) \rangle \]

\[ \text{Bool}^* . \text{reify} = \lambda b . b . \text{tm} \]

It remains to define the introduction and elimination forms.

\[ \text{true}^* = (\text{tt}, \eta(\iota_2(0, *))) \]

\[ \text{false}^* = (\text{ff}, \eta(\iota_2(1, *))) \]

The elimination form is defined by constructing a map out of \( \bullet X \), as in **Lemma 5.6**:

\[ \text{if}^*(A, t_0, t_1, b = \langle \text{tm}, \text{prf} \rangle) = \]

\[ \begin{cases} 
\text{if}(z, T_m, t_0, t_1, s) & \text{prf} = \iota_1(z) \\
\downarrow_{A, b} \text{if}(\lambda b . A(\uparrow_0). \text{code}, \downarrow_0, \downarrow_1, e) & \text{prf} = \iota_2(\iota_1(e, _*)) \\
\iota_l & \text{prf} = \iota_2(\iota_l(\_*, _*)) \end{cases} \]

**Lemma E.3.** \((\text{Ty}_m^*, \text{Tm}_m^*)\) is closed under intensional identity types.

**Proof:** We must implement the following constants:

\[ \text{Id}^* : (A : \text{Ty}_m^* (a_0, a_1 : \text{Tm}_m^*(A))) \]

\[ \rightarrow \{ \text{Tm}_m^*(\text{Id}(A, a_0, a_1)) \} \]

\[ \text{refl}^* : (A : \text{Ty}_m^* (a_0 : \text{Tm}_m^*(A))) \]

\[ \rightarrow \{ \text{Tm}_m^*(\text{Id}(A, a_0)) \mid z : \text{syn} \mapsto \text{refl}(z, A, a) \} \]

\[ J^* : (A : \text{Ty}_m^*) \]

\[ \rightarrow (B : \{ a_0, a_1 : \text{Tm}_m^*(A) \}) \rightarrow \text{Tm}_m^*(\text{Id}^* (A, a_0, a_1)) \rightarrow \text{Ty}_m^* \]

\[ \rightarrow (b : (a : \text{Tm}_m^*(A)) \rightarrow \text{Tm}_m^*(\text{Id}^* (A, a_0, a_1))) \rightarrow \text{Ty}_m^* \]

\[ \rightarrow (a_0, a_1 : \text{Tm}_m^*(A)) \rightarrow \text{Tm}_m^*(B(a, a, \text{refl}(a))) \]

\[ \rightarrow \{ \text{Tm}_m^*(B(a_0, a_1, p)) \mid z : \text{syn} \mapsto J^*(z, B, b, p) \} \]

\[ \rightarrow (a : \text{Tm}_m^*(A)) \rightarrow J^*(A, B, b, \text{refl}^*(a)) = b(a) \]

Fix \( A : \text{Ty}_m^* \) and \( a_0, a_1 : \text{Tm}_m^*(A) \). Just as with the normalization structure for booleans, we begin by defining \( \Phi \) by realignment:

**record** \( \Phi : \{ U_1 \mid z : \text{syn} \mapsto \text{Tm}_m(z, \text{Id}(A, a_0, a_1)) \} \)

where

\[ \text{tm} : \text{Nf}_m(\text{Id}(A, a_0, a_1)) \]

\[ \text{prf} : ( \sum_{e : \text{Nem}(\text{Id}(A, a_0, a_1))} \text{tm} = \text{refl}(\downarrow A a) ) \]

We now define \( \text{Id}^* \):

\[ \text{Id}^*(A, a_0, a_1). \text{code} = \text{Id}_{\text{code}A}(\uparrow A a_0, \uparrow A a_1) \]

\[ \text{Id}^*(A, a_0, a_1). \text{pred} = \Phi \]

\[ \text{Id}^*(A, a_0, a_1). \text{reflect} = \lambda e . \langle \text{up}(e), \eta(\iota_1(e, *)) \rangle \]

\[ \text{Id}^*(A, a_0, a_1). \text{reify} = \lambda b . b . \text{tm} \]

We define reflexivity by \( \text{refl}^* = (\text{refl}, \eta(\iota_2(*, *)) \).

Finally, as is now routine, the elimination principle is defined using the induction principle for \( \bullet X \).

\[ J^*(B, b, a_0, a_1, p = \langle \text{tm}, \text{prf} \rangle) = \]

\[ \begin{cases} 
J(z, B, b, a_0, a_1, p) & \text{prf} = \iota_1(z) \\
J(\lambda l, r, p . B(\uparrow l, \uparrow r, \uparrow p). \text{code}, \downarrow_0, \downarrow_1, e) & \text{prf} = \iota_2(\iota_1(e, _*)) \\
b(a_0) & q = \iota_2(\iota_2(\_*, _*)) \end{cases} \]

**Lemma E.4.** \((\text{Ty}_m^*, \text{Tm}_m^*)\) is closed under universes.

**Proof:** We show only that the following constants exist as their closure under various connectives is straightforward.

\[ \text{Uni}^* : (\text{Ty}_m^* \mid z : \text{syn} \mapsto \text{Uni}) \]

\[ \text{El}^* : (A : \text{Tm}_m^*(\text{Uni})) \rightarrow \{ \text{Tm}_m^* \mid z : \text{syn} \mapsto \text{El}(A) \} \]

At this point we take advantage of the fact that \( \text{pred} \) is an element of \( U_1 \); in particular, we use the fact that is a universe \( U_0 \) small enough to fit inside \( U_1 \).

We may then define \( \Psi \) by realigning the following element of \( U_1 \) along the evident isomorphism to \( \text{Tm}_m^*(z, \text{Uni}(z)) \):

**record** \( \Psi : \{ U_1 \mid z : \text{syn} \mapsto \text{Tm}_m(z, \text{Uni}) \} \)

where

\[ \text{code} : \text{Nf}_{\text{Uni}}(\text{Uni}) \]

\[ \text{pred} : \langle U_0 \mid z : \text{syn} \mapsto \text{Tm}_m(z, \text{El}(\text{code})) \rangle \]

\[ \text{reflect} : \{ \text{Nem}(\text{El}(\text{code})) \} \rightarrow \text{pred} \mid z : \text{syn} \mapsto \text{id} \]

\[ \text{reify} : \{ \text{pred} \rightarrow \text{Nf}_{\text{Uni}}(\text{El}(\text{code})) \} \mid z : \text{syn} \mapsto \text{id} \]

With \( \Psi \) in hand, we may define \( \text{Uni}^* \):

\[ \text{Uni}^*. \text{code} = \text{Uni} \]

\[ \text{Uni}^*. \text{pred} = \Psi \]

\[ \text{Uni}^*. \text{reflect} = \lambda e . \langle \text{up}(e), \text{Nem}; \text{id}; \lambda e . \text{up}(e) \rangle \]

\[ \text{Uni}^*. \text{reify} = \lambda A . \text{A.code} \]

The definition of \( \text{El}^* \) is essentially cumulativity:

\[ \text{El}^*(\langle \text{code}; \text{pred}; \text{reify}; \text{reflect} \rangle) = \]

\[ \langle \text{El}(\text{code}); \text{pred}; \text{reify}; \text{reflect} \rangle \]
F  Crisp identity induction principles

The addition of crisp identity induction requires two changes to primitive constants added to MSTC by Section 5.1 compared to Appendix E: one altering both the definition of cosmoi and the other the definition of neutral forms.

\[ J_\mu : (\mu \mid A : Ty_n) \]
\[ (B : (\mu \mid a_0, a_1 : Tm_n(A)) (\mu \mid p : Tm_n(Id(A, a_0, a_1)))) \rightarrow Ty_m \]
\[ \rightarrow ((\mu \mid a : Tm_n(A)) \rightarrow Tm_m(B(a, a, refl(a)))) \]
\[ \rightarrow (\mu \mid a_0, a_1 : Tm_n(A)) (\mu \mid p : Tm_n(Id(A, a_0, a_1)))) \rightarrow Tm_m(B(a_0, a_1, p)) \]

\[ J_\mu : (\mu \mid A : \bigcirc Ty_n) \]
\[ (B : (\mu \mid a_0, a_1 : V_n(A)) (\mu \mid p : V_m(Id(A, a_0, a_1)))) \rightarrow NfTy_m \]
\[ \rightarrow ((\mu \mid a : V_n(A)) \rightarrow Nf_m(B(a, a, refl(a)))) \]
\[ \rightarrow (\mu \mid a_0, a_1 : \bigcirc Tm_n(z, A(z))) (\mu \mid p : Nf_n(Id(A, a_0, a_1))) \rightarrow Nf_m(B(a_0, a_1, \eta(p))) \]

It only remains to define \( J_\mu ^* \) analogous to Lemma E.3:

**Lemma F.1.** \((Ty_n^*, Tm_n^*)\) is supports crisp identity induction.

**Proof.** This argument is similar to Lemma 5.6, as the induction principle for modal types is always ‘crisp’ in MTT.

We must implement the following constant.

\[ J_\mu ^* : (\mu \mid A : Ty_n^*) \]
\[ (B : (\mu \mid a_0, a_1 : Tm_n^*(A)) (\mu \mid p : Tm * n(Id'(A, a_0, a_1)))) \rightarrow Ty_m^* \]
\[ \rightarrow (b : (\mu \mid a : Tm_n^*(A)) \rightarrow Tm_m^*(B(a, a, refl'(a)))) \]
\[ \rightarrow (\mu \mid a_0, a_1 : Tm_n^*(A)) (\mu \mid p : Tm_m^*(Id(A, a_0, a_1)))) \rightarrow \]
\[ \rightarrow \{Tm_m^*(B(a_0, a_1, p)) \mid z : \text{syn} \mapsto J_\mu (A, B, b, p)) \]

Let us fix \( A, B, b, a_0, a_1, \) and \( p \) with the types described above. Recalling the definition of pred'd'(A, a_0, a_1) from Lemma E.3, we can commute \( \langle \mu \mid - \rangle \) past the dependent sum, closed modalities, equality types, and coproducts to decompose \( p \) into a pair of the following:

\[ (\mu \mid tm : Nf_n(Id(A, a_0, a_1))) \]
\[ \text{prf} : \bullet \sum e : (\mu \mid Nf_n(Id(A, a_0, a_1))) \uparrow e \oplus e = \text{mod}_\mu (m) \]
\[ + \text{mod}_\mu (m) = \text{mod}_\mu (\text{refl}(a_0)) \]

We then define \( J_\mu ^*(B, b, a_0, a_1, p) \) by analyzing \( \text{prf} \):

\[ J(z, B, b, a_0, a_1, p) \]
\[ \downarrow J(\lambda a_0. a_1, p. B(\uparrow a_0, \uparrow a_1, \uparrow p). \text{code}, \lambda a. \downarrow b(\uparrow a), e) \]
\[ b(a_0) \]
\[ \text{prf} = t_1(z) \]
\[ q = t_2(t_1(e, _)) \]
\[ q = t_2(t_2(_)) \]