Type-Theoretic Signatures for Algebraic Theories and Inductive Types

András Kovács

Submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

Eötvös Loránd University
Doctoral School of Informatics

September 2021

Contents

CHAPTER 1

Introduction

This thesis develops the usage of certain type theories as specification languages for algebraic theories and inductive types.

Type theories have emerged as popular metatheoretic settings for mechanized mathematics. One reason is that the field of type theory is generally aware of the issue of overheads in representation, and it is a common endeavor to search for concise "synthetic" ways to talk about different areas of mathematics. In type theory, it is a virtue to be able to directly say what we mean, and in a way such that simple-minded computers are able to verify it.

Algebraic theories are certain mathematical structures which are especially well-behaved, and which are ubiquitous in mathematics, such as groups or categories. In type theories, *inductive types* are certain freely generated (initial) models of algebraic theories. Inductive types are a core feature in implementations of type theories, widely used in mathematical formalization, but also as the primary way to define the data structures which are used in programming.

This thesis observes that if we are to specify more complicated algebraic theories, dependent type theories provide the natural tool to manage complexities. The expressive power of type theory which makes it suitable as a foundation for mechanized mathematics, also proves useful for the more specialized task of specifying algebraic signatures.

There is a trade-off between the complexity of a mathematical language and the ease of usage of the language. Minimal languages are convenient to reason about and develop metatheory for, but they often require an excessive amount of boilerplate to work in. However, it is a worthwhile effort to try to move towards the Pareto frontier of this trade-off. We believe that the current thesis makes

progress in this respect.

Our signatures are useful in broader mathematical contexts, but we are also concerned with potential implementation in proof assistants. Although it is unlikely that our signatures can be deployed in practice exactly as they are, they should be still helpful as formal bases of practical implementations.

1.1 Overview

In **Chapter 2**, we present a minimal example of a type theory of signatures. This allows specifying single-sorted signatures without equations. The purpose of the chapter is didactic. We develop just enough semantics to get notions of initiality and induction for algebras. We also present a *term algebra construction*: this shows that the initial algebra for each signature can be constructed from the syntax of signatures itself.

In **Chapter 3** we describe a metatheoretic setting which is often used in the thesis. This is *two-level type theory* [ACKS19]. It allows us to develop general semantics for signatures, while still working inside a convenient type theory. As a demonstration, we generalize the semantics from Chapter 2 so that it is given internally to arbitrary categories-with-families. As a special case, signatures can be interpreted in arbitrary categories with finite products.

In **Chapter 4** we describe finitary quotient inductive-inductive signatures. These are close to generalized algebraic theories [Car86] in expressive power. In particular, most type theories themselves can be specified with finitary quotient inductive-inductive signatures. We significantly expand the semantics of signatures, now for each signature we provide a category of algebras with certain extra structure, which is equivalent to having finite limits. This allows us to prove for each signature the equivalence of initiality and induction. Also, owing to two-level type theory, signatures can be interpreted internally to any category with finite limits. Additionally

- We present a term algebra construction.
- We show that morphisms of signatures are interpreted as right adjoint functors in the semantics.
- We present how self-description of signatures can be exploited to minimize metatheoretic assumptions.

- We describe ways to construct initial algebras from simple type formers.

In **Chapter 5**, we describe infinitary quotient inductive-inductive signatures. These allow specification of infinitely branching trees (as initial algebras). We adapt the semantics from the previous chapter. We also revisit term models, left adjoints of signature morphisms and self-description of signatures. Self-description in particular is significantly strengthened, since the full theories of signatures in Chapters 4-5 can be now described using infinitary quotient inductive-inductive signatures.

In **Chapter 6**, we describe higher inductive-inductive signatures. These differ from the previous signatures mostly in their intended semantics, whose context is now homotopy type theory [Uni13], and which allows specified equalities to be proof-relevant. The higher-dimensional generalization of types and equalities makes semantics more complicated, so we only present enough semantics to specify notions of initiality and induction for each signature. Additionally, we consider two different notions of algebra morphisms: one preserves structure strictly (up to definitional equality), while the other preserves structure up to paths.

1.2 How to Read This Thesis

We list several general references which could be helpful for readers.

- It is useful to have some user experience with a type-theory-based proof assistant or programming language, such as Agda, Coq, Lean or Idris. In the author's view, mechanized formalization is the most effective way to build intuition about working in type theories.
- We heavily use categories-with-families [CCD19, Hof97, Dyb95] throughout the thesis.
- We use a modest amount of category theory, for which [Awo10] should be a sufficient reference.
- For Chapter 6, the Homotopy Type Theory book [Uni13] provides context and motivation.

This thesis is mostly written in a linear fashion, as later chapters often revisit or generalize earlier concepts. There are some breaks from linearity though, so we summarize dependencies between chapters as follows:

- Chapter 3 depends on Chapter 2.
- All chapters after Chapter 3 depend on it.
- Chapter 5 depends on Chapter 4 as it revisits most constructions from there.
- Chapter 6 only depends on Chapter 3.

1.3 Formalization

Chapter 2 is fully formalized in Agda, and the semantics of weak signatures in Chapter 6 is mostly formalized, with some omissions and shortcuts. The formalization can be found in [**?**].

1.4 Notation and Conventions

Throughout this thesis, we always work in some sort of type theory, although the exact flavor of the type theory will vary. We summarize here the notations and conventions that will stay consistent. Our style of notation is a mostly a mix of the homotopy type theory book [Uni13] and the syntax of the proof assistant Agda.

Σ-types

We write a dependent pair type as $(a : A) \times B$, where B may refer to a. Pairing is (t, u), and projections are proj_1 and proj_1. Iterated Σ-types can written as $(a : A) \times (b : B) \times C$, for example. We often silently re-associate left-nested Σ-types to the right, e.g. write $(a : A) \times (b : B) \times C$ instead of $(ab : (a : A) \times B) \times C$.

Field projection notation: we reuse binder names in Σ-types as field projections. For example, if we have $t : (\mathsf{foo} : A) \times B$, then foo_t projects the first component from t. To make this a bit more convenient, we also allow to name the last components, for example if $t : (\mathsf{foo} : A) \times (\mathsf{bar} : B)$, then we have $\mathsf{foo}_t : A$, and $\mathsf{bar}_t : B[\mathsf{foo} \mapsto \mathsf{foo}_t]$. This notation is useful when we handle components of more complicated algebraic structures.

Unit type

Whenever the unit type is available, we name it $\top$, and its inhabitant tt.

Π-types

Dependent function types are written as $(a : A) \to B$, where B may depend on the a variable. It is possible to group multiple binders with the same type, as in $(x\,y : A) \to B$. For non-dependent function types, we write plainly $A \to B$. Functions are defined as $\lambda\, x.\, t$. We may group multiple binders, as in $\lambda\, x\, y\, z.\, t$, and optionally add type annotation to binders, as in $\lambda\,(x : A).\, t$.

We also use Agda-like implicit arguments: a function type $\{a : A\} \to B$ signals that we usually omit the argument in function applications. For example, if $\mathsf{id} : \{A : \mathsf{Set}\} \to A \to A$, we write $\mathsf{id}\,\mathsf{true} : \mathsf{Bool}$. We can still make these arguments explicit, by using bracketed application, as in $\mathsf{id}\,\{\mathsf{Bool}\}\,\mathsf{true}$. Similarly, we may use bracketed λ, as in $\lambda\,\{A : \mathsf{Set}\}\,(x : A).\, x$, to bind implicit arguments.

Sometimes we also write pattern matching abstraction, as in $\lambda\,(x,\, y).\, t$ for a function with a Σ domain.

We may use implicit quantification as well: argument binders and types may be entirely omitted when it is clear where they are quantified. This resembles the implicit generalization in the Haskell or Idris programming languages. For example, the A and B types are implicitly quantified below:

$$\mathsf{map} : (A \to B) \to \mathsf{List}\,A \to \mathsf{List}\,B$$
$$\mathsf{map} :\equiv \ldots$$

Identity types

We use $- \equiv -$ and $- = -$ to denote identity types. We always use $- \equiv -$ as a "strict" equality which satisfies uniqueness of identity proofs. Reflexivity of identity is always written as refl. We use $- = -$ as intensional identity in Chapter 2. In later chapters, $- = -$ denotes the identity type in the inner layer of a two-level type theory, and $- \equiv -$ denotes the outer identity type.

Definitions

We give definitions using $:\equiv$, for example as in

$$\mathsf{id} : \{A : \mathsf{Set}\} \to A \to A$$
$$\mathsf{id}\,a :\equiv a$$

Note that we write the function argument on the left of $:\equiv$, instead of writing a λ on the right. We may switch between the two styles. The type signature can be omitted in a definition. We may also use pattern matching, like in $\mathsf{foo}\,(x,\, y) :\equiv ....$

CHAPTER 2

Simple Signatures

In this chapter, we take a look at a very simple notion of algebraic signature. The motivation for doing so is to present the basic ideas of this thesis in the easiest possible setting, with explicit definitions. The later chapters are greatly generalized and expanded compared to the current one, and are not feasible (and probably not that useful) to present in full formal detail. We also include a complete Agda formalization of the contents of this chapter, in less than 200 lines.

The mantra throughout this dissertation is the following: algebraic theories are specified by typing contexts in certain *theories of signatures*. For each class of algebraic theories, there is a corresponding theory of signatures, which is viewed as a proper type theory and comes equipped with a model theory. *Semantics* of signatures is given by interpreting them in certain models of the theory of signatures. Semantics should at least provide a notion of induction principle for each signature; in this chapter we provide a bit more than that, and we will do substantially more in Chapters 4 and 5.

Metatheory

We work in an intensional type theory which supports Π, Σ, $\top$, intensional identity $- = -$, inductive families, and two universes Set and Set_1 closed under the mentioned type formers, with $\mathsf{Set} : \mathsf{Set}_1$. Since the contents of this chapter are formalized in Agda, and our notation is reminiscent of Agda too, we can think of the metatheory as a subset of Agda.

2.1 Theory of Signatures

Generally, more expressive theories of signatures can describe larger classes of theories. As we are aiming for minimalism right now, the current theory of signatures is as follows:

Definition 1. The **theory of signatures**, or ToS for short, is a simple type theory equipped with the following features:

- An empty base type ι.
- A *first-order function type* $\iota \to -$; this is a function whose domain is fixed to be ι. Moreover, first-order functions only have neutral terms: there is application, but no λ-abstraction.

We can specify the full syntax using the following Agda-like inductive definitions.

$$\begin{array}{ll} \mathsf{Ty} & : \mathsf{Set} \\ \iota & : \mathsf{Ty} \\ \iota \to - & : \mathsf{Ty} \to \mathsf{Ty} \end{array} \qquad \begin{array}{ll} \mathsf{Var} & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set} \\ \mathsf{vz} & : \mathsf{Var}\,(\Gamma \triangleright A)\,A \\ \mathsf{vs} & : \mathsf{Var}\,\Gamma\,A \to \mathsf{Var}\,(\Gamma \triangleright B)\,A \end{array}$$

$$\begin{array}{ll} \mathsf{Con} & : \mathsf{Set} \\ \bullet & : \mathsf{Con} \\ - \triangleright - & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Con} \end{array} \qquad \begin{array}{ll} \mathsf{Tm} & : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set} \\ \mathsf{var} & : \mathsf{Var}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \\ \mathsf{app} & : \mathsf{Tm}\,\Gamma\,(\iota \to A) \to \mathsf{Tm}\,\Gamma\,\iota \to \mathsf{Tm}\,\Gamma\,A \end{array}$$

Here, Con contexts are lists of types, and Var specifies well-typed De Bruijn indices, where vz represents the zero index, and vs takes the successor of an index.

Notation 1. We use capital Greek letters starting from Γ to refer to contexts, A, B, C to refer to types, and t, u, v to refer to terms. In examples, we may use a nameful notation instead of De Bruijn indices. For example, we may write $x : \mathsf{Tm}\,(\bullet \triangleright (x : \iota) \triangleright (y : \iota))\,\iota$ instead of $\mathsf{var}\,(\mathsf{vs}\,\mathsf{vz}) : \mathsf{Tm}\,(\bullet \triangleright \iota \triangleright \iota)\,\iota$. Additionally, we may write $t\,u$ instead of $\mathsf{app}\,t\,u$ for t and u terms.

Definition 2. Parallel substitutions map variables to terms.

$$\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$$
$$\mathsf{Sub}\,\Gamma\,\Delta \equiv \{A : \mathsf{Ty}\} \to \mathsf{Var}\,\Delta\,A \to \mathsf{Tm}\,\Gamma\,A$$

We use σ and δ to refer to substitutions. We also recursively define the action of substitution on terms:

$$
\begin{aligned}
&-[-] : \mathsf{Tm}\,\Delta\,A \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Tm}\,\Gamma\,A \\
&(\mathsf{var}\,x) \quad [\sigma] :\equiv \sigma\,x \\
&(\mathsf{app}\,t\,u)[\sigma] :\equiv \mathsf{app}\,(t[\sigma])\,(u[\sigma])
\end{aligned}
$$

The identity substitution id is defined simply as var. It is easy to see that $t[\mathsf{id}] = t$ for all t. Substitution composition is as follows.

$$
\begin{aligned}
&-\circ- : \mathsf{Sub}\,\Delta\,\Xi \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Sub}\,\Gamma\,\Xi \\
&(\sigma \circ \delta)\,x :\equiv (\sigma\,x)[\delta]
\end{aligned}
$$

Example 1. We may write signatures for natural numbers and binary trees respectively as follows.

$$
\begin{aligned}
\mathsf{NatSig} &:\equiv \bullet \triangleright (\mathit{zero} : \iota) \triangleright (\mathit{suc} : \iota \to \iota) \\
\mathsf{TreeSig} &:\equiv \bullet \triangleright (\mathit{leaf} : \iota) \triangleright (\mathit{node} : \iota \to \iota \to \iota)
\end{aligned}
$$

In short, the current ToS allows signatures which are

- *Single-sorted*: this means that we have a single type constructior, corresponding to ι.
- *Closed*: signatures cannot refer to any externally existing type. For example, we cannot write a signature for "lists of natural number" in a direct fashion, since there is no way to refer to the type of natural numbers.
- *Finitary*: inductive types corresponding to signatures are always finitely branching trees. Being closed implies being finitary, since an infinitely branching node would require some external type to index subtrees with. For example, $\mathit{node} : (\mathbb{N} \to \iota) \to \iota$ would specify an infinite branching (if such type was allowed in ToS).

Remark. We omit λ-expressions from ToS for the sake of simplicity: this causes terms to be always in normal form (neutral, to be precise), and thus we can skip talking about conversion rules. Later, starting from Chapter 4 we include proper $\beta\eta$-rules in theories of signatures.

2.2 Semantics

For each signature, we need to know what it means for a type theory to support the corresponding inductive type. For this, we need at least a notion of *algebras*, which can be viewed as a bundle of all type and value constructors, and what it means for an algebra to support an *induction principle*. Additionally, we may want to know what it means to support a *recursion principle*, which can be viewed as a non-dependent variant of induction. In the following, we define these notions by induction on ToS syntax.

Remark. We use "algebra" and "model" synonymously throughout this thesis.

2.2.1 Algebras

First, we calculate types of algebras. This is simply a standard interpretation into the Set universe. We define the following operations by induction; the $-^A$ name is overloaded for Con, Ty and Tm.

$$
\begin{aligned}
&-^A : \mathsf{Ty} \to \mathsf{Set} \to \mathsf{Set}\\
&\iota^A \qquad\quad X :\equiv X\\
&(\iota \to A)^A\, X :\equiv X \to A^A\, X
\end{aligned}
$$

$$
\begin{aligned}
&-^A : \mathsf{Con} \to \mathsf{Set} \to \mathsf{Set}\\
&\Gamma^A\, X :\equiv \{A : \mathsf{Ty}\} \to \mathsf{Var}\,\Gamma\, A \to A^A\, X
\end{aligned}
$$

$$
\begin{aligned}
&-^A : \mathsf{Tm}\,\Gamma\, A \to \{X : \mathsf{Set}\} \to \Gamma^A\, X \to A^A\, X\\
&(\mathsf{var}\, x)^A \quad \gamma :\equiv \gamma\, x\\
&(\mathsf{app}\, t\, u)^A\, \gamma :\equiv t^A\, \gamma\, (u^A\, \gamma)
\end{aligned}
$$

$$
\begin{aligned}
&-^A : \mathsf{Sub}\,\Gamma\,\Delta \to \{X : \mathsf{Set}\} \to \Gamma^A\, X \to \Delta^A\, X\\
&\sigma^A\, \gamma\, x :\equiv (\sigma\, x)^A\, \gamma
\end{aligned}
$$

Here, types and contexts depend on some $X : \mathsf{Set}$, which serves as the interpretation of ι. We define Γ^A as a product: for each variable in the context, we get a semantic type. This trick, along with the definition of Sub, makes formalization

a bit more compact. Terms and substitutions are interpreted as natural maps. Substitutions are interpreted by pointwise interpreting the contained terms.

Notation 2. We may write values of Γ^A using notation for Σ-types. For example, we may write $(zero : X) \times (suc : X \to X)$ for the result of computing $\mathsf{NatSig}^A\, X$.

Definition 3. We define **algebras** as follows.

$$\mathsf{Alg} : \mathsf{Con} \to \mathsf{Set}_1$$
$$\mathsf{Alg}\,\Gamma :\equiv (X : \mathsf{Set}) \times \Gamma^A\, X$$

Example 2. $\mathsf{Alg}\,\mathsf{NatSig}$ is computed to $(X : \mathsf{Set}) \times (zero : X) \times (suc : X \to X)$.

2.2.2 Morphisms

Now, we compute notions of morphisms of algebras. In this case, morphisms are functions between underlying sets which preserve all specified structure. The interpretation for calculating morphisms is a *logical relation interpretation* [HRR14] over the $-^A$ interpretation. The key part is the interpretation of types:

$$-^M : (A : \mathsf{Ty})\{X_0\, X_1 : \mathsf{Set}\}(X^M : X_0 \to X_1) \to A^A\, X_0 \to A^A\, X_1 \to \mathsf{Set}$$
$$\iota^M \qquad X^M\, \alpha_0\ \alpha_1 :\equiv X^M\, \alpha_0 = \alpha_1$$
$$(\iota \to A)^M\, X^M\, \alpha_0\ \alpha_1 :\equiv (x : X_0) \to A^M\, X^M\, (\alpha_0\, x)\, (\alpha_1\, (X^M\, x))$$

We again assume an interpretation for the base type ι, as X_0, X_1 and $X^M : X_0 \to X_1$. X^M is function between underlying sets of algebras, and A^M computes what it means that X^M preserves an operation with type A. At the base type, preservation is simply equality. At the first-order function type, preservation is a quantified statement over X_0. We define morphisms for Con pointwise:

$$-^M : (\Gamma : \mathsf{Con})\{X_0\, X_1 : \mathsf{Set}\} \to (X_0 \to X_1) \to \Gamma^A\, X_0 \to \Gamma^A\, X_1 \to \mathsf{Set}$$
$$\Gamma^M\, X^M\, \gamma_0\, \gamma_1 :\equiv \{A : \mathsf{Ty}\}(x : \mathsf{Var}\,\Gamma\, A) \to A^M\, X^M\, (\gamma_0\, x)\, (\gamma_1\, x)$$

For terms and substitutions, we get preservation statements, which are sometimes called *fundamental lemmas* in discussions of logical relations [HRR14].

$$-^M : (t : \mathsf{Tm}\,\Gamma\, A) \to \Gamma^M\, X^M\, \gamma_0\, \gamma_1 \to A^M\, X^M\, (t^A\, \gamma_0)\, (t^A\, \gamma_1)$$
$$(\mathsf{var}\, x)^M\ \ \gamma^M :\equiv \gamma^M\, x$$
$$(\mathsf{app}\, t\, u)^M \gamma^M :\equiv t^M\, \gamma^M\, (u^A\, \gamma_0)$$

$$
\begin{aligned}
&{-^M} : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \Gamma^M\,X^M\,\gamma_0\,\gamma_1 \to \Delta^M\,X^M\,(\sigma^A\,\gamma_0)\,(\sigma^A\,\gamma_1)\\
&\sigma^M\,\gamma^M\,x :\equiv (\sigma\,x)^M\,\gamma^M
\end{aligned}
$$

The definition of $(\mathsf{app}\,t\,u)^M$ is well-typed by the induction hypothesis $u^M\,\gamma^M : X^M\,(u^A\,\gamma_0) = u^A\,\gamma_1$.

Definition 4. To get notions of **algebra morphisms**, we again pack up Γ^M with the interpretation of ι.

$$
\begin{aligned}
&\mathsf{Mor} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Alg}\,\Gamma \to \mathsf{Alg}\,\Gamma \to \mathsf{Set}\\
&\mathsf{Mor}\,\{\Gamma\}\,(X_0,\,\gamma_0)\,(X_1,\,\gamma_1) :\equiv (X^M : X_0 \to X_1) \times \Gamma^M\,X^M\,\gamma_0\,\gamma_1
\end{aligned}
$$

Example 3. We have the following computation:

$$
\begin{aligned}
&\mathsf{Mor}\,\{\mathsf{NatSig}\}\,(X_0,\,\mathit{zero}_0,\,\mathit{suc}_0)\,(X_0,\,\mathit{zero}_1,\,\mathit{suc}_1) :\equiv\\
&\qquad (X^M : X_0 \to X_1)\\
&\quad \times (X^M\,\mathit{zero}_0 = \mathit{zero}_1)\\
&\quad \times ((x : X_0) \to X^M\,(\mathit{suc}_0\,x) = \mathit{suc}_1\,(X^M\,x))
\end{aligned}
$$

Definition 5. We state **initiality** as a predicate on algebras:

$$
\begin{aligned}
&\mathsf{Initial} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Alg}\,\Gamma \to \mathsf{Set}\\
&\mathsf{Initial}\,\{\Gamma\}\,\gamma :\equiv (\gamma' : \mathsf{Alg}\,\Gamma) \to \mathsf{isContr}\,(\mathsf{Mor}\,\gamma\,\gamma')
\end{aligned}
$$

Here $\mathsf{isContr}$ refers to unique existence [Uni13, Section 3.11]. If we drop $\mathsf{isContr}$ from the definition, we get the notion of weak initiality, which corresponds to the recursion principle for Γ. Although we call this predicate $\mathsf{Initial}$, in this chapter we do not yet show that algebras form a category. We will show this in a more general setting in Chapter 4.

Morphisms vs. logical relations. The ${-^M}$ interpretation can be viewed as a special case of logical relations over the ${-^A}$ model: every morphism is a *functional* logical relation, where the chosen relation between the underlying sets happens to be a function. Consider now a more general relational interpretation for types:

$$
\begin{aligned}
&{-^R} : (A : \mathsf{Ty})\{X_0\,X_1 : \mathsf{Set}\}(X^R : X_0 \to X_1 \to \mathsf{Set}) \to A^A\,X_0 \to A^A\,X_1 \to \mathsf{Set}\\
&\iota^R \qquad\quad X^R\,\alpha_0\;\alpha_1 :\equiv X^R\,\alpha_0\,\alpha_1\\
&(\iota \to A)^R\,X^R\,\alpha_0\;\alpha_1 :\equiv (x_0 : X_0)(x_1 : X_1) \to X^R\,x_0\,x_1 \to A^R\,X^R\,(\alpha_0\,x_0)\,(\alpha_1\,x_1)
\end{aligned}
$$

Here, functions are related if they map related inputs to related outputs. If we know that $X^M\,\alpha_0\,\alpha_1 \equiv (f\,\alpha_0 = \alpha_1)$ for some f function, we get

$$(x_0 : X_0)(x_1 : X_1) \to f\,x_0 = x_1 \to A^R\,X^R\,(\alpha_0\,x_0)\,(\alpha_1\,x_1)$$

Now, we can simply substitute along the input equality proof in the above type, to get the previous definition for $(\iota \to A)^M$:

$$(x_0 : X_0) \to A^R\,X^R\,(\alpha_0\,x_0)\,(\alpha_1\,(f\,x_0))$$

This substitution along the equation is called "singleton contraction" in the jargon of homotopy type theory [Uni13]. The ability to perform contraction here is at the heart of the *strict positivity restriction* for inductive signatures. Strict positivity in our setting corresponds to only having first-order function types in signatures. If we allowed function domains to be arbitrary types, in the definition of $(A \to B)^M$ we would only have a black-box $A^M\,X^M : A^A\,X_0 \to A^A\,X_1 \to \mathsf{Set}$ relation, which is not known to be given as an equality.

In Chapter 4 we expand on this. As a preliminary summary: although higher-order functions have relational interpretation, such relations do not generally compose. What we eventually aim to have is a category of algebras and algebra morphisms, where morphisms do compose. We need a *directed* model of the theory of signatures, where every signature becomes a category of algebras. The way to achieve this, is to prohibit higher-order functions, thereby avoiding the polarity issues that prevent a directed interpretation for general function types.

2.2.3 Displayed Algebras

At this point we do not yet have specification for induction principles. We use the term *displayed algebra* to refer to "dependent" algebras, where every displayed algebra component lies over corresponding components in the base algebra. For the purpose of specifying induction, displayed algebras can be viewed as bundles of induction motives and methods.

Displayed algebras over some $\gamma : \mathsf{Alg}\,\Gamma$ are equivalent to slices over γ in the category of Γ-algebras; we will show this in Chapter 4. A slice $f : \mathsf{Sub}\,\Gamma\,\gamma'\,\gamma$ maps elements of γ's underlying set to elements in the base algebra. Why do we need displayed algebras, then? The main reason is that if we are to eventually implement inductive types in a programming language or proof assistant, we need to compute induction principles exactly, not merely up to isomorphisms.

For more illustration of using displayed algebras in a type-theoretic setting, see [AL19]. We adapt the term "displayed algebra" from ibid. as a generalization of displayed categories (and functors, natural transformations) to other algebraic structures.

The displayed algebra interpretation is a *logical predicate* interpretation, defined as follows.

$$
\begin{aligned}
&-^D : (A : \mathsf{Ty})\{X\} \to (X \to \mathsf{Set}) \to A^A\, X \to \mathsf{Set}\\
&\iota^D \qquad\quad X^D\, \alpha :\equiv X^D\, \alpha\\
&(\iota \to A)^D\, X^D\, \alpha :\equiv (x : X)(x^D : X^D\, x) \to A^D\, X^D\, (\alpha\, x)
\end{aligned}
$$

$$
\begin{aligned}
&-^D : (\Gamma : \mathsf{Con})\{X\} \to (X \to \mathsf{Set}) \to \Gamma^A\, X \to \mathsf{Set}\\
&\Gamma^D\, X^D\, \gamma :\equiv \{A : \mathsf{Ty}\}(x : \mathsf{Var}\, \Gamma\, A) \to A^D\, X^D\, (\gamma\, x)
\end{aligned}
$$

$$
\begin{aligned}
&-^D : (t : \mathsf{Tm}\, \Gamma\, A) \to \Gamma^D\, X^D\, \gamma \to A^D\, X^D\, (t^A\, \gamma)\\
&(\mathsf{var}\, x)^D \quad \gamma^D :\equiv \gamma^D\, x\\
&(\mathsf{app}\, t\, u)^D\, \gamma^D :\equiv t^D\, \gamma^D\, (u^A\, \gamma)\, (u^D\, \gamma^D)
\end{aligned}
$$

$$
\begin{aligned}
&-^D : (\sigma : \mathsf{Sub}\, \Gamma\, \Delta) \to \Gamma^D\, X^D\, \gamma \to \Delta^D\, X^D\, (\sigma^A\, \gamma)\\
&\sigma^D\, \gamma^D\, x :\equiv (\sigma\, x)^D\, \gamma^D
\end{aligned}
$$

Analogously to before, everything depends on a predicate interpretation $X^D : X \to \mathsf{Set}$ for ι. For types, a predicate holds for a function if the function preserves predicates. The interpretation of terms is again a fundamental lemma, and we again have pointwise definitions for contexts and substitutions.

Definition 6 (**displayed algebras**)**.**

$$
\begin{aligned}
&\mathsf{DispAlg} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Alg}\, \Gamma \to \mathsf{Set}_1\\
&\mathsf{DispAlg}\, \{\Gamma\}\, (X,\, \gamma) :\equiv (X^D : X \to \mathsf{Set}) \times \Gamma^D\, X^D\, \gamma
\end{aligned}
$$

Example 4. We have the following computation.

$$
\begin{aligned}
&\mathsf{DispAlg}\, \{\mathsf{NatSig}\}\, (X,\, \mathit{zero},\, \mathit{suc}) \equiv\\
&\qquad (X^D \quad : X \to \mathsf{Set})\\
&\qquad \times (\mathit{zero}^D : X^D\, \mathit{zero})\\
&\qquad \times (\mathit{suc}^D \ : (n : X) \to X^D\, n \to X^D\, (\mathit{suc}\, n))
\end{aligned}
$$

2.2.4 Sections

Sections of displayed algebras are "dependent" analogues of algebra morphisms, where the codomain is displayed over the domain.

$$
\begin{aligned}
&-^S : (A : \mathsf{Ty})\{X\ X^D\}(X^S : (x : X) \to X^D\, x) \to (\alpha : A^A\, X) \to A^D\, X^D\, \alpha \to \mathsf{Set}\\
&\iota^S \qquad X^S\, \alpha\ \alpha^D :\equiv X^S\, \alpha = \alpha^D\\
&(\iota \to A)^S\, X^S\, \alpha\ \alpha^D :\equiv (x : X) \to A^S\, X^S\, (\alpha\, x)\, (\alpha^D\, (X^S\, x))
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Con}^S : (\Gamma : \mathsf{Con})\{X\ X^D\}(X^S : (x : X) \to X^D\, x) \to (\gamma : \Gamma^A\, X) \to \Gamma^D\, X^D\, \gamma \to \mathsf{Set}\\
&\Gamma^S\, X^S\, \gamma_0\, \gamma_1 :\equiv \{A : \mathsf{Ty}\}(x : \mathsf{Var}\, \Gamma\, A) \to A^S\, X^S\, (\gamma_0\, x)\, (\gamma_1\, x)
\end{aligned}
$$

$$
\begin{aligned}
&-^S : (t : \mathsf{Tm}\, \Gamma\, A) \to \Gamma^S\, X^S\, \gamma\, \gamma^D \to A^S\, X^S\, (t^A\, \gamma)\, (t^D\, \gamma^D)\\
&(\mathsf{var}\, x)^S \quad \gamma^S :\equiv \gamma^S\, x\\
&(\mathsf{app}\, t\, u)^S \gamma^S :\equiv t^S\, \gamma^S\, (u^A\, \gamma)
\end{aligned}
$$

$$
\begin{aligned}
&-^S : (\sigma : \mathsf{Sub}\, \Gamma\, \Delta) \to \Gamma^S\, X^S\, \gamma\, \gamma^D \to \Delta^S\, X^S\, (\sigma^A\, \gamma)\, (\sigma^A\, \gamma^D)\\
&\sigma^S\, \gamma^S\, x = (\sigma\, x)^S\, \gamma^S
\end{aligned}
$$

Definition 7 (**Displayed algebra sections** ("sections" in short))**.**

$$
\begin{aligned}
&\mathsf{Section} : \{\Gamma : \mathsf{Con}\} \to (\gamma : \mathsf{Alg}\, \Gamma) \to \mathsf{DispAlg}\, \gamma \to \mathsf{Set}\\
&\mathsf{Section}\, (X,\, \gamma)\, (X^D\, \gamma^D) :\equiv (X^S : (x : X) \to X^D\, x) \times \Gamma^S\, X^S\, \gamma\, \gamma^D
\end{aligned}
$$

Example 5. We have the following computation.

$$
\begin{aligned}
\mathsf{Section}\, \{\mathsf{NatSig}\}\, (X,\, \mathit{zero},\, \mathit{suc})\, (X^D,\, \mathit{zero}^D,\, \mathit{suc}^D) \equiv\\
\qquad (X^S \quad : (x : X) \to X^D\, x)\\
\qquad \times (\mathit{zero}^S : X^S\, \mathit{zero} = \mathit{zero}^D)\\
\qquad \times (\mathit{suc}^S \ : (n : X) \to X^S\, (\mathit{suc}\, n) = \mathit{suc}^D\, n\, (X^S\, n))
\end{aligned}
$$

Definition 8 (**Induction**)**.** We define a predicate which holds if an algebra supports induction.

$$
\begin{aligned}
&\mathsf{Inductive} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Alg}\, \Gamma \to \mathsf{Set}_1\\
&\mathsf{Inductive}\, \{\Gamma\}\, \gamma :\equiv (\gamma^D : \mathsf{DispAlg}\, \gamma) \to \mathsf{Section}\, \gamma\, \gamma^D
\end{aligned}
$$

We can observe that $\mathsf{Inductive}\,\{\mathsf{NatSig}\}\,(X, \mathsf{zero}, \mathsf{suc})$ computes exactly to the usual induction principle for natural numbers. The input $\mathsf{DispAlg}$ is a bundle of the induction motive and the methods, and the output $\mathsf{Section}$ contains the X^S eliminator function together with its β-rules.

2.3 Term Algebras

In this section we show that if a type theory supports the inductive types comprising the theory of signatures, it also supports every inductive type which is described by the signatures.

Note that we specified Tm and Sub, but did not need either of them when specifying signatures, or when computing induction principles. That signatures do not depend on terms, is a property specific to simple signatures; this will not be the case in Chapter 4 when we move to more general signatures. However, terms and substitutions are already useful in the construction of term algebras.

The idea is that terms in contexts comprise initial algebras. For example, $\mathsf{Tm}\,\mathsf{NatSig}\,\iota$ is the set of natural numbers (up to isomorphism). Informally, this is because the only way to construct terms is by applying the suc variable (given by $\mathsf{var}\,\mathsf{vz}$) finitely many times to the zero variable (given by $\mathsf{var}\,(\mathsf{vs}\,\mathsf{vz})$).

Definition 9 (**Term algebras**)**.** Fix an $\Omega : \mathsf{Con}$. We abbreviate $\mathsf{Tm}\,\Omega\,\iota$ as T; this will serve as the carrier set of the term algebra. We additionally define the following.

$$
\begin{aligned}
&{-^T} : (A : \mathsf{Ty}) \to \mathsf{Tm}\,\Omega\,A \to A^A\,\mathsf{T}\\
&\iota^T \qquad\quad t :\equiv t\\
&(\iota \to A)^T\,t :\equiv \lambda\,u.\,A^T\,(\mathsf{app}\,t\,u)
\end{aligned}
$$

$$
\begin{aligned}
&{-^T} : (\Gamma : \mathsf{Con}) \to \mathsf{Sub}\,\Omega\,\Gamma \to \Gamma^A\,\mathsf{T}\\
&\Gamma^T\,\nu\,\{A\}\,x :\equiv A^T\,(\nu\,x)
\end{aligned}
$$

$$
\begin{aligned}
&{-^T} : (t : \mathsf{Tm}\,\Gamma\,A)(\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^T\,(t[\nu]) = t^A\,(\Gamma^T\,\nu)\\
&(\mathsf{var}\,x)^T \quad \nu \text{ holds by } \mathsf{refl}\\
&(\mathsf{app}\,t\,u)^T\,\nu \text{ holds by } t^T\,\nu \text{ and } u^T\,\nu
\end{aligned}
$$

$$
\begin{aligned}
&{}^{-T} : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta)(\nu : \mathsf{Sub}\,\Omega\,\Gamma)\{A\}(x : \mathsf{Var}\,\Delta\,A) \\
&\qquad \to \Delta^T\,(\sigma \circ \nu)\,x = \sigma^A\,(\Gamma^T\,\nu)\,x \\
&\sigma^T\,\nu\,x :\equiv (\sigma\,x)^T\,\nu
\end{aligned}
$$

Now we can define the term algebra for Ω itself:

$$
\begin{aligned}
&\mathsf{TmAlg}_\Omega : \mathsf{Alg}\,\Omega \\
&\mathsf{TmAlg}_\Omega :\equiv \Omega^T\,\Omega\,\mathsf{id}
\end{aligned}
$$

In the interpretation for contexts, it is important that Ω is fixed, and we do induction on all Γ contexts such that there is a $\mathsf{Sub}\,\Omega\,\Gamma$. It would not work to try to compute term algebras by direct induction on contexts, because we need to refer to the same T set in the interpretation of every type in a signature.

The interpretation of types embeds terms as A-algebras. At the base type ι, this embedding is simply the identity function, since $\iota^A\,\mathsf{T} \equiv \mathsf{T} \equiv \mathsf{Tm}\,\Omega\,\iota$. At function types we recursively proceed under a semantic λ. The interpretation of contexts is pointwise.

The interpretations of terms and substitutions are coherence properties, which relate the term algebra construction to term evaluation in the ${}^{-A}$ model. For terms, if we pick $\nu \equiv \mathsf{id}$, we get $A^T\,t = t^A\,\mathsf{TmAlg}_\Omega$. The left side embeds t in the term model via ${}^{-T}$, while the right hand side evaluates t in the term model.

One way to view the term algebra construction, is that we are working in a *slice model* over the fixed Ω, and every $\nu : \mathsf{Sub}\,\Omega\,\Gamma$ can be viewed as an internal Γ-algebra in this model. The term algebra construction demonstrates that every such internal algebra yields an external element of Γ^A.

2.3.1 Recursor Construction

We show that TmAlg_Ω supports a recursion principle, i.e. it is weakly initial.

Definition 10 (**Recursor construction**)**.** We assume $(X, \omega) : \mathsf{Alg}\,\Omega$; recall that $X : \mathsf{Set}$ and $\omega : \Omega^A\,X$. We define $\mathsf{R} : \mathsf{T} \to X$ as $\mathsf{R}\,t :\equiv t^A\,\omega$. We additionally define the following.

$$
\begin{aligned}
&{}^{-R} : (A : \mathsf{Ty})(t : \mathsf{Tm}\,\Omega\,A) \to A^M\,\mathsf{R}\,(A^T\,t)\,(t^A\,\omega) \\
&\iota^R \qquad\quad t :\equiv (\mathsf{refl} : t^A\,\omega = t^A\,\omega) \\
&(\iota \to A)^R\,t :\equiv \lambda\,u.\,A^R\,(\mathsf{app}\,t\,u)
\end{aligned}
$$

$$
\begin{aligned}
&{-^R} : (\Gamma : \mathsf{Con})(\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^M\,\mathsf{R}\,(\Gamma^T\,\nu)\,(\nu^A\,\omega)\\
&\Gamma^R\,\nu\,x :\equiv A^R\,(\nu\,x)
\end{aligned}
$$

We define the recursor for Ω as

$$
\begin{aligned}
&\mathsf{Rec}_\Omega : (alg : \mathsf{Alg}\,\Omega) \to \mathsf{Mor}\,\mathsf{TmAlg}_\Omega\,alg\\
&\mathsf{Rec}_\Omega\,(X,\,\omega) :\equiv (\mathsf{R},\,\Omega^R\,\Omega\,\mathsf{id})
\end{aligned}
$$

In short, the way we get recursion is by evaluating terms in arbitrary $(X,\,\omega)$ algebras using $-^A$. The $-^R$ operation for types and contexts confirms that R preserves structure appropriately, so that R indeed yields algebra morphisms.

We skip interpreting terms and substitutions by $-^R$. It is necessary to do so with more general signatures, but not in the current chapter.

2.3.2 Eliminator Construction

We take the idea of the previous section a bit further. We have seen that recursion for term algebras is given by evaluation in the "standard" Set model. Now, we show that induction for term algebras is obtained from the $-^D$ interpretation into the logical predicate model over the Set model.

Definition 11 (**Eliminator construction**)**.** We assume $(X^D,\,\omega^D) : \mathsf{DispAlg}\,\mathsf{TmAlg}_\Omega$. Recall that $X^D : \mathsf{T} \to \mathsf{Set}$ and $\omega^D : \Omega^D\,X^D\,(\Omega^T\,\Omega\,\mathsf{id})$. Like before, we first interpret the underlying set:

$$
\begin{aligned}
&\mathsf{E} : (t : \mathsf{T}) \to X^D\,t\\
&\mathsf{E}\,t :\equiv t^D\,\omega^D
\end{aligned}
$$

However, this definition is not immediately well-typed, since $t^D\,\omega^D$ has type $X^D\,(t^A\,(\Omega^T\,\Omega\,\mathsf{id}))$, so we have to show that $t^A\,(\Omega^T\,\Omega\,\mathsf{id}) = t$. This equation says that nothing happens if we evaluate a term with type ι in the term model. We get it from the $-^T$ interpretation of terms: $t^T\,\mathsf{id} : t[\mathsf{id}] = t^A\,(\Omega^T\,\Omega\,\mathsf{id})$, and we also know that $t[\mathsf{id}] = t$. We interpret types and contexts as well:

$$
\begin{aligned}
&{-^E} : (A : \mathsf{Ty})(t : \mathsf{Tm}\,\Omega\,A) \to A^S\,\mathsf{E}\,(t^A\,(\Omega^T\,\Omega\,\mathsf{id}))\,(t^D\,\omega^D)\\
&\iota^E \qquad\quad t : (t^A\,(\Omega^T\,\Omega\,\mathsf{id}))^D\,\omega^D = t^D\,\omega^D\\
&(\iota \to A)^E\,t :\equiv \lambda\,u.\,A^E\,(\mathsf{app}\,t\,u)
\end{aligned}
$$

$$
\begin{aligned}
&{-^E} : (\Gamma : \mathsf{Con})(\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^S\,\mathsf{E}\,(\nu^A\,(\Omega^T\,\Omega\,\mathsf{id}))\,(\nu^D\,\omega^D)\\
&\Gamma^E\,\nu\,x :\equiv A^E\,(\nu\,x)
\end{aligned}
$$

In ι^E we use the same equation as in the definition of E. In $(\iota \to A)^E$ the definition is well-typed because of the same equation, but instantiated for the abstracted u term this time. All of this amounts to some additional path induction and transport fiddling in the (intensional) Agda formalization. We get induction for Ω as below.

$$
\begin{aligned}
&\mathsf{Ind}_\Omega : (alg : \mathsf{DispAlg}\,\mathsf{TmAlg}_\Omega) \to \mathsf{Section}\,\mathsf{TmAlg}_\Omega\,alg\\
&\mathsf{Ind}_\Omega\,(X^D,\,\omega^D) :\equiv (E,\,\Omega^E\,\Omega\,\mathsf{id})
\end{aligned}
$$

2.4 Discussion & Related Work

2.4.1 Comparison to F-algebras

A well-known alternative way for treating inductive types is to use certain cocontinuous endofunctors as a more semantic notion of signatures.

For example, single-sorted inductive types can be presented as endofunctors which preserve colimits of some ordinal-indexed chains. For instance, if we have a κ-cocontinuous $F : \mathbb{C} \to \mathbb{C}$, then algebras are given as $(X : |\mathbb{C}|) \times (\mathbb{C}(F\,X,\,X))$, morphisms as commuting squares, and Adámek's theorem [AK79] establishes the existence of initial algebras.

An advantage of this approach is that we can describe different classes of signatures by choosing different $\mathbb{C}$ categories:

- If $\mathbb{C}$ is **Set**, we get simple inductive types.
- If $\mathbb{C}$ is $\mathbf{Set}^I$ for some set I, we get indexed inductive types.
- If $\mathbb{C}$ is $\mathbf{Set}/I$, we get inductive-recursive types.

Another advantage of F-algebras is that signatures are a fairly semantic notion: they make sense even if we have no syntactic presentation at hand. That said, often we do need syntactic signatures, for use in proof assistants, or just to have a convenient notation for a class of cocontinuous functors.

An elegant way of carving out a large class of such functors is to consider polynomials as signatures. For example, when working in **Set**, a signature is an

element of $(S : \mathsf{Set}) \times (P : S \to \mathsf{Set})$, and (S, P) is interpreted as a functor as $X \mapsto (s : S) \times (P\, s \to X)$. The initial algebra is the W-type specified by S shapes and P positions. This yields infinitary inductive types as well.

However, it is not known how to get *inductive-inductive* signatures by picking the right $\mathbb{C}$ category and a functor. In an inductive-inductive signature, there may be multiple sorts, which can be indexed over previously declared sorts. For example, in the signature for categories we have $\mathsf{Obj} : \mathsf{Set}$ and $\mathsf{Mor} : \mathsf{Obj} \to \mathsf{Obj} \to \mathsf{Set}$, indexed twice over Obj. Some extensions are required to the idea of F-algebras:

- For inductive-inductive definitions with two sorts, Forsberg gives a specification with two functors, and a considerably more complex notion of algebras, involving dialgebras [NF13][1].
- For an arbitrary number of sorts, Altenkirch et al. [ACD+18] use a "list" of functors, specified mutually with categories of algebras: each functor has as domain the semantic category of all previous sorts.

The functors-as-signatures approach gets significantly less convenient as we consider more general specifications. The approach of this thesis is to skip the middle ground between syntactic signatures and semantic categories of algebras: we treat syntactic signatures as a key component, and give direct semantic interpretation for them. Although we lose the semantic nature of F-algebras, our approach scales extremely well, all the way up to infinitary quotient-inductive-inductive types in Chapter 5, and to some extent to higher inductive-inductive types as well in Chapter 6.

If we look back at $-^A : \mathsf{Con} \to \mathsf{Set} \to \mathsf{Set}$, we may note that Γ^A yields a functor, in fact the same functor (up to isomorphism) that we would get from an F-algebra presentation. However, this is a coincidence in the single-sorted case. With the F-algebra presentation, we can view $(X : |\mathbb{C}|) \times (\mathbb{C}(F\, X, X))$ as specifying the category of algebras as the total category of a displayed category (by viewing the Σ-type here as taking total categories; a Σ in $\mathbf{Cat}$). In our approach, we aim to get the displayed categories directly, without talking about functors.

[1] However, the dialgebra specification only covers restricted signatures, where $B : A \to \mathsf{Set}$ constructor types may refer to $A : \mathsf{Set}$ constructors, but no other dependency is allowed. There is a more general and yet more complicated notion of signature in [NF13], which is not anymore represented with functors.

2.4.2 Generic Programming

Let us consider now our signatures and term algebras in the context of generic programming. This is largely future work, and we do not elaborate it much. But we can draw some preliminary conclusions and make some comparisons.

If a language can formalize inductive signatures and their semantics, that can be viewed as an implementation of generic programming over the described types. Compared to a purely mathematical motivation for this formalization, the requirements for practical generic programming are a bit more stringent.

- *Encoding overhead*: there should be an acceptable overhead in program size and performance when using generic representations. Size blowup can be an issue when writing proofs as well, when types and expressions become too large to mentally parse.
- *Strictness properties*: generic representations should compute as much as possible, ideally in exactly the same way as their non-generic counterparts.

Fixpoints of functors

There is a sizable literature of using fixpoints of functors in generic programming, mainly in Haskell [Swi08, BH11, YHLJ09] and Agda [LM11, AAC+20]. We give a minimal example below for an Agda-like implementation.

We have an inductive syntax for some strictly positive functors, covering essentially the same signatures as Con.

$$\begin{aligned}
\mathsf{Sig} &: \mathsf{Set} \\
\mathsf{Id} &: \mathsf{Sig} \\
\mathsf{K}\top &: \mathsf{Sig} \\
- \otimes - &: \mathsf{Sig} \to \mathsf{Sig} \to \mathsf{Sig} \\
- \oplus - &: \mathsf{Sig} \to \mathsf{Sig} \to \mathsf{Sig}
\end{aligned}$$

Id codes the identity functor, and $\mathsf{K}\top$ codes the functor which is constantly $\top$. $-\otimes-$ and $-\oplus-$ are pointwise products and coproducts respectively. So we have the evident interpretation functions:

$$\begin{aligned}
&[\![-]\!] \ : \mathsf{Sig} \to \mathsf{Set} \to \mathsf{Set} \\
&\mathsf{map} : (F : \mathsf{Sig}) \to (X_0 \to X_1) \to [\![F]\!]\, X_0 \to [\![F]\!]\, X_1
\end{aligned}$$

In Haskell and Agda, the easiest way to get initial algebras is to directly define the inductive fixpoint for each assumed $F : \mathsf{Sig}$:

$$\begin{aligned}\mathsf{Fix_F} &: \mathsf{Set}\\ \mathsf{con_F} &: [\![F]\!]\,\mathsf{Fix_F} \to \mathsf{Fix_F}\end{aligned}$$

In Haskell, this definition is valid for arbitrary *semantic* F functor, because there is no termination checking or positivity checking. In Agda, the above definition is valid because the positivity checker is willing to look inside the definition of $[\![-]\!]$. However, Coq and Lean reject this definition. Next we establish weak initiality, by defining the recursor:

$$\begin{aligned}&\mathsf{rec} : ([\![F]\!]\,X \to X) \to \mathsf{Fix_F} \to X\\ &\mathsf{rec}\,f\,(\mathsf{con}\,x) :\equiv f\,(\mathsf{map}\,(\mathsf{rec}\,f)\,x)\end{aligned}$$

This is again fine in Haskell, but it unfortunately does not pass Agda's termination checker. The most conservative solution is to inline the recursive call into map, so that the definition becomes transparent to termination checking.

$$\begin{aligned}&\mathsf{rec} : ([\![F]\!]\,X \to X) \to \mathsf{Fix_F} \to X\\ &\mathsf{rec}\,f\,(\mathsf{con}\,x) :\equiv f\,(\mathsf{maprec}\,f\,x)\end{aligned}$$

$$\begin{aligned}&\mathsf{maprec} : \{G\} \to ([\![F]\!]\,X \to X) \to [\![G]\!]\,\mathsf{Fix_F} \to [\![G]\!]\,X\\ &\mathsf{maprec}\,\{\mathsf{Id}\}\quad f\,x\quad :\equiv \mathsf{rec}\,f\,x\\ &\mathsf{maprec}\,\{\mathsf{K}\top\}\quad f\,x\quad :\equiv x\\ &\mathsf{maprec}\,\{G \otimes H\}\,f\,(x,\,y)\quad :\equiv (\mathsf{maprec}\,f\,x,\,\mathsf{maprec}\,f\,y)\\ &\mathsf{maprec}\,\{G \oplus H\}\,f\,(\mathsf{inj_1}\,x) :\equiv \mathsf{maprec}\,f\,x\\ &\mathsf{maprec}\,\{G \oplus H\}\,f\,(\mathsf{inj_2}\,x) :\equiv \mathsf{maprec}\,f\,x\end{aligned}$$

Alternatively, we might use *sized types* [AVW17], as in [AAC+20]. The drawback is dependence on an additional language feature which is only supported in Agda, and which has had several soundness issues since its introduction; see e.g. [dev17].

There is yet another possible approach: defining initial algebras as sequential colimits, using Adámek's theorem. This approach was taken by Ahrens, Matthes and Mörtberg in [AMM19]. However, the encoding overhead is excessive, and it is practically unusable for generic programming. Another drawback is that defining colimits requires quotient types, which are often not available natively.

W-types

Given a polynomial $(S,\ P) : (S : \mathsf{Set}) \times (S \to \mathsf{Set})$, the corresponding W-type is inductively specified as below:

$$\begin{aligned} &\mathsf{W}_{S,\,P} : \mathsf{Set} \\ &\mathsf{sup} \quad : (s : S) \to (P\,s \to \mathsf{W}_{S,\,P}) \to \mathsf{W}_{S,\,P} \end{aligned}$$

If we assume $\top$, $\bot$, Bool, Π, Σ, W-types, universes and an intensional identity type, a large class of inductive types can be derived, including infinitary and finitary indexed inductive families; this was shown by Hugunin [Hug21]. The encoding also yields definitional β-rules for recursion and elimination. However, there is also significant encoding overhead here.

- First, there is a translation from more convenient signatures to $(S,\ P)$ polynomials.
- Then, we take the $\mathsf{W}_{S\,P}$ type, but we need to additionally restrict it to the *canonical* elements by a predicate, as in $(x : \mathsf{W}_{S\,P}) \times \mathsf{canonical}\,x$. This is required because the only way to represent inductive branching is by functions, but functions sometimes contain too many elements up to definitional equality. For example, $\bot \to A$ has infinitely many definitonally distinct inhabitants.

Also, there is a performance overhead imposed by the mandatory higher-order constructors.

Term algebras

The main advantage of the term algebras described in this chapter is that they can be defined using plain inductive families, which are available in every major dependently typed language. Even in GHC Haskell, the support for generalized ADTs [JVWW06] is sufficient for implementing term algebras. In contrast, as we have seen, the direct inductive definition of fixpoints is only possible in Agda, while the colimit definition requires quotients and is rather heavyweight. Even in Agda, the drawback of the direct fixpoint definition is that many generic operations only pass termination checking after manual inlining. With term algebras, generic operations can be defined by induction on $\mathsf{Tm}\,\Gamma\,A$ in an obviously terminating way.

For practical usage it makes sense to slightly modify terms. We switch to a *spine neutral* definition. We mutually inductively define Spine and Tm:

$$\begin{aligned}
&\mathsf{Spine} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Set}\\
&\epsilon \quad\;\; : \{A\} \to \mathsf{Spine}\,\Gamma\,A\,A\\
&\text{–,–} \;\; : \{B\,C\} \to \mathsf{Tm}\,\Gamma\,\iota \to \mathsf{Spine}\,\Gamma\,B\,C \to \mathsf{Spine}\,\Gamma\,(\iota \to B)\,C\\
&\mathsf{Tm} \quad : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set}\\
&\text{–}\$\text{–} \;\; : \{A\,B\} \to \mathsf{Var}\,\Gamma\,A \to \mathsf{Spine}\,\Gamma\,A\,B \to \mathsf{Tm}\,\Gamma\,B
\end{aligned}$$

With this representation, a term is always a variable applied to a list of arguments. This can be useful, because pattern matching implementations in metalanguages (e.g. Agda or Idris) are more likely to know about which constructors are possible in patterns. Using this, we give here an ad-hoc definition of natural numbers in pseudo-Agda.

$$\begin{aligned}
&\mathsf{Nat} : \mathsf{Set} && \mathsf{zero} :\equiv \mathsf{vz}\,\$\,\epsilon\\
&\mathsf{Nat} :\equiv \mathsf{Tm}\,(\bullet \triangleright \iota \to \iota \triangleright \iota)\,\iota && \mathsf{suc}\,n :\equiv \mathsf{vs}\,\mathsf{vz}\,\$\,(n,\,\epsilon)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{NatElim} : (P : \mathsf{Nat} \to \mathsf{Set}) \to P\,\mathsf{zero} \to ((n : \mathsf{Nat}) \to P\,n \to P\,(\mathsf{suc}\,n))\\
&\qquad\qquad \to (n : \mathsf{Nat}) \to P\,n\\
&\mathsf{NatElim}\,P\,\mathsf{pz}\,\mathsf{ps}\,(\mathsf{vz}\,\$\,\epsilon) \qquad\quad :\equiv \mathsf{pz}\\
&\mathsf{NatElim}\,P\,\mathsf{pz}\,\mathsf{ps}\,(\mathsf{vs}\,\mathsf{vz}\,\$\,(n,\,\epsilon)) :\equiv \mathsf{ps}\,n\,(\mathsf{NatElim}\,P\,\mathsf{pz}\,\mathsf{ps}\,n)
\end{aligned}$$

The actual Agda definition can be found in the supplementary formalization, and it is pretty much the same as above. We recover the exact same behavior with respect to pattern matching as with native inductive definitions.

Rec_Ω and Ind_Ω can be adapted to spine neutral terms with minor adjustments. But what about the β-rules which they produce as part of their output, are they definitional (i.e. proven by refl)? In this chapter we do not have a rigorous way of reasoning about definitional equalities; in the next chapter we develop such reasoning and show that Rec_Ω enjoys definitional β-rules (with or without the spine neutral definition).

However, Ind_Ω only supports propositional β-rules. The issue is the extra transporting in the definition of $\mathsf{E} : (t : \mathsf{Tm}\,\Omega\,\iota) \to X^D\,t$: transports do not strictly commute with constructors. This appears to be a point of advantage for the direct

fixpoint definition in Agda, as it allows generic elimination with strict computation rules.

The term algebra presentation can be easily extended to indexed families. In that case, signatures and terms are still definable with basic inductive families, without requiring quotients or complicated encodings; see Kaposi and von Raumer [KvR20].

The *sum-of-products* generics by De Vries and Löh [dVL14] is also related. There, signatures for functors are in a normal form: we cannot freely take products and coproducts, instead a signature looks very much like a Con in this chapter (except in an indexed form). The authors observe that several generic programming patterns are easier to express with normalized signatures. However, they still use explicit fixpoints as the way to get initial algebras.

CHAPTER 3

Semantics in Two-Level Type Theory

In this chapter we describe how two-level type theory is used as a metatheoretic setting in the rest of this thesis. First, we provide motivation and overview. Second, we describe models of type theories in general, and models of two-level type theories as extensions. Third, we describe presheaf models of two-level type theories. Finally, we generalize the semantics and the term algebra construction from Chapter 2 in two-level type theory, as a way to illustrate the applications.

3.1 Motivation

We note two shortcomings of the semantics presented in the previous chapter.

First, the semantics that we provided was not as general as it could be. We used the internal Set universe to specify algebras, but algebras make sense in many different categories. A crude way to generalize semantics is to simply say that our formalization, which was carried out in the syntax (i.e. initial model) of some intensional type theory, can be interpreted in any model of the type theory. But this is wasteful: for simple inductive signatures, it is enough to assume a category with finite products as semantic setting. We do not need all the extra baggage that comes with a model of a type theory.

Second, we were not able to reason about definitional equalities, only propositional ones. We have a formalization of signatures and semantics in intensional Agda, where the two notions differ[1], but only propositional equality is subject to internal reasoning. For instance, we would like to show that term algebras support recursion with strict β-rules, and for this we need to reason about strict equality.

[1] As opposed to in extensional type theory, where they are the same.

Notation 3. We use $\bullet$ for the terminal object in a $\mathbb{C}$ category, with $\epsilon : \mathbb{C}(A, \bullet)$ for the unique morphism. For products, we use $- \otimes -$ with $(-,-) : \mathbb{C}(A, B) \to \mathbb{C}(A, C) \to \mathbb{C}(A, B \otimes C)$ and p and q for first and second projections respectively.

Example 6. Assuming a $\mathbb{C}$ category with finite products, we specify natural number algebras and binary tree algebras as follows.

$$\mathsf{Alg}_{\mathsf{NatSig}} :\equiv (X : |\mathbb{C}|) \times \mathbb{C}(\bullet, X) \times \mathbb{C}(X, X)$$
$$\mathsf{Alg}_{\mathsf{TreeSig}} :\equiv (X : |\mathbb{C}|) \times \mathbb{C}(\bullet, X) \times \mathbb{C}(X \otimes X, X)$$

Here, $\mathsf{Alg}_{\mathsf{NatSig}}$ and $\mathsf{Alg}_{\mathsf{TreeSig}}$ are both sets in some metatheory, and the $\times$ in the definitions refer to the metatheoretic Σ. Algebras can be viewed as diagrams which preserve finite products, and algebra morphisms are natural transformations.

How should we adjust Alg from the previous chapter to compute algebras in $\mathbb{C}$, and Mor to compute their morphisms? While it is possible to do this in a direct fashion, working directly with objects and morphisms of $\mathbb{C}$ is rather unwieldy. $\mathbb{C}$ is missing many convenience features of type theories.

- There are no variables or binders. We are forced to work in a point-free style or chase diagrams; both become difficult to handle after a certain point of complexity.
- There are no functions, universes or inductive types.
- Substitution (with weakening as a special case) has to be handled explicitly and manually. Substitutions are certain morphisms, while "terms" are also morphisms, and we have to use composition to substitute terms. In contrast, if we are working internally in a type theory, terms and substitutions are distinct, and we only have to explicitly deal with terms, and substitutions are automated and implicit.

The above overlaps with motivations for working in *internal languages* [nc21] of structured categories: they aid calculation and compact formalization by hiding bureaucratic structural details.

A finite product category $\mathbb{C}$ does not have much of an internal language, it is too bare-bones. But we can work instead in the internal language of $\hat{\mathbb{C}}$, the category of presheaves over $\mathbb{C}$. This allows faithful reasoning about $\mathbb{C}$, while also including all convenience features of extensional type theory.

Two-level type theories [ACKS19], or 2LTT in short, are type theories such that they have "standard" interpretations in presheaf categories. A 2LTT has an inner layer, where types and terms arise by embedding $\mathbb{C}$ in $\hat{\mathbb{C}}$, and an outer layer, where constructions are inherited from $\hat{\mathbb{C}}$. The exact details of the syntax may vary depending on what structures $\mathbb{C}$ supports, and which type formers we assume in the outer layer. Although it is possible to add assumptions to a 2LTT which preclude standard presheaf semantics [ACKS19, Section 2.4.], we stick to basic 2LTT in this thesis. By using 2LTT, we are able to use a type-theoretic syntax which differs only modestly from the style of definitions that we have seen so far.

From a programming perspective, basic 2LTT provides a convenient syntax for writing metaprograms. This can be viewed as *two-stage compilation*: if we have a 2LTT program with an inner type, we can run it, and it returns another program, which lives purely in the inner theory.

3.2 Models of Type Theories

Before explaining 2LTT-specific features, we review models of type theories in general. Variants of 2LTT will be obtained by adding extra features on the top of more conventional TTs.

It is also worth to take a more general look at models at this point, because the notions presented in this subsection (categories with families, type formers) will be reused several times in this thesis, when specifying theories of signatures.

3.2.1 The Algebraic View

We take an algebraic view of models and syntaxes of type theories throughout this thesis. Models of type theories are algebraic structures: they are categories with certain extra structure. The syntax of a type theory is understood to be its initial model. In initial models, the underlying category is the category of typing contexts and parallel substitutions, while the extra structure corresponds to type and term formers, and equations quotient the syntax by definitional equality.

Type theories can be described with quotient inductive-inductive (QII) signatures, and their initial models are quotient inductive-inductive types (QIITs). Hence, 2LTT is also a QII theory. We will first talk about QIITs in Chapter 4. Until then, we shall make do with an informal understanding of categorical semantics for type theories, without using anything in particular from the metatheory of

QIITs. There is some circularity here, that we talk about QIITs in this thesis, but we employ QIITs when talking about them. However, this is only an annoyance in exposition and not a fundamental issue: Sections 4.6 and 5.7 describe how to eliminate circularity by a form of bootstrapping.

The algebraic view lets us dispense with all kinds of "raw" syntactic objects. We only ever talk about well-typed and well-formed objects, moreover, every construction must respect definitional equalities. For terms in the algebraic syntax, definitional equality coincides with metatheoretic equality. This mirrors equality of morphisms in 1-category theory, where we usually reuse metatheoretic equality in the same way.

In the following we specify notions of models for type theories. We split this in two parts: categories with families and type formers.

3.2.2 Categories With Families

Definition 12. A **category with family** (cwf) [Dyb95] is a way to specify the basic structural rules for contexts, substitutions, types and terms. It yields a dependently typed explicit substitution calculus. A cwf consists of the following.

- A category with a terminal object. We denote the set of objects as $\mathsf{Con} : \mathsf{Set}$ and use capital Greek letters starting from Γ to refer to objects. The set of morphisms is $\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$, and we use σ, δ and so on to refer to morphisms. We write id for the identity morphism and $- \circ -$ for composition. The terminal object is $\bullet$ with unique morphism $\epsilon : \mathsf{Sub}\,\Gamma\,\bullet$. In initial models (that is, syntaxes) of type theories, objects correspond to typing contexts, morphisms to parallel substitutions and the terminal object to the empty context; this informs the naming scheme.

- A *family structure*, containing $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Set}$ and $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$. We use A, B, C to refer to types and t, u, v to refer to terms. Ty is a presheaf over the category of contexts and Tm is a displayed presheaf over Ty. This means that types and terms can be substituted:

$$-[-] : \mathsf{Ty}\,\Delta \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma$$
$$-[-] : \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma])$$

  Substitution is functorial: we have $A[\mathsf{id}] \equiv A$ and $A[\sigma \circ \delta] \equiv A[\sigma][\delta]$, and likewise for terms.

A family structure is additionally equipped with *context comprehension* which consists of a context extension operation $- \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$ together with an isomorphism $\mathsf{Sub}\,\Gamma\,(\Delta \triangleright A) \simeq ((\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \times \mathsf{Tm}\,\Gamma\,(A[\sigma]))$ which is natural in Γ.

The following notions are derivable from the comprehension structure:

- By going right-to-left along the isomorphism, we recover *substitution extension* $-, - : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma]) \to \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)$. This means that starting from ϵ or the identity substitution id, we can iterate $-, -$ to build substitutions as lists of terms.

- By going left-to-right, and starting from $\mathsf{id} : \mathsf{Sub}\,(\Gamma \triangleright A)\,(\Gamma \triangleright A)$, we recover the *weakening substitution* $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$ and the *zero variable* $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])$.

- By weakening q, we recover a notion of variables as De Bruijn indices. In general, the n-th De Bruijn index is defined as $\mathsf{q}[\mathsf{p}^n]$, where p^n denotes n-fold composition.

Comprehension can be characterized either by taking $-, -$, p and q as primitive, or the natural isomorphism. The two are equivalent, and we may switch between them, depending on which is more convenient.

There are other ways for presenting the basic categorical structure of models, which are nonetheless equivalent to cwfs, including natural models [Awo18] and categories with attributes [Car78]. We use the cwf presentation for its immediately algebraic character and closeness to conventional explicit substitution syntax.

Notation 4. As De Bruijn indices are hard to read, we will mostly use nameful notation for binders. For example, assuming $\mathsf{Nat} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Ty}\,\Gamma$ and $\mathsf{Id} : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\,\Gamma) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma$, we may write $\bullet \triangleright n : \mathsf{Nat} \triangleright p : \mathsf{Id}\,\mathsf{Nat}\,n\,n$ for a typing context, instead of using numbered variables or cwf combinators as in $\bullet \triangleright \mathsf{Nat} \triangleright \mathsf{Id}\,\mathsf{Nat}\,\mathsf{q}\,\mathsf{q}$.

Notation 5. In the following, we will denote family structures by (Ty,Tm) pairs and overload context extension $- \triangleright -$ for different families.

Definition 13. The following derivable operations are commonly used.

- *Single substitution* can be derived from parallel substitution as follows. Assume $t : \mathsf{Tm}\,(\Gamma \triangleright A)\,B$, and $u : \mathsf{Tm}\,\Gamma\,A$. t is a term which may depend on

the last variable in the context, which has A type. We can substitute that variable with the u term as $t[\mathsf{id}, u] : \mathsf{Tm}\,\Gamma\,(\mathsf{A}[\mathsf{id}, u])$. Note that term substitution causes the type to be substituted as well. $(\mathsf{id}, u) : \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A)$ is well-typed because $u : \mathsf{Tm}\,\Gamma\,A$ hence also $u : \mathsf{Tm}\,\Gamma\,(A[\mathsf{id}])$.

- We can *lift substitutions* over binders as follows. Assuming $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$ and $A : \mathsf{Ty}\,\Delta$, we construct a lifting of σ which maps an additional A-variable to itself: $(\sigma \circ \mathsf{p}, \mathsf{q}) : \mathsf{Sub}\,(\Gamma \triangleright A[\sigma])\,(\Delta \triangleright A)$. Let us see why this is well-typed. We have $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A[\sigma])\,\Gamma$ and $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$, so $\sigma \circ \mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A[\sigma])\,\Delta$. Also, $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A[\sigma])\,(A[\sigma][\mathsf{p}])$, hence $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A[\sigma])\,(A[\sigma \circ \mathsf{p}])$, thus $(\sigma \circ \mathsf{p}, \mathsf{q})$ typechecks.

Notation 6. As a nameful notation for substitutions, we may write $t[x \mapsto u]$, for a single substitution, or $t[x \mapsto u_1, y \mapsto u_2]$ and so on.

In nameful notation we leave all weakening implicit, including substitution lifting. Formally, if we have $t : \mathsf{Tm}\,\Gamma\,A$, we can only mention t in Γ. If we need to mention it in $\Gamma \triangleright B$, we need to use $t[\mathsf{p}]$ instead. In the nameful notation, $t : \mathsf{Tm}\,(\Gamma \triangleright x : B)\,A$ may be used.[2]

3.2.3 Type formers

A family structure in a cwf may be closed under certain type formers, such as functions, Σ-types, universes or inductive types. We give some examples here for their specification. First, we look at common negative type formers, which can be specified using isomorphisms. Then, we consider positive type formers, and finally universes.

Negative types

Definition 14. A $(\mathsf{Ty}, \mathsf{Tm})$ family supports **Π-types** if it supports the following.

$$
\begin{aligned}
\Pi \quad &: (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma \\
\Pi[] \quad &: (\Pi\,A\,B)[\sigma] \equiv \Pi\,(A[\sigma])\,(B[\sigma \circ \mathsf{p}, \mathsf{q}]) \\
\mathsf{app} \quad &: \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to \mathsf{Tm}\,(\Gamma \triangleright A)\,B \\
\mathsf{lam} \quad &: \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)
\end{aligned}
$$

[2]Moreover, when working in the internal syntax of a theory, we just write Agda-like type-theoretic notation, without noting contexts and substitutions in any way.

$$\begin{aligned}
&\Pi\beta \quad : \mathsf{app}\,(\mathsf{lam}\,t) \equiv t\\
&\Pi\eta \quad : \mathsf{lam}\,(\mathsf{app}\,t) \equiv t\\
&\mathsf{lam}[] : (\mathsf{lam}\,t)[\sigma] \equiv \mathsf{lam}\,(t[\sigma \circ \mathsf{p},\, \mathsf{q}])
\end{aligned}$$

Here, Π is the type formation rule. $\mathsf{\Pi}[]$ is the type substitution rule, expressing that substituting Π proceeds structurally on constituent types. Note $B[\sigma \circ \mathsf{p},\, \mathsf{q}]$, where we lift σ over the additional binder.

The rest of the rules specify a natural isomorphism $\mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \simeq \mathsf{Tm}\,(\Gamma \triangleright A)\,B$. We only need a substitution rule (i.e. a naturality rule) for one direction of the isomorphism, since the naturality of the other map is derivable.

This way of specifying Π-types is very convenient if we have explicit substitutions. The usual "pointful" specification is equivalent to this. For example, we have the following derivation of pointful application:

$$\begin{aligned}
&\mathsf{app}' : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (u : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\mathsf{id},\, u])\\
&\mathsf{app}'\,t\,u :\equiv (\mathsf{app}\,t)[\mathsf{id},\, u]
\end{aligned}$$

Remark on naturality. The above specification for Π can be written more compactly if we assume that everything is natural with respect to substitution.

$$\begin{aligned}
&\Pi \qquad\qquad : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma\\
&(\mathsf{app},\, \mathsf{lam}) : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \simeq \mathsf{Tm}\,(\Gamma \triangleright A)\,B
\end{aligned}$$

This is a reasonable assumption; in the rest of the thesis we only ever define structures on cwfs which are natural in this way.

Notation 7. From now on, when specifying type formers in family structures, we assume that everything is natural, and thus omit substitution equations.

There are ways to make this idea more precise, and take it a step further by working in languages where only natural constructions are possible. The term *higher-order abstract syntax* (HOAS) is sometimes used for this style. It lets us also omit contexts, so we would only need to write

$$\begin{aligned}
&\Pi \qquad\qquad : (A : \mathsf{Ty}) \to (\mathsf{Tm}\,A \to \mathsf{Ty}) \to \mathsf{Ty}\\
&(\mathsf{app},\, \mathsf{lam}) : \mathsf{Tm}\,(\Pi\,A\,B) \simeq ((a : \mathsf{Tm}\,A) \to \mathsf{Tm}\,(B\,a))
\end{aligned}$$

Recently several promising works emerged in this area [Uem19, SA21, BKS21]. Although this technology is likely to be the preferred future direction in the metatheory of type theories, this thesis does not make use of it. The field is rather fresh,

with several different approaches and limited amount of pedagogical exposition, and the new techniques would also raise the level of abstraction in this thesis, contributing to making it less accessible. It is also not obvious how exactly HOAS-style could be employed to aid formalization here, and it would require significant additional research. Often, a setup with multiple modalities ("multimodal" [GKNB20]) is required [BKS21], because we work with presheaves over different cwfs. It seems that a synthetic notion of dependent modes would be also required to formalize constructions in this thesis, since we often work with displayed presheaves over displayed cwfs. This is however not yet developed in the literature.

Definition 15. A family structure supports **constant families** if we have the following.

$$\begin{aligned}\mathsf{K} &: \mathsf{Con} \to \{\Gamma : \mathsf{Con}\} \to \mathsf{Ty}\,\Gamma \\ (\mathsf{app}_\mathsf{K},\, \mathsf{lam}_\mathsf{K}) &: \mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta) \simeq \mathsf{Sub}\,\Gamma\,\Delta\end{aligned}$$

Constant families express that every context can be viewed as a non-dependent type in any context. Having constant families is equivalent to the *democracy* property for a cwf [CD14, NF13]. Constant families are convenient when building models, because they let us model non-dependent types as semantic contexts, which are often simpler structures than semantic types. From a programming perspective, constant families specify closed record types, where $\mathsf{K}\,\Delta$ has Δ-many fields.

If we have equalities of sets for the specification, i.e. $\mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta) \equiv \mathsf{Sub}\,\Gamma\,\Delta$, we have **strict constant families**.

Definition 16. A family structure supports **Σ-types** if we have

$$\begin{aligned}\Sigma &: (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma \\ (\mathsf{proj},\, (-,-)) &: \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B) \simeq ((t : \mathsf{Tm}\,\Gamma\,A) \times \mathsf{Tm}\,\Gamma\,(B[\mathsf{id},\, t]))\end{aligned}$$

We use the shorter specification above, where everything is assumed to be natural. We may write proj_1 and proj_2 for composing the metatheoretic first and second projections with proj.

Definition 17. A family structure supports the **unit type** if we have $\top : \mathsf{Ty}\,\Gamma$ such that $\mathsf{Tm}\,\Gamma\,\top \simeq \top$, where the $\top$ on the right is the metatheoretic unit type, and we overload $\top$ for the internal unit type. From this, we get the internal $\mathsf{tt} : \mathsf{Tm}\,\Gamma\,\top$, which is definitionally unique.

Definition 18. A family structure supports **extensional identity** types if there is $\mathsf{Id} : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma$ such that $(\mathsf{reflect}, \mathsf{refl}) : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,t\,u) \simeq (t \equiv u)$.

It is also possible to give a positive definition for identity types, in which case we get intensional identity. Extensional identity corresponds to a categorical equalizer of terms (a limit), while the Martin-Löf-style intensional identity is characterized as the initial reflexive relation on a type (a colimit).

This choice between negative and positive specification generally exists for type formers with a single term construction rule. For example, Σ can be defined as a positive type, with an elimination rule that behaves like pattern matching. Positive Σ is equivalent to negative Σ, although it only supports propositional η-rules. In contrast, positive identity is usually *not* equivalent to negative identity.

$\mathsf{refl} : t \equiv u \to \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,t\,u)$ expresses reflexivity of identity: definitionally equal terms are provably equal. $\mathsf{reflect}$, which goes the other way around, is called *equality reflection*: provably equal terms are identified in the metatheory.

Uniqueness of identity proofs (UIP) is often ascribed to the extensional identity type (see e.g. [Hof95]). UIP means that $\mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,t\,u)$ has at most a single inhabitant up to Id. However, UIP is not something which is inherent in the negative specification, instead it is inherited from the metatheory. If Tm forms a homotopy set in the metatheory, then internal equality proofs inherit uniqueness through the defining isomorphism.

Positive types

We do not dwell much on positive types here, as elsewhere in this thesis we talk a lot about specifying such types anyway. We provide here some background and a small example.

The motivation is to specify initial internal algebras in a cwf. However, specifying the uniqueness of recursors using definitional equality is problematic, if we are to have decidable and efficient conversion checking for a type theory. Consider the specification of Bool together with its recursor.

$$
\begin{aligned}
&\mathsf{Bool} &&: \mathsf{Ty}\,\Gamma\\
&\mathsf{true} &&: \mathsf{Tm}\,\Gamma\,\mathsf{Bool}\\
&\mathsf{false} &&: \mathsf{Tm}\,\Gamma\,\mathsf{Bool}\\
&\mathsf{BoolRec} &&: (B : \mathsf{Ty}\,\Gamma) \to \mathsf{Tm}\,\Gamma\,B \to \mathsf{Tm}\,\Gamma\,B \to \mathsf{Tm}\,\Gamma\,\mathsf{Bool} \to \mathsf{Tm}\,\Gamma\,B
\end{aligned}
$$

$$\mathsf{true}\beta \quad : \mathsf{BoolRec}\, B\, t\, f\, \mathsf{true} \equiv t$$
$$\mathsf{false}\beta \quad : \mathsf{BoolRec}\, B\, t\, f\, \mathsf{false} \equiv f$$

$\mathsf{BoolRec}$ together with the β-rules specifies an internal Bool-algebra morphism. A possible way to specify definitional uniqueness is as follows. Assuming $B : \mathsf{Ty}\,\Gamma$, $t : \mathsf{Tm}\,\Gamma\,B$, $f : \mathsf{Tm}\,\Gamma\,B$ and $m : \mathsf{Tm}\,(\Gamma \triangleright b : \mathsf{Bool})\,B$, such that $m[b \mapsto \mathsf{true}] \equiv t$ and $m[b \mapsto \mathsf{false}] \equiv f$, it follows that $\mathsf{BoolRec}\,B\,t\,f\,b : \mathsf{Tm}\,(\Gamma \triangleright b : \mathsf{Bool})\,B$ is equal to m.

Unfortunately, deciding conversion with this rule entails deciding pointwise equality of arbitrary Bool functions, which can be done in exponential time in the number of Bool arguments. More generally, Scherer presented a decision algorithm for conversion checking with strong finite sums and products in simple type theory [Sch17], which also takes exponential time. If we move to natural numbers with definitionally unique recursion, conversion checking becomes undecidable.

One solution is to have propositionally unique recursion instead. However, if such equations are postulated, that would break the canonicity property in intensional type theories, since now we would have equality proofs other than refl in the empty context.

The standard solution is to have dependent elimination principles instead: this allows inductive reasoning, canonicity and effectively decidable definitional equality at the same time. For Bool, we would have

$$\begin{aligned}\mathsf{BoolInd} : (B : \mathsf{Ty}\,(\Gamma \triangleright b : \mathsf{Bool})) &\to \mathsf{Tm}\,\Gamma\,(B[b \mapsto \mathsf{true}]) \\ &\to \mathsf{Tm}\,\Gamma\,(B[b \mapsto \mathsf{false}]) \to (t : \mathsf{Tm}\,\Gamma\,\mathsf{Bool}) \to \mathsf{Tm}\,\Gamma\,(B[b \mapsto t])\end{aligned}$$

together with $\mathsf{BoolInd}\,B\,t\,f\,\mathsf{true} \equiv t$ and $\mathsf{BoolInd}\,B\,t\,f\,\mathsf{false} \equiv f$.

Of course, if we assume extensional identity types, we have undecidable conversion anyway, and definitionally unique recursion is equivalent to induction. But decidable conversion is a pivotal part of type theory, which makes it possible to relegate a deluge of boilerplate to computers, so decidable conversion should be kept in mind.

Universes

Universes are types which classify types. There are several different flavors of universes.

Definition 19. A **Tarski-style** universe consists of the following data:

$$\mathsf{U} : \mathsf{Ty}\,\Gamma \qquad \mathsf{El} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma$$

This is a weak classifier, since not all $\mathsf{Ty}\,\Gamma$ are necessarily represented as terms of the universe. Rather, this kind of universe can be viewed as an internal sub-family of (Ty, Tm). Like families, Tarski universes can be closed under type formers. For instance, if U has Nat, we have the following:

$$\mathsf{Nat} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \quad \mathit{zero} : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,\mathsf{Nat}) \quad \mathit{suc} : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,\mathsf{Nat}) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,\mathsf{Nat})$$

$$\begin{aligned}\mathsf{NatElim} : \quad & (P : \mathsf{Ty}\,(\Gamma \triangleright n : \mathsf{El}\,\mathsf{Nat})) \\ & \to \mathsf{Tm}\,\Gamma\,(P[n \mapsto \mathit{zero}]) \\ & \to \mathsf{Tm}\,(\Gamma \triangleright n : \mathsf{El}\,\mathsf{Nat} \triangleright np : P[n \mapsto n])\,(P[n \mapsto \mathit{suc}\,n]) \\ & \to (n : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,\mathsf{Nat})) \to \mathsf{Tm}\,\Gamma\,(P[n \mapsto n])\end{aligned}$$

If all type formers in U follow this scheme, U may be called a **weakly Tarski** universe. If we assume that every type former in U is also duplicated in (Ty, Tm), moreover El preserves all type formers, so that e.g. $\mathsf{El}\,\mathsf{Nat}$ is definitionally equal to the natural number type in Ty, then U is **strongly Tarski**.

It is often more convenient to have stronger classifiers as universes, so that *all* types in a given family structure are represented.

Definition 20. Ignoring size issues for now, **Coquand universes** [Coq18] are specified as follows:

$$\mathsf{U} : \mathsf{Ty}\,\Gamma \quad (\mathsf{El},\,\mathsf{c}) : \mathsf{Tm}\,\Gamma\,\mathsf{U} \simeq \mathsf{Ty}\,\Gamma$$

c maps every type in Ty to a code in U. Now we can ignore El when specifying type formers, as c can be always used to get a code in U for a type.

Unfortunately, the exact specification above yields an inconsistent "type-in-type" system, because U itself has a code in U. The standard solution is to have multiple family structures (Ty_i, Tm_i), indexed by universe levels, and have $\mathsf{U}_i : \mathsf{Ty}_{i+1}\,\Gamma$ and $\mathsf{Tm}_{i+1}\,\Gamma\,\mathsf{U}_i \simeq \mathsf{Ty}_i\,\Gamma$. For a general specification of consistent universe hierarchies, see [Kov21]. As mentioned in Section **??**, we omit universe indices in the following, and implicitly assume "just enough" universes for particular purposes.

Definition 21. Russell universes are Coquand universes additionally satisfying $\mathsf{Tm}\,\Gamma\,\mathsf{U} \equiv \mathsf{Ty}\,\Gamma$ as an equality of sets, and also $\mathsf{El}\,t \equiv t$. This justifies omitting El and c from informal notation, implicitly casting between $\mathsf{Tm}\,\Gamma\,\mathsf{U}$ and $\mathsf{Ty}\,\Gamma$.

Russell-style universes are commonly supported in set-theoretic models. They are also often inherited from meta-type-theories which themselves have Russell-universes. Major implementations of type theories (Coq, Lean, Agda, Idris) are all such.

3.3 Two-Level Type Theory

3.3.1 Models

We describe models of 2LTT in the following. This is not the only possible way to present 2LTT; our approach differs from [ACKS19] in some ways. We will summarize the differences at the end of this section.

Definition 22. A model of a **two-level type theory** is a model of type theory such that

- It supports a Tarski-style universe $\mathsf{Ty}_0 : \mathsf{Ty}\,\Gamma$ with decoding $\mathsf{Tm}_0 : \mathsf{Tm}\,\Gamma\,\mathsf{Ty}_0 \to \mathsf{Ty}\,\Gamma$.
- Ty_0 may be closed under arbitrary type formers, however, it is only possible to eliminate from Ty_0 type formers to types in Ty_0.

Types in Ty_0 are called *inner* types, while other types are *outer*. Alternatively, we may talk about *object-level* and *meta-level* types.

For example, if we have inner functions, we have the following:

$$\begin{aligned}
&\Pi_0 && : (A : \mathsf{Tm}\,\Gamma\,\mathsf{Ty}_0) \to \mathsf{Tm}\,(\Gamma \triangleright \mathsf{Tm}_0\,A) \to \mathsf{Tm}\,\Gamma\,\mathsf{Ty}_0\\
&(\mathsf{app}_0, \mathsf{lam}_0) && : \mathsf{Tm}\,\Gamma\,(\mathsf{Tm}_0\,(\Pi_0\,A\,B)) \simeq \mathsf{Tm}\,(\Gamma \triangleright \mathsf{Tm}_0\,A)\,(\mathsf{Tm}_0\,B)
\end{aligned}$$

If we have inner Booleans, we have the following (with β-rules omitted):

$$\begin{aligned}
&\mathsf{Bool}_0 && : \mathsf{Tm}\,\Gamma\,\mathsf{Ty}_0\\
&\mathsf{true}_0 && : \mathsf{Tm}\,\Gamma\,(\mathsf{Tm}_0\,\mathsf{Bool}_0)\\
&\mathsf{false}_0 && : \mathsf{Tm}\,\Gamma\,(\mathsf{Tm}_0\,\mathsf{Bool}_0)\\
&\mathsf{BoolInd}_0 && : (B : \mathsf{Tm}\,(\Gamma \triangleright b : \mathsf{Tm}_0\,\mathsf{Bool}_0)\,\mathsf{Ty}_0)\\
&&& \quad \to \mathsf{Tm}\,\Gamma\,(\mathsf{Tm}_0\,(B[b \mapsto \mathsf{true}_0]))\\
&&& \quad \to \mathsf{Tm}\,\Gamma\,(\mathsf{Tm}_0\,(B[b \mapsto \mathsf{false}_0]))\\
&&& \quad \to (t : \mathsf{Tm}\,\Gamma\,(\mathsf{Tm}_0\,\mathsf{Bool}_0)) \to \mathsf{Tm}\,\Gamma\,(\mathsf{Tm}_0\,(B[b \mapsto t]))
\end{aligned}$$

Intuitively, we can view outer types and terms as metatheoretical, while Ty_0 represents the set of types in the object theory, and Tm_0 witnesses that any object type can be mapped to a metatheoretical set of object terms. The restriction on elimination is crucial. If we have a Boolean term in the object language, we can use the object-level elimination principle to construct new object terms. But it makes no sense to eliminate into the metatheory. In fact, an object-level Boolean term is not necessarily true or false, it can also be just a variable or neutral term in some context, or it can be an arbitrary non-canonical value in a given model.

We review some properties of 2LTT. An important point is the action of Tm_0 on type formers. In general, Tm_0 preserves the negative type formers but not others.

For example, we have the isomorphism $\mathsf{Tm}_0\,(\Pi_0\,A\,B) \simeq \Pi_1\,(\mathsf{Tm}_0\,A)\,(\mathsf{Tm}_0\,B)$, where Π_1 denotes outer functions. We move left-to-right by mapping t to $\mathsf{lam}_1\,(\mathsf{app}_1\,t)$, and the other way by mapping t to $\mathsf{lam}_0\,(\mathsf{app}_0\,t)$. The preservation of Σ, $\top$, K and extensional identity is analogous.

In contrast, we can map from outer positive types to inner ones, but not the other way around. From $b : \mathsf{Tm}\,\Gamma\,\mathsf{Bool}_1$, we can use the outer Bool_1 recursor to return in $\mathsf{Tm}_0\,\mathsf{Bool}_0$. In the other direction, we only have constant functions since the Bool_0 recursor only targets types in Ty_0.

It may be the case that there are universes in the inner layer. For example, disregarding size issues (or just accepting an inconsistent inner theory), there may be an U_0 in Ty_0 such that we have $\mathsf{Tm}\,\Gamma\,(\mathsf{Tm}_0\,\mathsf{U}_0) \equiv \mathsf{Tm}\,\Gamma\,\mathsf{Ty}_0$. This amounts to having a Russell-style inner universe with type-in-type. Assume that we have U_1 as well, as a meta-level Russell universe. Then we can map from $\mathsf{Tm}_0\,\mathsf{U}_0$ to U_1, by taking A to $\mathsf{Tm}_0\,A$, but we cannot map in the other direction.

3.3.2 Internal Syntax and Notation

In the rest of this thesis we will often work internally to a 2LTT, i.e. we use 2LTT as metatheory. We adapt the metatheoretical notations used so far. We list used features and conventions below.

- We keep previous notation for type formers. For instance, Π-types are written as $(x : A) \to B$ or as $A \to B$.
- We assume a Coquand-style universe in the outer layer, named Set. As before, we leave the sizing levels implicit; if we were fully precise, we would

write Set_i for a hierarchy of outer universes. Despite having a Coquand universe, we shall omit encoding and decoding in the internal syntax, and instead work in Russell-style. In practical implementations, elaborating Russell-style notation to Coquand-style is straightforward to do.

- If the same type formers are supported both in the inner and outer layers, we may distinguish them by $_0$ and $_1$ subscripts, e.g. by having Bool_0 and Bool_1. We omit some inferable subscripts, e.g. for Π and Σ-types. In these cases, we usually know from the type parameters which type former is meant. For example, $\mathsf{Tm}_0\,\mathsf{Bool}_0 \to \mathsf{Bool}_1$ can only refer to outer functions.

- We have the convention that $- = -$ refers to the inner equality type, while $- \equiv -$ refers to the outer equality type. If the inner equality is extensional, the choice between $- = -$ and $- \equiv -$ is immaterial, but in Section 3.5 and Chapter 6 we do have intensional inner equality.

- By having Set, we are able to have $\mathsf{Ty}_0 : \mathsf{Set}$ and $\mathsf{Tm}_0 : \mathsf{Ty}_0 \to \mathsf{Set}$. So we do not have to deal with proper meta-level types, and have a more uniform notation. Notation and specification for inner type formers changes accordingly. For example, for inner Π-types we may write $(x : A) \to B$ if $A : \mathsf{Ty}_0$ and B depends on $x : \mathsf{Tm}_0\,A$. This also enables a higher-order specification: if $B : \mathsf{Tm}_0\,A \to \mathsf{Ty}_0$, then $(x : A) \to B\,x : \mathsf{Ty}_0$, and the specifying isomorphism for Π can be written as $\mathsf{Tm}_0\,((x : A) \to B\,x) \simeq ((x : \mathsf{Tm}_0\,A) \to \mathsf{Tm}_0\,(B\,x))$.

  Notation 8. An explicit notation for inner function abstraction would look like $\mathsf{lam}_0\,t$ for $t : (x : \mathsf{Tm}_0\,A) \to \mathsf{Tm}_0\,(B\,x)$. This results in "double" abstraction, e.g. in $\mathsf{lam}_0\,(\lambda\,x.\,\mathsf{suc}_0\,(\mathsf{suc}_0\,x)) : \mathsf{Tm}_0\,(\mathsf{Nat}_0 \to \mathsf{Nat}_0)$. Instead of this, we write $\lambda_0\,x.\,t$ as a notation, thus we write $\lambda_0\,x.\,\mathsf{suc}_0\,(\mathsf{suc}_0\,x)$ for the above example. We may also group multiple λ_0 binders together the same way as with λ.

- We may omit inferable Tm_0 applications. For instance, $\mathsf{Bool}_1 \to \mathsf{Bool}_0$ can be "elaborated" to $\mathsf{Bool}_1 \to \mathsf{Tm}_0\,\mathsf{Bool}_0$ without ambiguity, since the function codomain must be on the same level as the domain, and the only thing we can do to make sense of this is to lift the codomain by Tm_0. Sometimes there is some ambiguity: $(\mathsf{Bool}_0 \to \mathsf{Bool}_0) \to \mathsf{Bool}_1$ can be elaborated both to $\mathsf{Tm}_0\,(\mathsf{Bool}_0 \to \mathsf{Bool}_0) \to \mathsf{Bool}_1$ and to $(\mathsf{Tm}_0\,\mathsf{Bool}_0 \to \mathsf{Tm}_0\,\mathsf{Bool}_0) \to \mathsf{Bool}_1$. However, in this case the two output types are definitionally isomorphic,

because of the Π-preservation by Tm_0. Hence, the elaboration choice does not make much difference, so we may still omit Tm_0-s in situations like this.

Example 7. Working in the internal syntax of 2LTT, the specification of Bool_0 looks like the following (omitting β again):

$$\begin{aligned}
&\mathsf{Bool}_0 &&: \mathsf{Ty}_0\\
&\mathsf{true}_0 &&: \mathsf{Bool}_0\\
&\mathsf{false}_0 &&: \mathsf{Bool}_0\\
&\mathsf{BoolInd}_0 &&: (B : \mathsf{Bool}_0 \to \mathsf{Ty}_0) \to B\,\mathsf{true}_0 \to B\,\mathsf{false}_0 \to (t : \mathsf{Bool}_0) \to B\,t
\end{aligned}$$

If we elaborate the type of $\mathsf{BoolInd}_0$, we get the following:

$$\begin{aligned}
\mathsf{BoolInd}_0 : (B : \mathsf{Tm}_0\,\mathsf{Bool}_0 \to \mathsf{Ty}_0) &\to \mathsf{Tm}_0\,(B\,\mathsf{true}_0) \to \mathsf{Tm}_0\,(B\,\mathsf{false}_0)\\
&\to (t : \mathsf{Tm}_0\,\mathsf{Bool}_0) \to \mathsf{Tm}_0\,(B\,t)
\end{aligned}$$

Here, the type is forced to live in the outer level because of the dependency on Ty_0. Since Ty_0 is an outer type, $\mathsf{Bool}_0 \to \mathsf{Ty}_0$ must be lifted, which in turn requires all other types to be lifted as well.

3.3.3 Alternative Presentation for 2LTT

We digress a bit on a different way to present 2LTT. In the primary 2LTT reference [ACKS19], inner and outer layers are specified as follows. We have two different *family structures* on the base cwf, (Ty_0, Tm_0) and (Ty_1, Tm_1), and a morphism between them. A family morphism is natural transformation mapping types to types and terms to terms, which is an isomorphism on terms. We might name the component maps as follows:

$$\begin{aligned}
&\Uparrow : \mathsf{Ty}_0\,\Gamma \to \mathsf{Ty}_1\,\Gamma\\
&\uparrow : \mathsf{Tm}_0\,\Gamma\,A \to \mathsf{Tm}_1\,\Gamma\,(\Uparrow A)\\
&\downarrow : \mathsf{Tm}_1\,\Gamma\,(\Uparrow A) \to \mathsf{Tm}_0\,\Gamma\,A
\end{aligned}$$

An advantage of this presentation is that we may close (Ty_0, Tm_0) under type formers without any encoding overhead, for example by having $\mathsf{Bool}_0 : \mathsf{Ty}_0\,\Gamma$, $\mathsf{true}_0 : \mathsf{Tm}_0\,\Gamma\,\mathsf{Bool}_0$, etc., without the Tarski-style decoding. On the other hand, we do not automatically get an outer universe of inner types. We can recover that in two ways:

- We can assume an inner universe $\mathsf{U}_0 : \mathsf{Ty}_0\,\Gamma$, which can be lifted to the outer theory as $\Uparrow \mathsf{U}_0$. However, we may not want to make this assumption, in order to keep the inner theory as simple as possible.

- We can assume an outer universe which classifies elements of $\mathsf{Ty}_0\,\Gamma$. This amounts to reproducing the Ty_0 *type* from our 2LTT presentation, as an additional assumption. But in this case, we might as well skip the two family structures and the $\Uparrow$ morphism.

In this thesis we make ubiquitous use of the outer universe of inner types, so we choose that to be the primitive notion, instead of having two family structures.

Do we lose anything by this? For the purposes of this thesis, not really. However, if we want to implement 2LTT as a system for two-stage compilation, the $\Uparrow$ syntax appears to be closer to existing systems. Staging is about computing all outer redexes but no inner ones, thereby outputting syntax which is purely in the inner theory. This could be implemented as a stage-aware variant of normalization-by-evaluation [Abe13, AÖV18, WB18]. We can give an intuitive staging interpretation for the operators in the $\Uparrow$ syntax:

- $\Uparrow A$ is the type of A-expressions. This corresponds to $a\,\mathsf{code}$ in MetaOcaml [Kis14] and $\mathsf{TExp}\,a$ in typed Template Haskell.

- $\uparrow$ is *quoting*, which creates an expression from any inner term. This is $.\langle - \rangle$. in MetaOCaml and $[||-||]$ in typed Template Haskell.

- $\downarrow$ is *splicing*, which inserts the result of a meta-level computation into an object-level expression. This is $\sim(-)$ in MetaOCaml and $\$\$(-)$ in typed Template Haskell.

For example, in the $\Uparrow$ syntax, we might write a polymorphic identity function which acts on inner types in two different ways:

$$\begin{aligned} &\mathsf{id} : (A : \mathsf{U}_0) \to A \to A && \mathsf{id}' : (A : \Uparrow \mathsf{U}_0) \to \Uparrow(\downarrow A) \to \Uparrow(\downarrow A) \\ &\mathsf{id} :\equiv \lambda_0\, A\, x.\, x && \mathsf{id}' :\equiv \lambda_1\, A\, x.\, x \end{aligned}$$

The first one lives in the inner family structure. The second one is the same thing, but lifted to the outer theory. The choice between the two allows us to control staging-time evaluation. If we write $\mathsf{id}\,\mathsf{Bool}_0\,\mathsf{true}_0$, that is an inner expression which goes into the staging output as it is. On the other hand, $\downarrow(\mathsf{id}'\,(\uparrow\,\mathsf{Bool}_0)\,(\uparrow\,\mathsf{true}_0))$

reduces to $\downarrow\,(\uparrow\,\mathsf{true}_0)$ which in turn reduces to true_0. The same choice can be expressed in our syntax as well:

$$\mathsf{id} : \mathsf{Tm}_0\,((A : \mathsf{U}_0) \to A \to A) \qquad \mathsf{id}' : (A : \mathsf{Tm}_0\,\mathsf{U}_0) \to \mathsf{Tm}_0\,A \to \mathsf{Tm}_0\,A$$
$$\mathsf{id} :\equiv \lambda_0\,A\,x.\,x \qquad \mathsf{id}' :\equiv \lambda\,A\,x.\,x$$

It remains to be checked which style is preferable in a staging implementation. In the $\Uparrow$ style, the quoting and splicing operations add noise to core syntax, but they are also mostly inferable during elaboration, and they pack stage-changing information into $\uparrow$ and $\downarrow$, thereby making it feasible to omit stage annotations in other places in the core syntax. In the Ty_0 style, we do not have quote/splice, but we have to keep track of stages in all type/term formers. It would be interesting to compare the two flavors in prototype implementations of staged systems.

3.4 Presheaf Semantics of 2LTT

We review the standard semantics of 2LTT which we use in the rest of the thesis. This justifies the metaprogramming view, that 2LTT allows meta-level reasoning about an inner theory.

We present it two steps, by assuming progressively more structure in the inner theory. First, we only assume a category. This already lets us present a presheaf semantics for the outer layer. Then, we assume a cwf as the inner theory, which lets us interpret Ty_0 and Tm_0 and also consider inner type formers.

3.4.1 Presheaf Model of the Outer Layer

In this subsection we present a presheaf model for the outer layer of 2LTT, that is, the base category together with the terminal object, the $(\mathsf{Ty}, \mathsf{Tm})$ family and some type formers. This presheaf semantics is well-known in the literature [Hof97]. We give a specification which follows [Hub16] most closely.

In the following, we work outside 2LTT (since we are defining a model of 2LTT), in a suitable metatheory; an extensional type theory with enough Set universes suffices.

We assume a $\mathbb{C}$ category. We write $i,\, j,\, k : |\mathbb{C}|$ for objects and $f,\, g,\, h : \mathbb{C}(i,\, j)$ for morphisms. We use a different notation than for cwfs before, in order to disambiguate components in $\mathbb{C}$ from components in the presheaf model of 2LTT.

We use $\hat{\mathbb{C}}$ to refer to the model which is being defined. We use the same component names for $\hat{\mathbb{C}}$ as in Section 3.2.

Model of cwf

Definition 23. $\Gamma : \mathsf{Con}$ is a presheaf over $\mathbb{C}$. Its components are as follows.

$$\begin{aligned}
|\Gamma| \quad & : |\mathbb{C}| \to \mathsf{Set} \\
-\langle - \rangle \quad & : |\Gamma|\, j \to \mathbb{C}(i,\, j) \to |\Gamma|\, i \\
\gamma\langle \mathsf{id} \rangle \quad & \equiv \gamma \\
\gamma\langle f \circ g \rangle \quad & \equiv \gamma\langle f \rangle\langle g \rangle
\end{aligned}$$

We flip around the order of arguments in the action of Γ on morphisms. This is more convenient because of the contravariance; we can observe this in the statement of preservation laws already. The action on morphisms is sometimes called *restriction*.

Definition 24. $\sigma : \mathsf{Sub}\, \Gamma\, \Delta$ is a natural transformation from Γ to Δ. It has action $|\sigma| : |\Gamma|\, i \to |\Delta|\, i$, such that $|\sigma|(\gamma\langle f \rangle) \equiv (|\sigma|\gamma)$.

Definition 25. $A : \mathsf{Ty}\, \Gamma$ is a displayed presheaf over Γ. The "displayed" here is used in exactly the same sense as in "displayed algebra" before. As we will see in Chapter 4, presheaves can be specified with a signature, in which case a presheaf is an algebra, and a displayed presheaf is a displayed algebra. The definition here is equivalent to saying that A is a presheaf over the category of elements of Γ, but it is more convenient to use in concrete definitions and calculations. The components of A are as follows.

$$\begin{aligned}
|A| \quad & : |\Gamma|\, i \to \mathsf{Set} \\
-\langle - \rangle \quad & : |A|\, \gamma \to (f : \mathbb{C}(i,\, j)) \to |A|\, (\gamma\langle f \rangle) \\
\alpha\langle \mathsf{id} \rangle \quad & \equiv \alpha \\
\alpha\langle f \circ g \rangle \quad & \equiv \alpha\langle f \rangle\langle g \rangle
\end{aligned}$$

Definition 26. $t : \mathsf{Tm}\, \Gamma\, A$ is a section of the displayed presheaf A. This is again the same notion of section that we have seen before, instantiated for presheaves.

$$\begin{aligned}
& |t| : (\gamma : |\Gamma|)\, i \to |A|\, \gamma \\
& |t|(\gamma\langle f \rangle) \equiv (|t|\gamma)\langle f \rangle
\end{aligned}$$

Definition 27. $\Gamma \triangleright A : \mathsf{Con}$ is the total presheaf of the displayed presheaf A. Its action on objects and morphisms is the following.

$$\begin{aligned} |\Gamma \triangleright A| &:\equiv (\gamma : |\Gamma|) \times |A\,\gamma| \\ (\gamma,\, \alpha)\langle f \rangle &:\equiv (\gamma\langle f\rangle,\, \alpha\langle f\rangle) \end{aligned}$$

The id and $- \circ -$ preservation laws follow immediately.

Definition 28. $A[\sigma] : \mathsf{Ty}\,\Gamma$ is defined as follows, assuming $A : \mathsf{Ty}\,\Delta$ and $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$.

$$\begin{aligned} |A[\sigma]|\,\gamma &:\equiv |A|\,(|\sigma|\,\gamma) \\ \alpha\langle f\rangle &:\equiv \alpha\langle f\rangle \end{aligned}$$

In the second component, we use $-\langle - \rangle$ for A on the right hand side. The definition is well-typed since $|A|\,(|\sigma|\,(\gamma\langle f\rangle)) \equiv |A|\,((|\sigma|\,\gamma)\langle f\rangle)$ by the naturality of σ. Functoriality follows from functoriality of A.

It is easy to check that the above definitions can be extended to a cwf.

- For the base category, we take the category of presheaves.
- The empty context $\bullet$ is the terminal presheaf, i.e. the constantly $\top$ functor.
- Type substitution is functorial, as it is defined as simple function composition of actions on objects.
- Term substitution is defined as composition of a section and a natural transformation; and also functorial for the same reason.
- Context comprehension structure follows from the Σ-based definition for context extension.

Yoneda embedding

Before continuing with interpreting type formers in $\hat{\mathbb{C}}$, we review the Yoneda embedding, as it is useful in subsequent definitions.

Definition 29. The **Yoneda embedding**, denoted y, is a functor from $\mathbb{C}$ to the underlying category of $\hat{\mathbb{C}}$, defined as follows.

$$\begin{aligned} \mathsf{y} &: |\mathbb{C}| \to \mathsf{Con} & \mathsf{y} &: \mathbb{C}(i,\, j) \to \mathsf{Sub}\,(\mathsf{y}\,i)\,(\mathsf{y}\,j) \\ \mathsf{y}\,i &:\equiv \mathbb{C}(-,\, i) & |\mathsf{y}\,f|\,g &:\equiv f \circ g \end{aligned}$$

Lemma 1 (**Yoneda lemma**)**.** We have $\mathsf{Sub}\,(\mathsf{y}\,i)\,\Gamma \simeq |\Gamma|\,i$ as an isomorphism of sets, natural in i [ML98, Section III.2].

Corollary. If we choose Γ to be $\mathsf{y}j$, it follows that $\mathsf{Sub}\,(\mathsf{y}\,i)\,(\mathsf{y}\,j) \simeq \mathbb{C}(i,\,j)$, i.e. that y is bijective on morphisms; hence it is an embedding.

Notation 9. For $\gamma : |\Gamma|\,i$, we use $\gamma\langle -\rangle : \mathsf{Sub}\,(\mathsf{y}\,i)\,\Gamma$ to denote transporting right-to-left along the Yoneda lemma. In the other direction we do not really need a notation, since from $\sigma : \mathsf{Sub}\,(\mathsf{y}\,i)\,\Gamma$ we get $\sigma\,\mathsf{id} : |\Gamma|\,i$.

Type formers

Definition 30. Constant families are displayed presheaves which do not depend on their context.

$$\begin{aligned}
&\mathsf{K} && : \mathsf{Con} \to \{\Gamma : \mathsf{Con}\} \to \mathsf{Ty}\,\Gamma \\
&|\mathsf{K}\,\Delta|\,\{i\}\,\gamma && :\equiv |\Delta|\,i \\
&\delta\langle f\rangle && :\equiv \delta\langle f\rangle
\end{aligned}$$

With this definition, we have $\mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta) \equiv \mathsf{Sub}\,\Gamma\,\Delta$ so we have strict constant families.

Notation 10. It is useful to consider any set as a constant presheaf, so given $A : \mathsf{Set}$ we may write $A : \mathsf{Con}$ for the constant presheaf as well.

Definition 31. From any $A : \mathsf{Set}$, we get $\mathsf{K}\,A : \mathsf{Ty}\,\Gamma$. This can be used to model negative or positive **closed type formers**. For example, natural numbers are modeled as $\mathsf{K}\,\mathbb{N}$, Booleans as $\mathsf{K}\,\mathsf{Bool}$, the unit type as $\mathsf{K}\,\top$, and so on.

Definition 32. Coquand universes can be defined as follows. We write $\mathsf{Set}_{\hat{\mathbb{C}}}$ for the outer universe in the model, to distinguish it from the external Set. Since the $\mathsf{Set}_{\hat{\mathbb{C}}}$ is a non-dependent type, it is helpful to define it as a $\mathsf{Set}_{\hat{\mathbb{C}}} : \mathsf{Con}$ such that $\mathsf{Sub}\,\Gamma\,\mathsf{Set}_{\hat{\mathbb{C}}} \simeq \mathsf{Ty}\,\Gamma$. The usual universe can be derived from this as $\mathsf{K}\,\mathsf{Set}_{\hat{\mathbb{C}}}$. Again, we ignore size issues; the fully formal definition would involve indexing constructions in $\hat{\mathbb{C}}$ by universe levels.

We can take a hint from the Yoneda lemma. We aim to define $|\mathsf{Set}_{\hat{\mathbb{C}}}|\,i$, but by the Yoneda lemma it is isomorphic to $\mathsf{Sub}\,(\mathsf{y}i)\,\mathsf{Set}_{\hat{\mathbb{C}}}$. However, by specification this

should be isomorphic to $\mathsf{Ty}\,(\mathsf{y}\,i)$, so we take this as definition:

$$\begin{aligned}\mathsf{Set}_{\hat{\mathbb{C}}} \quad &: \mathsf{Con}\\ |\mathsf{Set}_{\hat{\mathbb{C}}}|\,i &:\equiv \mathsf{Ty}\,(\mathsf{y}\,i)\\ A\langle f\rangle \quad &:\equiv A[\mathsf{y}f]\end{aligned}$$

In the $A\langle f\rangle$ definition, we substitute $A : \mathsf{Ty}\,(\mathsf{y}\,i)$ with $\mathsf{y}f : \mathsf{Sub}\,(\mathsf{y}j)\,(\mathsf{y}i)$ to get an element of $\mathsf{Ty}\,(\mathsf{y}j)$. The required $\mathsf{Sub}\,\Gamma\,\mathsf{Set}_{\hat{\mathbb{C}}} \simeq \mathsf{Ty}\,\Gamma$ is straightforward, so we omit the definition.

We note that Russell universes are not supported in the outer layer, as $\mathsf{Sub}\,\Gamma\,\mathsf{Set}_{\hat{\mathbb{C}}}$ and $\mathsf{Ty}\,\Gamma$ are not strictly the same, in particular they have a different number of components as iterated Σ-types. Nevertheless, as we mentioned in Section 3.3.2, we use Russell-style notation in the internal 2LTT syntax, and assume that encoding/decoding is inserted by elaboration.

Definition 33. Σ-types are defined pointwise. The definitions for pairing and projections follow straightforwardly.

$$\begin{aligned}\Sigma \qquad\qquad &: (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma\\ |\Sigma\,A\,B|\,\gamma &:\equiv (\alpha : |A|\,\gamma) \times |B|\,(\gamma,\,\alpha)\\ (\alpha,\,\beta)\langle f\rangle &:\equiv (\alpha\langle f\rangle,\,\beta\langle f\rangle)\end{aligned}$$

Definition 34. We define **Π-types** in the following. This is a bit more complicated, so first we look at the simpler case of presheaf exponentials. We source this example from [MM12, Section I.]. The reader may refer to ibid. for an overview of constructions in presheaf categories.

The exponential $\Delta^\Gamma : \mathsf{Con}$ is characterized by the isomorphism $\mathsf{Sub}\,(\Gamma \otimes \Delta)\,\Xi \simeq \mathsf{Sub}\,\Gamma\,(\Xi^\Delta)$, where we write $\otimes$ for the pointwise product of two presheaves. We can again use the Yoneda lemma. We want to define $|\Delta^\Gamma|\,i$, but this is isomorphic to $\mathsf{Sub}\,(\mathsf{y}i)\,(\Delta^\Gamma)$, which should be isomorphic to $\mathsf{Sub}\,(\mathsf{y}i \otimes \Gamma)\,\Delta$ by the specification of exponentials. Hence:

$$\begin{aligned}|\Delta^\Gamma|\,i &:\equiv \mathsf{Sub}\,(\mathsf{y}i \otimes \Gamma)\,\Delta\\ \sigma\langle f\rangle \quad &:\equiv \sigma \circ (\mathsf{y}f \circ \mathsf{p},\,\mathsf{q})\end{aligned}$$

In the definition of presheaf restriction, we use p, q as projections and $-,-$ as pairing for $\otimes$. In short, $(\mathsf{y}f \circ \mathsf{p},\,\mathsf{q})$ is the same as the morphism lifting from Definition 13: it weakens $\mathsf{y}f : \mathsf{Sub}\,(\mathsf{y}j)\,(\mathsf{y}i)$ to $\mathsf{Sub}\,(\mathsf{y}j \otimes \Gamma)\,(\mathsf{y}i \otimes \Gamma)$.

The dependently typed case follows the same pattern, except that we use Tm and $-\triangleright-$ instead of Sub and $-\otimes-$. Additionally, the action on objects depends on $\gamma : |\Gamma|\, i$, and we make use of $\gamma\langle - \rangle \ : \mathsf{Sub}\,(\mathsf{y}i)\,\Gamma$ (introduced in Notation 9).

$$\begin{aligned}
&\Pi && : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma \\
&|\Pi\, A\, B|\,\{i\}\,\gamma && :\equiv \mathsf{Tm}\,(\mathsf{y}i \triangleright A[\gamma\langle -\rangle])\,(B[\gamma\langle -\rangle \circ \mathsf{p},\, \mathsf{q}]) \\
&t\langle f\rangle && :\equiv t[\mathsf{y}f \circ \mathsf{p},\, \mathsf{q}]
\end{aligned}$$

Let us unfold the above definition a bit. Assuming $t : |\Pi\, A\, B|\,\{i\}\,\gamma$, we have

$$|t| : \{j : |\mathbb{C}|\} \to ((f,\, \alpha) : (f : \mathbb{C}(j,\, i)) \times |A|\,(\gamma\langle f\rangle)) \to |B|\,(\gamma\langle f\rangle,\, \alpha)$$

This is a bit clearer if we remove the Σ-type by currying.

$$|t| : \{j : |\mathbb{C}|\}(f : \mathbb{C}(j,\, i))(\alpha : |A|\,(\gamma\langle f\rangle)) \to |B|\,(\gamma\langle f\rangle,\, \alpha)$$

Restriction is functorial since it is defined as Tm substitution. The definitions for lam and app are left to the reader.

Definition 35. Extensional identity is defined as pointwise equality of sections:

$$\begin{aligned}
&\mathsf{Id} : \mathsf{Tm}\,\Gamma\, A \to \mathsf{Tm}\,\Gamma\, A \to \mathsf{Ty}\,\Gamma \\
&|\mathsf{Id}\, t\, u|\,\gamma :\equiv |t|\,\gamma \equiv |u|\,\gamma
\end{aligned}$$

For the restriction operation, we have to show that $|t|\,\gamma \equiv |u|\,\gamma$ implies $|t|\,(\gamma\langle f\rangle) \equiv |u|\,(\gamma\langle f\rangle)$. This follows from congruence by $-\langle f\rangle$ and naturality of t and u. The defining $(\mathsf{reflect},\, \mathsf{refl}) : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\, t\, u) \simeq (t \equiv u)$ isomorphism is evident, assuming UIP and function extensionality for the metatheoretic $-\equiv-$ relation (which we do assume).

3.4.2 Modeling the Inner Layer

We assume now that $\mathbb{C}$ is a cwf. We write types as a, b, $c : \mathsf{Ty}_{\mathbb{C}}\, i$ and terms as t, u, $v : \mathsf{Tm}_{\mathbb{C}}\, i\, a$. We reuse $\bullet$ for the terminal object and $-\triangleright-$ for context extension, and likewise reuse notation for substitutions.

Definition 36 (Ty_0**,** Tm_0)**.** First, note that $\mathsf{Ty}_{\mathbb{C}}$ is a presheaf over $\mathbb{C}$, and $\mathsf{Tm}_{\mathbb{C}}$ is a displayed presheaf over $\mathsf{Ty}_{\mathbb{C}}$; this follows from the requirement that they form a family structure over $\mathbb{C}$. Hence, in the presheaf model $\mathsf{Ty}_{\mathbb{C}}$ is an element of Con

and $\mathsf{Tm}_{\mathbb{C}}$ is an element of $\mathsf{Ty}\,\mathsf{Ty}_{\mathbb{C}}$. Also recall from Definition 30 that $\mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta) \equiv \mathsf{Sub}\,\Gamma\,\Delta$. With this is mind, we give the following definitions:

$$\begin{aligned} &\mathsf{Ty}_0 : \mathsf{Ty}\,\Gamma && \mathsf{Tm}_0 : \mathsf{Tm}\,\Gamma\,\mathsf{Ty}_0 \to \mathsf{Ty}\,\Gamma \\ &\mathsf{Ty}_0 :\equiv \mathsf{K}\,\mathsf{Ty}_{\mathbb{C}} && \mathsf{Tm}_0\,A :\equiv \mathsf{Tm}_{\mathbb{C}}[A] \end{aligned}$$

$\mathsf{Tm}_{\mathbb{C}}[A]$ is well-typed since $A : \mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\mathsf{Ty}_{\mathbb{C}})$, thus $A : \mathsf{Sub}\,\Gamma\,\mathsf{Ty}_{\mathbb{C}}$. In other words, A is a natural transformation from Γ to the presheaf of inner types.

Inner type formers

Can type formers in $(\mathsf{Ty}_{\mathbb{C}},\, \mathsf{Tm}_{\mathbb{C}})$ be transferred to $(\mathsf{Ty}_0,\, \mathsf{Tm}_0)$ in the presheaf model of 2LTT? For example, if $\mathbb{C}$ supports Bool, we would like to model Bool_0 in Ty_0 as well. The following explanation is adapted from Capriotti [Cap17, Section 2.3].

Generally, a type former in $\mathbb{C}$ transfers to $\hat{\mathbb{C}}$ if it can be specified in the internal language of $\hat{\mathbb{C}}$; if the type former "always has been" in $\hat{\mathbb{C}}$ to begin with. To be describable in $\hat{\mathbb{C}}$, a type former needs to be natural with respect to $\mathbb{C}$ morphisms. This is also a core idea of HOAS: when working in $\hat{\mathbb{C}}$, everything is natural, and we can omit boilerplate related to contexts and substitutions. For example, consider the specification of inner Π-types in the internal syntax of 2LTT:

$$\begin{aligned} &\Pi_0 && : (A : \mathsf{Ty}_0) \to (\mathsf{Tm}_0\,A \to \mathsf{Ty}_0) \to \mathsf{Ty}_0 \\ &(\mathsf{app}_0,\, \mathsf{lam}_0) && : \Pi_0\,A\,B \simeq ((a :\ \mathsf{Tm}_0\,A) \to\ \mathsf{Tm}_0\,(B\,a)) \end{aligned}$$

We can say that this *defines* what it means for $\mathbb{C}$ to support Π. We recover the usual non-higher-order specification of Π in the following way, up to isomorphism:

- First, we interpret the higher-order specification as a context or closed Σ-type in the standard presheaf model of 2LTT. This yields a presheaf over $\mathbb{C}$.
- Then, we evaluate the resulting presheaf at the terminal object $\bullet$. This yields a set which is isomorphic to the conventional specification of Π.

In summary, if by "type formers" we mean extra structure on $(\mathsf{Ty}_0,\, \mathsf{Tm}_0)$ which is definable in 2LTT, then *by definition* all such type formers transfer from $\mathbb{C}$ to $(\mathsf{Ty}_0,\, \mathsf{Tm}_0)$. This holds for every type former mentioned in this thesis.

3.4.3 Functions With Inner Domains

There is a useful semantic simplification in the standard presheaf model, in cases where we have functions of the form $\Pi\,(\mathsf{Tm}_0\,A)\,B$. This greatly reduces encoding overhead when interpreting inductive signatures in 2LTT; we look at examples in Section 3.5. First we look at the simply-typed case with presheaf exponentials.

Lemma 2. y preserves finite products up to isomorphism, i.e. $\mathsf{y}\bullet \simeq \bullet$ and $\mathsf{y}(i \otimes j) \simeq (\mathsf{y}i \otimes \mathsf{y}j)$.

Proof. $\mathsf{y}\bullet$ is $\mathbb{C}(-, \bullet)$ by definition, which is pointwise isomorphic to $\top$, hence isomorphic to $\bullet \equiv \mathsf{K}\,\top$. $\mathsf{y}(i \otimes j)$ is $\mathbb{C}(-, i \otimes j)$, which is isomorphic to $\mathsf{y}i \otimes \mathsf{y}j$ by the specification of products. □

Lemma 3. We have the following isomorphism.

$$
\begin{array}{ll}
|\Gamma^{\mathsf{y}i}|\,j \equiv & \\
\mathsf{Sub}\,(\mathsf{y}j \otimes \mathsf{y}i)\,\Gamma \simeq & \text{by product preservation} \\
\mathsf{Sub}\,(\mathsf{y}(j \otimes i))\,\Gamma \simeq & \text{by Yoneda lemma} \\
|\Gamma|\,(j \otimes i) &
\end{array}
$$

It is possible to rephrase the above derivation for Π-types. For that, we would need to define the action of y on types and terms, consider the preservation of $- \triangleright -$ by y, and also specify a "dependent" Yoneda lemma for Tm. For the sake of brevity, we omit this, and present the result directly:

$$|\Pi\,(\mathsf{Tm}_0\,A)\,B|\,\{i\}\,\gamma \simeq |B|\,\{i \triangleright |A|\,\gamma\}\,(\gamma\langle\mathsf{p}\rangle,\,\mathsf{q})$$

In short, depending on an inner domain is the same as depending on an extended context in $\mathbb{C}$. We expand a bit on the typing of the right hand side. We have $\gamma : |\Gamma|\,i$, moreover

$$
\begin{array}{ll}
|B| & : \{j : \mathbb{C}\} \to |\Gamma \ \triangleright \mathsf{Tm}_0\,A|\,j \to \mathsf{Set} \\
|B| & : \{j : \mathbb{C}\} \to ((\gamma' : |\Gamma|\,j) \times \mathsf{Tm}_{\mathbb{C}}\,j\,(|A|\,\gamma')) \to \mathsf{Set} \\
|B|\,\{i \triangleright |A|\,\gamma\} & : ((\gamma' : |\Gamma|\,(i \triangleright |A|\,\gamma)) \times \mathsf{Tm}_{\mathbb{C}}\,(i \triangleright |A|\,\gamma)\,(|A|\,\gamma')) \to \mathsf{Set} \\
\gamma\langle\mathsf{p}\rangle & : |\Gamma|\,(i \triangleright |A|\,\gamma) \\
\mathsf{q} & : \mathsf{Tm}_{\mathbb{C}}\,(i \triangleright |A|\,\gamma)\,((|A|\,\gamma)[\mathsf{p}]) \\
\mathsf{q} & : \mathsf{Tm}_{\mathbb{C}}\,(i \triangleright |A|\,\gamma)\,(|A|\,(\gamma\langle\mathsf{p}\rangle))
\end{array}
$$

3.5 Simple Signatures in 2LTT

We revisit simple inductive signatures in this section, working internally to 2LTT. We review the concepts introduced in Chapter 2 in the same order.

Notation 11. In this section we shall be fairly explicit about writing Tm_0-s and transporting along definitional isomorphisms. The simple setting makes it feasible to be explicit; in later chapters we are more terse, as signatures and semantics get more complicated.

3.5.1 Theory of Signatures

Signatures are defined exactly in the same way as before: we have $\mathsf{Con} : \mathsf{Set}$, $\mathsf{Ty} : \mathsf{Set}$, $\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$, $\mathsf{Var} : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set}$ and $\mathsf{Tm} : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set}$. However, now by Set we mean the outer universe of 2LTT. Thus signatures are inductively defined in the outer layer.

3.5.2 Algebras

Again we compute algebras by induction on signatures, but now we use inner types for carriers of algebras. We interpret types as follows:

$$\begin{aligned}
&-^A : \mathsf{Ty} \to \mathsf{Ty}_0 \to \mathsf{Set}\\
&\iota^A \qquad\quad X :\equiv \mathsf{Tm}_0\, X\\
&(\iota \to A)^A\, X :\equiv \mathsf{Tm}_0\, X \to A^A\, X
\end{aligned}$$

Elsewhere, we change the type of the X parameters accordingly:

$$\begin{aligned}
&-^A : \mathsf{Con} \to \mathsf{Ty}_0 \to \mathsf{Set}\\
&-^A : \mathsf{Var}\, \Gamma\, A \to \{X : \mathsf{Ty}_0\} \to \Gamma^A\, X \to A^A\, X\\
&-^A : \mathsf{Tm}\, \Gamma\, A \to \{X : \mathsf{Ty}_0\} \to \Gamma^A\, X \to A^A\, X\\
&-^A : \mathsf{Sub}\, \Gamma\, \Delta \to \{X : \mathsf{Ty}_0\} \to \Gamma^A\, X \to \Delta^A\, X
\end{aligned}$$

We also define $\mathsf{Alg}\, \Gamma$ as $(X : \mathsf{Ty}_0) \times \Gamma^A\, X$.

Example 8. Inside 2LTT we have the following:[3]

$$\mathsf{Alg}\, \mathsf{NatSig} \equiv (X : \mathsf{Ty}_0) \times (\mathit{zero} : \mathsf{Tm}_0\, X) \times (\mathit{suc} : \mathsf{Tm}_0\, X \to \mathsf{Tm}_0\, X)$$

[3]Up to isomorphism, since we previously defined Γ^A as a function type instead of an iterated product type.

Then, we may assume any cwf $\mathbb{C}$, and interpret the above closed type in the presheaf model $\hat{\mathbb{C}}$, and evaluate the result at $\bullet$ and the unique element of the terminal presheaf $\mathsf{K}\,\top$:

$$|\mathsf{Alg}\,\mathsf{NatSig}|\,\{\bullet\}\,\mathsf{tt} : \mathsf{Set}$$

We compute the definitions now. We use the simplified semantics for $suc : \mathsf{Tm}_0\,X \to \mathsf{Tm}_0\,X$, since the function domain is an inner type.

$$|\mathsf{Alg}\,\mathsf{NatSig}|\,\{\bullet\}\,\mathsf{tt} \equiv (X : \mathsf{Ty}_{\mathbb{C}}\,\bullet) \times (\mathit{zero} : \mathsf{Tm}_{\mathbb{C}}\,\bullet\,X) \times (\mathit{suc} : \mathsf{Tm}_{\mathbb{C}}\,(\bullet \triangleright X)\,X)$$

Using the same computation, we get the following for binary trees:

$$|\mathsf{Alg}\,\mathsf{TreeSig}|\,\{\bullet\}\,\mathsf{tt} \equiv (X : \mathsf{Ty}_{\mathbb{C}}\,\bullet) \times (\mathit{leaf} : \mathsf{Tm}_{\mathbb{C}}\,\bullet\,X) \times (\mathit{node} : \mathsf{Tm}_{\mathbb{C}}\,(\bullet \triangleright X \triangleright X)\,X)$$

We can also get internal algebras in any $\mathbb{C}$ category with finite products, because we can build cwfs from all such $\mathbb{C}$.

Definition 37. Assuming $\mathbb{C}$ with finite products, we build a cwf by setting $\mathsf{Con} :\equiv |\mathbb{C}|$, $\mathsf{Ty}\,\Gamma :\equiv |\mathbb{C}|$, $\mathsf{Sub}\,\Gamma\,\Delta :\equiv \mathbb{C}(\Gamma,\,\Delta)$, $\mathsf{Tm}\,\Gamma\,A :\equiv \mathbb{C}(\Gamma,\,A)$, $\Gamma \triangleright A :\equiv \Gamma \otimes A$ and $\bullet :\equiv \bullet_{\mathbb{C}}$. In short, we build a non-dependent (simply-typed) cwf.

Now we can effectively interpret signatures in finite product categories. For example:

$$|\mathsf{Alg}\,\mathsf{NatSig}|\,\{\bullet\}\,\mathsf{tt} \equiv (X : |\mathbb{C}|) \times (\mathit{zero} : \mathbb{C}(\bullet,\,X)) \times (\mathit{suc} : \mathbb{C}(\bullet \otimes X,\,X))$$

This is almost the same as what we would write by hand for the specification of natural number objects; the only difference is the extra $\bullet \otimes -$ in *suc*.

3.5.3 Morphisms

We get an additional degree of freedom in the computation of morphisms: preservation equations can be inner or outer. The former option is *weak* or *propositional* preservation, while the latter is *strict* preservation. In the presheaf model of 2LTT, outer equality is definitional equality of inner terms, while inner equality is propositional equality in the inner theory. Of course, if the inner theory has extensional identity type, weak and strict equations in 2LTT are equivalent for inner types. We compute weak preservation for types as follows.

$$-^M : (A : \mathsf{Ty})\{X_0\,X_1 : \mathsf{Ty}_0\}(X^M : \mathsf{Tm}_0\,X_0 \to \mathsf{Tm}_0\,X_1) \to A^A\,X_0 \to A^A\,X_1 \to \mathsf{Set}$$

$$\iota^M \qquad X^M\,\alpha_0\,\alpha_1 :\equiv \mathsf{Tm}_0\,(X^M\,\alpha_0 = \alpha_1)$$

$$(\iota \to A)^M\,X^M\,\alpha_0\,\alpha_1 :\equiv (x : \mathsf{Tm}_0\,X_0) \to A^M\,X^M\,(\alpha_0\,x)\,(\alpha_1\,(X^M\,x))$$

For strict preservation, we simply change $\mathsf{Tm}_0\,(X^M\,\alpha_0 = \alpha_1)$ to $X^M\,\alpha_0 \equiv \alpha_1$. The definition of morphisms is the same as before:

$$-^M : (\Gamma : \mathsf{Con}_1)\{X_0\,X_1 : \mathsf{Ty}_0\} \to (\mathsf{Tm}_0\,X_0 \to \mathsf{Tm}_0\,X_1) \to \Gamma^A\,X_0 \to \Gamma^A\,X_1 \to \mathsf{Set}$$
$$\Gamma^M\,X^M\,\gamma_0\,\gamma_1 :\equiv \{A\}(x : \mathsf{Var}_1\,\Gamma\,A) \to A^M\,X^M\,(\gamma_0\,x)\,(\gamma_1\,x)$$

$$\mathsf{Mor} : \{\Gamma : \mathsf{Con}_1\} \to \mathsf{Alg}\,\Gamma \to \mathsf{Alg}\,\Gamma \to \mathsf{Set}$$
$$\mathsf{Mor}\,\{\Gamma\}\,(X_0,\,\gamma_0)\,(X_1,\,\gamma_1) :\equiv (X^M : \mathsf{Tm}_0\,X_0 \to \mathsf{Tm}_0\,X_1) \times \Gamma^M\,X^M\,\gamma_0\,\gamma_1$$

We omit here the $-^M$ definitions for terms and substitutions.

3.5.4 Displayed Algebras

We present $-^D$ only for types below.

$$-^D : (A : \mathsf{Ty})\{X\} \to (\mathsf{Tm}_0\,X \to \mathsf{Ty}_0) \to A^A\,X \to \mathsf{Set}$$
$$\iota^D \qquad X^D\,\alpha :\equiv \mathsf{Tm}_0\,(X^D\,\alpha)$$
$$(\iota \to A)^D\,X^D\,\alpha :\equiv (x : \mathsf{Tm}_0\,X)(x^D : \mathsf{Tm}_0\,(X^D\,x)) \to A^D\,X^D\,(\alpha\,x)$$

Note that in the presheaf model, inhabitants of $\mathsf{Tm}_0\,X \to \mathsf{Ty}_0$ are inner types depending on contexts extended with the interpretation of X.

Example 9. Assume a closed $(X,\,\mathit{zero},\,\mathit{suc})$ Nat-algebra in 2LTT. We have the following computation:

$$\begin{aligned}&\mathsf{DispAlg}\,\{\mathsf{NatSig}\}\,(X,\,\mathit{zero},\,\mathit{suc}) \equiv\\ &\qquad (X^D \quad : \mathsf{Tm}_0\,X \to \mathsf{Ty}_0)\\ &\qquad \times (\mathit{zero}^D : \mathsf{Tm}_0\,(X^D\,\mathit{zero}))\\ &\qquad \times (\mathit{suc}^D \quad : (n : \mathsf{Tm}_0\,X) \to \mathsf{Tm}_0\,(X^D\,n) \to \mathsf{Tm}_0\,(X^D\,(\mathit{suc}\,n)))\end{aligned}$$

Let us look at the presheaf interpretation. We simplify functions with inner domains everywhere. Also note that for $\mathsf{suc} : \mathsf{Tm}_0\,X \to \mathsf{Tm}_0\,X$, we get $|\mathsf{suc}|\,\mathsf{tt} : \mathsf{Tm}_{\mathbb{C}}\,(\bullet \triangleright n : |X|\,\mathsf{tt})\,(|X|\,\mathsf{tt})$ in the semantics, so a $\mathsf{suc}\,t$ application is translated as a substitution $(|\mathsf{suc}|\,\mathsf{tt})[n \mapsto |t|\,\mathsf{tt}]$.

$$\begin{aligned}&|\mathsf{DispAlg}\,\{\mathsf{NatSig}\}\,(X,\,\mathit{zero},\,\mathit{suc})|\,\{\bullet\}\,\mathsf{tt} \equiv\\ &(X^D \quad : \mathsf{Ty}_{\mathbb{C}}\,(\bullet \triangleright n : |X|\,\mathsf{tt}))\\ &\times (\mathit{zero}^D : \mathsf{Tm}_{\mathbb{C}}\,\bullet\,(X^D[n \mapsto |\mathit{zero}|\,\mathsf{tt}]))\\ &\times (\mathit{suc}^D \quad : \mathsf{Tm}_{\mathbb{C}}\,(\bullet \triangleright n : |X|\,\mathsf{tt} \triangleright n^D : X^D[n \mapsto |\mathit{zero}|\,\mathsf{tt}])\,(X^D[n \mapsto (|\mathit{suc}|\,\mathsf{tt})[n \mapsto n]]))\end{aligned}$$

To explain $(|\mathsf{suc}|\,\mathsf{tt})[n \mapsto n]$): we have $\mathsf{suc}\,n$ in 2LTT, where n is an inner variable, and in the presheaf model inner variables become actual variables in the inner theory. Hence, we map the n which suc depends on to the concrete n in the context.

We can also interpret displayed algebras in finite product categories:

$$\begin{aligned}
&|\mathsf{DispAlg}\,\{\mathsf{NatSig}\}\,(X,\,\mathit{zero},\,\mathit{suc})|\,\{\bullet\}\,\mathsf{tt} \equiv \\
&\quad (X^D \quad : |\mathbb{C}|) \\
&\times (\mathit{zero}^D : \mathbb{C}(\bullet,\, X^D)) \\
&\times (\mathit{suc}^D \;\; : \mathbb{C}(\bullet \otimes |X|\,\mathsf{tt} \otimes X^D,\, X^D))
\end{aligned}$$

While displayed algebras in cwfs can be used as bundles of induction motives and methods, in finite product categories they are argument bundles to *primitive recursion*; this is sometimes also called a *paramorphism* [MFP91]. In an internal syntax, the type of primitive recursion for natural numbers could be written more compactly as:

$$\mathsf{primrec} : (X : \mathsf{Set}) \to X \to (\mathsf{Nat} \to X \to X) \to \mathsf{Nat} \to X$$

This is not the same thing as the usual recursion principle (corresponding to weak initiality), because of the extra dependency on Nat in the method for successors.

3.5.5 Sections

Sections are analogous to morphisms. We again have a choice between weak and strict preservation; below we have weak preservation.

$$\begin{aligned}
&{-^S} : (A : \mathsf{Ty})\{X\,X^D\}(X^S : (x : \mathsf{Tm}_0\,X) \to \mathsf{Tm}_0\,(X^D\,x)) \\
&\qquad \to (\alpha : A^A\,X) \to A^D\,X^D\,\alpha \to \mathsf{Set} \\
&\iota^S \qquad\quad X^S\,\alpha\;\alpha^D :\equiv \mathsf{Tm}_0\,(X^S\,\alpha = \alpha^D) \\
&(\iota \to A)^S\,X^S\,\alpha\;\alpha^D :\equiv (x : \mathsf{Tm}_0\,X) \to A^S\,X^S\,(\alpha\,x)\,(\alpha^D\,(X^S\,x))
\end{aligned}$$

3.5.6 Term Algebras

For term algebras, we need to assume a bit more in the inner theory. For starters, it has to support the theory of signatures. In order to avoid name clashes down

the line, we use SigTy_0 to refer to signature types, and SigTm_0 for terms. That is, we have

$$\begin{aligned}
\mathsf{SigTy}_0 &: \mathsf{Ty}_0 \\
\mathsf{Con}_0 &: \mathsf{Ty}_0 \\
\mathsf{Var}_0 &: \mathsf{Tm}_0\,\mathsf{Con}_0 \to \mathsf{Tm}_0\,\mathsf{SigTy}_0 \to \mathsf{Ty}_0 \\
\mathsf{SigTm}_0 &: \mathsf{Tm}_0\,\mathsf{Con}_0 \to \mathsf{Tm}_0\,\mathsf{SigTy}_0 \to \mathsf{Ty}_0 \\
\mathsf{Sub}_0 &: \mathsf{Tm}_0\,\mathsf{Con}_0 \to \mathsf{Tm}_0\,\mathsf{Con}_0 \to \mathsf{Ty}_0
\end{aligned}$$

together with all constructors and induction principles. We also assume inner Π-types, because we previously defined Sub using functions.

Remark. If we only want to construct term algebras, it is not necessary to assume inner induction principles. In this section, our goal is to redo the constructions of Chapter 2 without making essential changes, so we just assume everything that was available there.

We still have ToS in the outer layer. To make the naming scheme consistent, we shall write outer ToS types as SigTy_1, SigTm_1, Con_1, Var_1 and Sub_1. We have conversion functions from the outer ToS to the inner ToS:

Definition 38. We have the following **lowering** functions which preserve all structure.

$$\begin{aligned}
\downarrow &: \mathsf{SigTy}_1 \to \mathsf{Tm}_0\,\mathsf{SigTy}_0 \\
\downarrow &: \mathsf{Con}_1 \to \mathsf{Tm}_0\,\mathsf{Con}_0 \\
\downarrow &: \mathsf{Var}_1\,\Gamma\,A \to \mathsf{Tm}_0\,(\mathsf{Var}_0\,(\downarrow\Gamma)\,(\downarrow A)) \\
\downarrow &: \mathsf{SigTm}_1\,\Gamma\,A \to \mathsf{Tm}_0\,(\mathsf{SigTm}_0\,(\downarrow\Gamma)\,(\downarrow A)) \\
\downarrow &: \mathsf{Sub}_1\,\Gamma\,\Delta \to \mathsf{Tm}_0\,(\mathsf{Sub}_0\,(\downarrow\Gamma)\,(\downarrow\delta))
\end{aligned}$$

These are called "lifting" or "serialization" in the context of multi-stage programming; see e.g. the Lift typeclass in Haskell [PWK19]. There, like here, the point is to build object-language terms from meta-level ("compile-time") values.

Lowering is straightforward to define for types, contexts, variables and terms, but there is a bit of a complication for Sub. Unfolding the definitions, we need to map from $\{A\} \to \mathsf{Var}_1\,\Delta\,A \to \mathsf{SigTm}_1\,\Gamma\,A$ to $\mathsf{Tm}_0\,(\{A\} \to \mathsf{Var}_0\,(\downarrow\,\Delta)\,A \to \mathsf{SigTm}_0\,(\downarrow\,\Gamma)\,A)$. It might appear problematic that we have types and variables in *negative* position, because we cannot map inner types/variables to outer ones.

Fortunately, $\mathsf{Sub}_1\,\Gamma\,\Delta$ is isomorphic to a finite product type, and we can lower a finite product component-wise.

Concretely, we define lowering by induction on Δ, while making use of a case splitting operation for Var_0. We use an informal case operation below, which can be defined using inner induction. Note that since $\mathsf{Var}_0 \bullet A$ is an empty type, case splitting on it behaves like elimination for the empty type.

$$
\begin{aligned}
&\downarrow_\Delta : \mathsf{Sub}_1\,\Gamma\,\Delta \to \mathsf{Tm}_0\,(\mathsf{Sub}_0\,(\downarrow\Gamma)\,(\downarrow\Delta))\\
&\downarrow_\bullet \quad \sigma :\equiv \lambda\,\{A\}\,(x : \mathsf{Var}_0 \bullet A).\,\mathsf{case}\,x\,\mathsf{of}\,()\\
&\downarrow_{\Delta \triangleright B}\,\sigma :\equiv \lambda\,\{A\}\,(x : \mathsf{Var}_1\,(\downarrow\Delta \triangleright \downarrow B)\,A).\,\mathsf{case}\,x\,\mathsf{of}\\
&\qquad \mathsf{vz} \quad \to \downarrow(\sigma\,\mathsf{vz})\\
&\qquad \mathsf{vs}\,x \to \downarrow_\Delta\,(\sigma \circ \mathsf{vs})\,x
\end{aligned}
$$

In general, for finite A type, functions of the form $A \to \mathsf{Tm}_0\,B$ can be represented as inner types up to isomorphism; they can be viewed as finite products of terms.

Remark. For infinite A this does not work anymore in our system. In [ACKS19], the assumption that this still works with $A \equiv \mathsf{Nat}_1$ is an important axiom ("cofibrancy of Nat_1") which makes it possible to embed higher categorical structures in 2LTT. From the metaprogramming perspective, cofibrancy of Nat_1 implies that the inner theory is *infinitary*, since we can form inner terms from infinite collections of inner terms. We do not assume this axiom in 2LTT, although we will consider infinitary (object) type theories in Chapters 4 and 5.

We continue to the definition of term algebras. We fix an $\Omega : \mathsf{Con}_1$, and define $\mathsf{T} : \mathsf{Ty}_0$ as $\mathsf{SigTm}_0\,(\downarrow\Omega)\,\iota$.

$$
\begin{aligned}
&{-^T} : (A : \mathsf{SigTy}_1) \to \mathsf{Tm}_0\,(\mathsf{SigTm}_0\,(\downarrow\Omega)\,(\downarrow A)) \to A^A\,\mathsf{T}\\
&\iota^T \qquad\quad t :\equiv t\\
&(\iota \to A)^T\,t :\equiv \lambda\,u.\,A^T\,(\mathsf{app}\,t\,u)
\end{aligned}
$$

$$
\begin{aligned}
&{-^T} : (\Gamma : \mathsf{Con}_1) \to \mathsf{Sub}_1\,\Omega\,\Gamma \to \Gamma^A\,\mathsf{T}\\
&\Gamma^T\,\nu\,\{A\}\,x :\equiv A^T\,(\downarrow(\nu\,x))
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{TmAlg}_\Omega : \mathsf{Alg}\,\Omega\\
&\mathsf{TmAlg}_\Omega :\equiv \Omega^T\,\Omega\,\mathsf{id}
\end{aligned}
$$

We omit the $-^T$ interpretation for terms and substitutions for now, as they require a bit more setup, and they are not needed just for term algebras.

3.5.7 Recursor Construction

Recall from Section 2.3.1 that recursion is implemented using the $-^A$ interpretation of terms. Since terms are now in the inner theory, we need to define an inner version of the same interpretation. We need to compute types by inner induction, so we additionally assume a Russell-style inner U_0 universe. The Russell style means that we may freely coerce between $\mathsf{Tm}_0\,\mathsf{U}_0$ and Ty_0. The following are defined the same way as $-^A$ before.

$$
\begin{aligned}
&-^A : \mathsf{Tm}_0\,(\mathsf{SigTy}_0 \to \mathsf{U}_0 \to \mathsf{U}_0)\\
&-^A : \mathsf{Tm}_0\,(\mathsf{Con}_0 \to \mathsf{U}_0 \to \mathsf{U}_0)\\
&-^A : \mathsf{Tm}_0\,(\mathsf{SigTm}_0\,\Gamma\,A \to \{X : \mathsf{U}_0\} \to \Gamma^A\,X \to A^A\,X)\\
&-^A : \mathsf{Tm}_0\,(\mathsf{Sub}_0\,\Gamma\,\Delta \to \{X : \mathsf{U}_0\} \to \Gamma^A\,X \to \Delta^A\,X)
\end{aligned}
$$

Since lowering preserves all structure, and $-^A$ is defined in the same way in both the inner and outer theories, lowering is compatible with $-^A$ in the following way.

Lemma 4. Assume $A : \mathsf{SigTy}_1$, $\Gamma : \mathsf{Con}_1$, $X : \mathsf{Ty}_0$, $\gamma : \Gamma^A\,X$ and $t : \mathsf{SigTm}_1\,\Gamma\,A$. We have the following:

- $(A^A_{\to},\, A^A_{\leftarrow}) : \mathsf{Tm}_0\,((\downarrow A)^A\,X) \simeq A^A\,X$
- $(\Gamma^A_{\to},\, \Gamma^A_{\leftarrow}) : \mathsf{Tm}_0\,((\downarrow \Gamma)^A\,X) \simeq \Gamma^A\,X$
- $t^A\,\gamma \equiv A^A_{\to}\,((\downarrow t)^A\,(\Gamma^A_{\leftarrow}\,\gamma))$

Proof. By induction on Γ, A and t. □

We construct recursors now, yielding strict algebra morphisms. We assume $(X, \omega) : \mathsf{Alg}\,\Omega$. Recall that $\omega : \Omega^A\,X$, thus $\Omega^A_{\leftarrow}\,\omega : \mathsf{Tm}_0\,((\downarrow \Omega)^A\,X)$. We define $\mathsf{R} : \mathsf{Tm}_0\,\mathsf{T} \to \mathsf{Tm}_0\,X$ as $\mathsf{R}\,t :\equiv t^A\,(\Omega^A_{\leftarrow}\,\omega)$.

$$
\begin{aligned}
&-^R : (A : \mathsf{SigTy}_1)(t : \mathsf{Tm}_0\,(\mathsf{SigTm}_0\,(\downarrow\Omega)\,(\downarrow A))) \to A^M\,\mathsf{R}\,(A^T\,t)\,(\mathsf{A}^A_{\to}\,(t^A\,(\Omega^A_{\leftarrow}\,\omega)))\\
&\iota^R \qquad\quad t : t^A\,(\Omega^A_{\leftarrow}\,\omega) \equiv \iota^A_{\to}\,(t^A\,(\Omega^A_{\leftarrow}\,\omega))\\
&(\iota \to A)^R\,t :\equiv \lambda\,u.\,A^R\,(\mathsf{app}\,t\,u)
\end{aligned}
$$

$$
\begin{aligned}
&-^R : (\Gamma : \mathsf{Con}_1)(\nu : \mathsf{Sub}_1\,\Omega\,\Gamma) \to \Gamma^M\,\mathsf{R}\,(\Gamma^T\,\nu)\,(\nu^A\,\omega)\\
&\Gamma^R\,\nu\,\{A\}\,x :\equiv A^R\,(\downarrow(\nu\,x))
\end{aligned}
$$

In the proof obligation for $t^A\,(\Omega^A_{\leftarrow}\,\omega) \equiv \iota^A_{\rightarrow}\,(t^A\,(\Omega^A_{\leftarrow}\,\omega))$, $\iota^A_{\rightarrow}$ computes to the identity function; note that $\iota^A_{\rightarrow} : \mathsf{Tm}_0\,X \to \mathsf{Tm}_0\,X$. Hence the equality becomes reflexive.

In $\Gamma^R\,\nu\,\{A\}\,x :\equiv A^R\,(\downarrow(\nu\,x))$, we have that

$$A^R\ \downarrow(\nu\,x) : A^M\,\mathsf{R}\,(A^T\,(\downarrow(\nu\,x)))\,(A^A_{\rightarrow}\,(\downarrow(\nu\,x)^A\,(\Omega^A_{\leftarrow}\,\omega)))$$

Hence by Lemma 4, we have

$$A^R\ \downarrow(\nu\,x) : A^M\,\mathsf{R}\,(A^T\,(\downarrow(\nu\,x)))\,((\nu\,x)^A\,\omega)$$

Hence, by the definition of $-^A$ for substitutions:

$$A^R\ \downarrow(\nu\,x) : A^M\,\mathsf{R}\,(A^T\,(\downarrow(\nu\,x)))\,(\nu^A\,\omega\,x)$$

Which is exactly what is required when we unfold the expected return type:

$$
\begin{aligned}
&-^R : (\Gamma : \mathsf{Con}_1)(\nu : \mathsf{Sub}_1\,\Omega\,\Gamma) \to \Gamma^M\,\mathsf{R}\,(\Gamma^T\,\nu)\,(\nu^A\,\omega)\\
&-^R : (\Gamma : \mathsf{Con}_1)(\nu : \mathsf{Sub}_1\,\Omega\,\Gamma) \to \{A\}(x : \mathsf{Var}_1\,\Gamma\,A) \to A^M\,\mathsf{R}\,(A^T\,(\downarrow(\nu\,x)))\,(\nu^A\,\omega\,x)
\end{aligned}
$$

The recursor is defined the same way as in Definition 10:

$$
\begin{aligned}
&\mathsf{Rec}_\Omega : (alg : \mathsf{Alg}\,\Omega) \to \mathsf{Mor}\,\mathsf{TmAlg}_\Omega\,alg\\
&\mathsf{Rec}_\Omega\,(X,\,\omega) :\equiv (\mathsf{R},\,\Omega^R\,\Omega\,\mathsf{id})
\end{aligned}
$$

3.5.8 Eliminator Construction

For induction, we need to additionally define $-^D$ in the inner layer.

$$
\begin{aligned}
&-^D : \mathsf{Tm}_0\,((A : \mathsf{SigTy}_0)\{X\} \to (\mathsf{Tm}_0\,X \to \mathsf{U}_0) \to A^A\,X \to \mathsf{U}_0)\\
&-^D : \mathsf{Tm}_0\,((\Gamma : \mathsf{Con}_0)\{X\} \to (\mathsf{Tm}_0\,X \to \mathsf{U}_0) \to \Gamma^A\,X \to \mathsf{U}_0)\\
&-^D : \mathsf{Tm}_0\,((t : \mathsf{SigTm}_0\,\Gamma\,A) \to \Gamma^D\,X^D\,\gamma \to A^D\,X^D\,(t^A\,\gamma))\\
&-^D : \mathsf{Tm}_0\,((\sigma : \mathsf{Sub}_0\,\Gamma\,\Delta) \to \Gamma^D\,X^D\,\gamma \to \Delta^D\,X^D\,(\sigma^A\,\gamma))
\end{aligned}
$$

Lemma 5. We have again compatibility of lowering with $-^D$. Assuming $(X,\,\gamma) : \mathsf{Alg}\,\Gamma$, $(X^D,\,\gamma^D) : \mathsf{DispAlg}\,(X,\,\gamma)$, $t : \mathsf{SigTm}_1\,\Gamma\,A$, and $\alpha : A^A\,X$, we have

- $(\mathsf{A}^D_{\rightarrow},\,\mathsf{A}^D_{\leftarrow}) : \mathsf{Tm}_0\,((\downarrow\mathsf{A})^D\,X^D\,(\mathsf{A}^A_{\leftarrow}\,\alpha)) \simeq \mathsf{A}^D\,X^D\,\alpha$

- $(\Gamma^D_{\rightarrow}, \Gamma^D_{\leftarrow}) : \mathsf{Tm}_0\,((\downarrow\Gamma)^D\,X^D\,(\Gamma^A_{\leftarrow}\,\gamma)) \simeq \Gamma^D\,X^D\,\gamma$
- $t^D\,\gamma^D \equiv A^D_{\rightarrow}\,((\downarrow t)^D\,(\Gamma^D_{\leftarrow}\,\gamma^D))$

The equation for $t^D\,\gamma^D$ is well-typed because of the term equation in Lemma 4.

Proof. Again by induction on Γ, A and t. □

We also need to extend $-^T$ with action on terms. Note that we return an inner equality, since we can only compute such equality by induction on the inner term input:

$$-^T : (t : \mathsf{SigTm}_0\,(\downarrow\Gamma)\,(\downarrow A))(\nu : \mathsf{Sub}_1\,\Omega\,\Gamma) \rightarrow \mathsf{Tm}_0\,(A^A_{\leftarrow}\,(A^T\,(t[\downarrow\nu])) = t^A\,(\Gamma^A_{\leftarrow}\,\nu))$$

We assume $(X^D, \omega^D) : \mathsf{DispAlg}\,\mathsf{TmAlg}_\Omega$, and define elimination as follows:

$$\begin{aligned}&\mathsf{E} : (t : \mathsf{Tm}_0\,\mathsf{T}) \rightarrow \mathsf{Tm}_0\,(X^D\,t)\\&\mathsf{E}\,t :\equiv t^D\,(\Omega^D_{\leftarrow}\,\omega^D)\end{aligned}$$

This definition is well-typed only up to $t^T\,\mathsf{id} : \mathsf{Tm}_0\,(t = t^A\,(\Omega^A_{\leftarrow}\,(\Omega^T\,\Omega\,\mathsf{id})))$. Since $t^T\,\mathsf{id}$ is an inner equality, in a fully formal intensional presentation we would have to write an explicit transport in the definition.

We shall skip the remainder of the eliminator construction; it goes the same way as in Definition 11. Intuitively, this is possible since the inner theory has all necessary features to reproduce the eliminator construction, and lowering preserves all structure.

Since t^T yields inner equations, this implies that the displayed algebra sections returned by the eliminator are *weak sections*, i.e. they contain β-rules expressed in inner equalities.

3.6 Discussion

3.6.1 Evaluation

Let us review how 2LTT addresses the shortcomings that we mentioned in Section 3.1.

Generalized semantics. For this purpose, there is minimal technical overhead to using 2LTT; we only need to sprinkle Ty_0 and Tm_0 around a bit, and we get

semantics internally to cwfs (including all finite product categories).

Weak vs. strict preservation in term algebras. Here, we had to deal with significantly more noise, because of the necessary lowerings and their preservation isomorphisms. However, this overhead should be compared to reasoning about object-level definitional equality in a deeply embedded way, which entails explicitly manipulating abstract syntax and its substitutions and weakenings. Compared to that, 2LTT is tremendously easier. One could also argue that lowering is an entirely mechanical affair, and we can just omit most of it if we are comfortable enough with the formalism.

3.6.2 Recursor vs. Eliminator Construction

The essential extra detail that we had to handle in Section 3.5 was the choice between strict and weak equations. This choice brings along further implementation constraints.

Strict equations are stronger as assumptions, because they represent definitional equality of inner terms. However, we can only produce strict equations by eliminating from outer types. Hence, if we aim to output strict equations, we have to assume every dependency in the outer theory, which in turn may require using lowering.

Weak equations are easier to produce: strict equality implies weak equality, plus we can prove weak equality by inner induction. But we cannot use weak equality to transport outer values.

In the recursor construction, we produced strict β-rules, which trivially imply weak β-rules as well. In contrast, the eliminator construction relies essentially on equations produced by $-^T$ for inner terms. Can we somehow get strict equations from $-^T$? This does not seem possible, at least without changing the approach in a major way: terms must be inner, or else we have no hope of inner recursion/induction. And if terms are inner, then E must act on arbitrary inner terms, hence $-^T$ must do so too.

It would be interesting to check if there is a possible alternative formulation of term algebras and constructed eliminators, which makes it possible to get strict eliminators. Looking back to Section 2.4.2, it appears that by using spine neutral terms, we get strict elimination for each signature. However, this relies on the Agda implementation of pattern matching and structurally recursive definitions,

so it would require more work to translate these definitions to 2LTT in a generic way.

CHAPTER 4

Finitary Quotient Inductive-Inductive Signatures

In this chapter we bump the expressive power of signatures by a large margin, and also substantially extend the semantics. However, we keep the basic approach the same; indeed its advantages become apparent with the more sophisticated signatures.

We use two different setups for semantics in this chapter.

- In Sections 4.1-4.3 we work in 2LTT, thereby getting a generalized semantics for signatures. Here we keep details about universe levels to the minimum.
- In Section 4.5, we work in an extensional type theory with cumulative universes. This is more suited for the term algebra construction, where (as we will see) 2LTT does not bring any advantage, but we do need to be more precise about universes.

4.1 Theory of Signatures

Signatures are once again given by contexts of a type theory, but now it is a dependent type theory, given as a cwf with certain type formers, in the style of Section 3.2.

Metatheory and terminology

We work in 2LTT with Ty_0 and Tm_0, and make the following assumptions:

- Ty_0 is closed under $\top$, Σ and extensional identity $-=-$. The inner identity reflects the outer one.

- The outer identity $- \equiv -$ is also extensional; it supports UIP, function extensionality, and it reflects strict equality in some unspeficied metatheory outside 2LTT. This reflection is used to justify omitting transports along $- \equiv -$ in our notation.

In the following we specify models of the theory of *finitary quotient inductive-inductive signatures*. The names involved are a bit of a mouthful, so we abbreviate "finitary quotient inductive-inductive" as FQII, and like before, we abbreviate "theory of signatures" as ToS. In this chapter, by signature we mean an FQII signature unless otherwise specified.

Additionally, we abbreviate "quotient inductive-inductive types" as QIIT, and we may qualify it to FQIIT if it is finitary. A *type* in this sense is simply the initial algebra for a given FQII signature. We shall use this naming in the rest of the thesis; an *inductive type* is an initial algebra for a signature. Also, we use *syntax* as a synonym for initial algebra.

Definition 39. A **model of the theory of signatures** consists of the following.

- A **cwf** with underlying sets Con, Sub, Ty and Tm, all returning in the outer Set universe of 2LTT.
- A **Tarski-style universe** U with decoding El.
- An **extensional identity type** $\mathsf{Id} : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma$, specified by $(\mathsf{reflect}, \mathsf{refl}) : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,t\,u) \simeq (t \equiv u)$.
- An **inductive function type** $\Pi : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Ty}\,(\Gamma \triangleright \mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma$, specified by $(\mathsf{app}, \mathsf{lam}) : \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) \simeq \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B$.
- An **external function type** $\Pi^{\mathsf{ext}} : (Ix : \mathsf{Ty}_0) \to (Ix \to \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,\Gamma$, specified by $(\mathsf{app}^{\mathsf{ext}}, \mathsf{lam}^{\mathsf{ext}}) : \mathsf{Tm}\,\Gamma\,(\Pi^{\mathsf{ext}}\,Ix\,B) \simeq ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(B\,i))$.

At this point we only have a notion of model for ToS, but as we will see in Chapter 5, ToS is also an algebraic theory, more specifically an infinitary QII one. It is infinitary because Π^{ext} and $\mathsf{lam}^{\mathsf{ext}}$ allow branching which is indexed over elements of arbitrary $Ix : \mathsf{Ty}_0$ types.

Because of the algebraic character of ToS, there is a category of ToS models where morphisms strictly preserve all structure, and the initial model corresponds to the syntax. We will make this precise in Chapter 5. We also assume that the ToS syntax exists.

Definition 40. An FQII **signature** is an element of Con in the syntax of ToS.

We review several example signatures in the following, using progressively more ToS type formers. We also introduce progressively more compact notation for signatures. As a rule of thumb, we shall use compact notation for larger and more complex signatures, but we shall be more explicit when we specify models of ToS later in this chapter.

Example 10. Simple inductive signatures can be evidently expressed using U and Π. By adding a single U to the signature, we introduce the inductive sort.

$$\mathsf{NatSig} :\equiv \bullet \triangleright (N : \mathsf{U}) \triangleright (\mathit{zero} : \mathsf{El}\, N) \triangleright (\mathit{suc} : \Pi(n : N)(\mathsf{El}\, N))$$
$$\mathsf{TreeSig} :\equiv \bullet \triangleright (T : \mathsf{U}) \triangleright (\mathit{leaf} : \mathsf{El}\, T) \triangleright (\mathit{node} : \Pi(t_1 : T)(\Pi(t_2 : T)(\mathsf{El}\, T)))$$

Observe that the domains in Π are terms with type U, while the codomains are proper types.

Notation 12. We write non-dependent function types in ToS as follows.

- $a \Rightarrow B$ for $\Pi\,(\_ : a)\,B$.
- $Ix \Rightarrow^{\mathsf{ext}} B$ for $\Pi^{\mathsf{ext}}\,Ix\,(\lambda\,\_.\,B)$.

Using this notation, we may write $\mathit{suc} : N \Rightarrow \mathsf{El}\, N$ and $\mathit{node} : T \Rightarrow T \Rightarrow \mathsf{El}\, T$.

Notation 13. The "categorical" application app with explicit substitutions is a bit inconvenient. Instead, we simply write whitespace for Π and Π^{ext} application:

$$t\,u :\equiv (\mathsf{app}\, t)[\mathsf{id}, u]$$
$$t\,u :\equiv (\mathsf{app}^{\mathsf{ext}}\, t)\,u$$

Example 11. We may have any number of sorts by adding more U to the signatures. Moreover, sorts can be indexed over previous sorts. Hence, using only U, El and Π, we can express any closed inductive-inductive type [NF13]. The following fragment of the the signature for categories is such:

$$\bullet \triangleright (\mathit{Obj} : \mathsf{U}) \triangleright (\mathit{Hom} : \mathit{Obj} \Rightarrow \mathit{Obj} \Rightarrow \mathsf{U}) \triangleright (\mathit{id} : \Pi(i : \mathit{Obj})\,(\mathsf{El}\,(\mathit{Hom}\, i\, i)))n$$

These inductive-inductive signatures are more flexible than those in prior literature [NF13], since we allow type constructors (sorts) and point constructors to

be arbitrarily mixed, as opposed to mandating that sorts are declared first. For example:

$$\bullet \triangleright (A : \mathsf{U}) \triangleright (a : \mathsf{El}\, A) \triangleright (B : A \Rightarrow \mathsf{U}) \triangleright (C : B\, a \Rightarrow \mathsf{U})$$

Here C is indexed over $B\, a$, where a is a point constructor of a, so a sort specification mentions a point constructor.

Example 12. Id lets us add equations to signatures. With this, we can write down the full signature for categories:

$$\begin{aligned}
&\bullet \triangleright (Obj \quad : \mathsf{U}) \\
&\quad \triangleright (Hom \ : Obj \Rightarrow Obj \Rightarrow \mathsf{U}) \\
&\quad \triangleright (id \quad\ : \Pi(i : Obj)\, (\mathsf{El}\, (Hom\, i\, i))) \\
&\quad \triangleright (comp : \Pi\, (i\, j\, k : Obj)\, (Hom\, j\, k \Rightarrow Hom\, i\, j \Rightarrow \mathsf{El}\, (Hom\, i\, k))) \\
&\quad \triangleright (idr \quad\ : \Pi\, (i\, j : Obj)(f : Hom\, i\, j)\, (\mathsf{Id}\, (comp\, i\, i\, j\, f\, (id\, i))\, f)) \\
&\quad \triangleright (idl \quad\ : \Pi\, (i\, j : Obj)(f : Hom\, i\, j)\, (\mathsf{Id}\, (comp\, i\, j\, j\, (id\, j)\, f)\, f)) \\
&\quad \triangleright (assoc : \Pi\, (i\, j\, k\, l : Obj)(f : Hom\, j\, l)(g : Hom\, j\, k)(h : Hom\, i\, j) \\
&\qquad\qquad (\mathsf{Id}\, (comp\, i\, j\, l\, (comp\, j\, k\, l\, f\, g)\, h)\, (comp\, i\, k\, l\, f\, (comp\, i\, j\, k\, g\, h))
\end{aligned}$$

Now, this is already rather hard to read, even together with a compressed notation for multiple Π binders.

Notation 14. For more complex signatures, we may entirely switch to an internal notation, where we mostly reuse the conventions in the metatheories. We use $(x : a) \to B$ for inductive functions, $(x : A) \to^{\mathsf{ext}} B$ for external functions, but we still write Id for the identity type and make U and El explicit. In this notation, a signature is just a listing of binders. The category signature becomes the following:

$$\begin{aligned}
&\mathsf{Obj} \quad : \mathsf{U} \\
&\mathsf{Hom} \ : \mathsf{Obj} \to \mathsf{Obj} \to \mathsf{U} \\
&\mathsf{id} \quad\ : \mathsf{El}\, (\mathsf{Hom}\, i\, i) \\
&- \circ - \ : \mathsf{Hom}\, j\, k \to \mathsf{Hom}\, i\, j \to \mathsf{El}\, (\mathsf{Hom}\, i\, k) \\
&\mathsf{idr} \quad\ : \mathsf{Id}\, (f \circ \mathsf{id})\, f \\
&\mathsf{idl} \quad\ : \mathsf{Id}\, (\mathsf{id} \circ f)\, f \\
&\mathsf{assoc} : \mathsf{Id}\, (f \circ (g \circ h))\, ((f \circ g) \circ h)
\end{aligned}$$

Example 13. The external function type makes it possible to reference inner types (in 2LTT) in signatures. Here "external" is meant relative to a given signature, and refers to types and inhabitants which are not introduced inside a signature. For example, we give a signature for lists by assuming $A : \mathsf{Ty}_0$ for the (external) type of list elements:

$$\begin{aligned}
&\mathsf{List} &&: \mathsf{U}\\
&\mathsf{nil} &&: \mathsf{El}\,\mathsf{List}\\
&\mathsf{cons} &&: A \to^{\mathsf{ext}} \mathsf{List} \to \mathsf{El}\,\mathsf{List}
\end{aligned}$$

Hence, "parameters" are always assumptions made in the metatheory. We can also *index* sorts by external values. Let us specify length-indexed vectors now; we keep the $A : \mathsf{Ty}_0$ assumption, but also assume that Ty_0 has natural numbers, with $\mathsf{Nat}_0 : \mathsf{Ty}_0$, zero_0 and suc_0.

$$\begin{aligned}
&\mathsf{Vec} &&: \mathsf{Nat}_0 \to^{\mathsf{ext}} \mathsf{U}\\
&\mathsf{nil} &&: \mathsf{El}\,(\mathsf{Vec}\,\mathsf{zero}_0)\\
&\mathsf{cons} &&: (n : \mathsf{Nat}_0) \to^{\mathsf{ext}} A \to^{\mathsf{ext}} \mathsf{Vec}\,n \to \mathsf{El}\,(\mathsf{Vec}\,(\mathsf{suc}_0\,n))
\end{aligned}$$

Example 14. We can also introduce *sort equations* using Id: this means equating terms of U, i.e. inductively specified sets. This is useful for specifying certain strict type formers. For example, a signature for cwfs can be extended with a specification for strict constant families.

$$\begin{aligned}
&\mathsf{Con} &&: \mathsf{U}\\
&\mathsf{Sub} &&: \mathsf{Con} \to \mathsf{Con} \to \mathsf{U}\\
&\mathsf{Ty} &&: \mathsf{Con} \to \mathsf{U}\\
&\mathsf{Tm} &&: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{U}\\
&... &&\\
&\mathsf{K} &&: \mathsf{Con} \to \{\Gamma : \mathsf{Con}\} \to \mathsf{El}\,(\mathsf{Ty}\,\Gamma)\\
&\mathsf{K}_{\mathsf{spec}} &&: \mathsf{Id}\,(\mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta))\,(\mathsf{Sub}\,\Gamma\,\Delta)
\end{aligned}$$

The equation for Russell-style universes is likewise a sort equation:

$$\begin{aligned}
&\mathsf{Univ} &&: \mathsf{El}\,(\mathsf{Ty}\,\Gamma)\\
&\mathsf{Russell} &&: \mathsf{Id}\,(\mathsf{Tm}\,\Gamma\,\mathsf{Univ})\,(\mathsf{Ty}\,\Gamma)
\end{aligned}$$

Example 15. As we mentioned in Definition 25, there is a signature for presheaves, so let us look at that now. Assume a category $\mathbb{C}$ in the inner theory; this means that objects and morphisms of $\mathbb{C}$ are in Ty_0.

$$\begin{aligned}
\mathsf{Obj} &: |\mathbb{C}| \to^{\mathsf{ext}} \mathsf{U} \\
\mathsf{Hom} &: \mathbb{C}(i,\, j) \to^{\mathsf{ext}} \mathsf{Obj}\, j \to \mathsf{El}\,(\mathsf{Obj}\, i) \\
\mathsf{Hom}_{\mathsf{id}} &: \mathsf{Id}\,(\mathsf{Hom}\,\mathsf{id}\, x)\, x \\
\mathsf{Hom}_{\circ} &: \mathsf{Id}\,(\mathsf{Hom}\,(f \circ g)\, x)\,(\mathsf{Hom}\, f\,(\mathsf{Hom}\, g\, x))
\end{aligned}$$

We depart from the sugary naming scheme in Definition 25, and name the action on objects Obj and the action on morphisms Hom. When we give semantics to this signature in Section 4.2, we will get as algebras functors from $\mathbb{C}^{\mathsf{op}}$ to the category of inner types. That category has elements of Ty_0 as objects and $\mathsf{Tm}_0\, A \to \mathsf{Tm}_0\, B$ functions as morphisms.

Strict positivity

Only strictly positive signatures are expressible. Similarly to the case with simple signatures, there is no way to abstract over a inductive functions, since inductive function domains are in U, and U has no type formers at all. With Π^{ext}, we can abstract over functions, but only those which are external to a signature and do not depend on internally specified constructors.

Limitation: nested induction

Nested induction means that external type functions may be applied to expressions internal to the theory of signatures. This is not possible in any of the signatures in this thesis. A common example is rose trees, assuming external $\mathsf{List} : \mathsf{Set} \to \mathsf{Set}$:

$$\begin{aligned}
&\mathsf{Tree} : \mathsf{Set} \\
&\mathsf{node} : \mathsf{List}\,\mathsf{Tree} \to \mathsf{Tree}
\end{aligned}$$

The $\mathsf{List}\,\mathsf{Tree}$ expression is not representable in a signature; the List function is external, while Tree would be an internal sort. This style of inductive definition requires reasoning about the polarity of all external type functions: only the strictly positive $\mathsf{Set} \to \mathsf{Set}$ functions should be allowed. With general type functions we would also need to track polarity of multiple parameters, or even higher-order polarity.

Many use cases of nested induction can be removed by "including" the external type constructor into the signature. In the case of rose trees, this means defining lists and trees mutually:

$$\begin{aligned}
&\mathsf{List} &&: \mathsf{U}\\
&\mathsf{Tree} &&: \mathsf{U}\\
&\mathsf{nil} &&: \mathsf{El}\,\mathsf{List}\\
&\mathsf{cons} &&: \mathsf{Tree} \to \mathsf{List} \to \mathsf{El}\,\mathsf{List}\\
&\mathsf{node} &&: \mathsf{List}\,\mathsf{Tree} \to \mathsf{Tree}
\end{aligned}$$

Of course, nested induction would be still desirable because of the code reuse that it enables.

4.2 Semantics

4.2.1 Overview

For simple signatures, we only gave semantics in enough detail so that notions of recursion and induction could be recovered. We aim to do more now. For each signature, we would like to have

1. A *category* of algebras, with homomorphisms as morphisms.
2. A notion of induction, which requires a notion of dependent algebras.
3. A proof that for algebras, initiality is equivalent to supporting induction.

We do this by creating a model of ToS where contexts (signatures) are categories with certain extra structure and substitutions are structure-preserving functors. Then, ToS signatures can be interpreted in this model, using the initiality of ToS syntax (i.e. the recursor).

Our semantics has a type-theoretic flavor, which is inspired by the cubical set model of Martin-Löf type theory by Bezem et al. [BCH14]. The idea is to avoid strictness issues by starting from basic ingredients which are already strict enough. Hence, instead of modeling ToS types as certain slices and substitution by pullback, we model types as displayed categories with extra structure, which naturally support strict reindexing/substitution.

We make a similar choice in the interpretation of signatures themselves: we use structured cwfs of algebras, where types correspond to displayed algebras. This choice is in contrast to having finitely complete categories of algebras. Preliminarily, the reason is that "native" displayed algebras and sections allow us to compute induction principles strictly as one would write in a type theory. In fact, in this chapter we recover exactly the same semantics for simple signatures that we already specified.

In contrast, displayed algebras are a derived notion in finitely complete categories, and the induction principles would be only up to isomorphism. This issue is perhaps not relevant from a purely categorical perspective, but we are concerned with eventually implementing QIITs in proof assistants. If we do not compute induction principles here in an exact way, we do not get them from anywhere else.

4.2.2 Separate vs. Bundled Models

Previously, we defined $-^A$, $-^M$, $-^D$ and $-^S$ interpretations of signatures separately, by doing induction anew for each one. Formally, this amounts to giving a plain model of ToS in order to define $-^A$, but then giving three *displayed* models of ToS to specify the other interpretations, because they sometimes need to refer to the recursors or eliminators of other interpretations.

For example, $-^A : \mathsf{Con} \to \mathsf{Set}$ while $-^D : (\Gamma : \mathsf{Con}) \to \Gamma^A \to \mathsf{Set}$, so displayed algebras already refer to $-^A$, which is part of the recursor for the corresponding model.

However, this piecewise style can be avoided: we can give a single non-displayed model which packs everything in a Σ-type, yielding just one interpretation function for signatures. Let us call that function $-^M$ now:

$$\begin{aligned} -^M : \mathsf{Con} \to {} & (A \ : \mathsf{Set}) \\ & \times \ (M : A \to A \to \mathsf{Set}) \\ & \times \ (D \ : A \to \mathsf{Set}) \\ & \times \ (S \ : (a : A) \to D\,a \to \mathsf{Set}) \end{aligned}$$

Note that it is often not possible to merge multiple recursors/eliminators by packing models together. For example, addition on natural numbers is defined by recursion, and so is multiplication; but since multiplication calls addition in an iterated fashion, it is not possible to define both operations by a single algebra.

Nevertheless, merging does work in our case. We will, in fact, get a formal vocabulary for merging models (and manipulating them in other ways) from the semantics of ToS itself.

In simple cases, and in Agda, the piecewise style is convenient, since we do not have to deal with Σ-s. However, for larger models, important organizing principles may become more apparent if we bundle things together.

In the following, we shall define a model **M** : ToS such that its Con component is a bundle containing all A, M, D, S components, plus a number of additional components. We present the components of **M** in the same order as in Definition 39. There is significant overlap in names and notations, so we use **bold** font to disambiguate components of **M** from components of other structures. For example, we use $\boldsymbol{\sigma}$ **: Sub** $\boldsymbol{\Gamma}$ $\boldsymbol{\Delta}$ to denote a substitution in **M**, while there could be a Sub-named components in other structures under consideration.

4.2.3 Finite Limit Cwfs

We define **Con** : Set as the type of finite limit cwfs. Recall that this specifies the objects of the underlying cwf of **M**. In the following we specify flcwfs and describe some internal constructions.

Definition 41. We define flcwf : Set as an iterated Σ-type with the following components:

1. A cwf with Con, Sub, Ty, Tm all returning in Set. *Remark:* this implies that flcwf : Set is in a larger universe than all of these internal components. We continue to elide universe sizing details.

2. Σ-types.

3. Extensional identity type Id with refl and reflect.

4. Strict constant families K.

Definition 42. We abbreviate the additional structure on cwfs consisting of Σ, Id and K as **fl-structure**.

We recover previous concepts as follows. Assuming Γ signature, we get an flcwf by interpreting Γ in **M**. In that flcwf we have

- Con as the type of algebras.

- Sub as the type of algebra morphisms.
- Ty as the type of displayed algebras.
- Tm as the type of displayed algebra sections.

From this, notions of initiality and induction are apparent as well. Initiality is the usual categorical notion.

Definition 43. Assuming $\mathbf{\Gamma} : \mathbf{Con}$, we define the **induction predicate** on objects:

$$\begin{aligned}&\mathsf{Inductive} : \mathsf{Con}_{\mathbf{\Gamma}} \to \mathsf{Set}\\ &\mathsf{Inductive}\,\Gamma :\equiv (A : \mathsf{Ty}_{\mathbf{\Gamma}}\,\Gamma) \to \mathsf{Tm}_{\mathbf{\Gamma}}\,\Gamma\,A\end{aligned}$$

In short, an algebra is inductive if every displayed algebra over it has a section. We also know that induction and initiality are equivalent.

Theorem 1. *An object $\Gamma : \mathsf{Con}_{\mathbf{\Gamma}}$ in an flcwf $\mathbf{\Gamma}$ supports induction if and only if it is initial. Moreover, induction and initiality are both mere properties.*

Proof. First, we show that induction implies initiality. We assume $\Gamma : \mathsf{Con}$, $\mathsf{ind} : \mathsf{Inductive}\,\Gamma$ and $\Delta : \mathsf{Con}$. We aim to show that there is a unique inhabitant of $\mathsf{Sub}\,\Gamma\,\Delta$. We have $\mathsf{ind}\,(\mathsf{K}\,\Delta) : \mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta)$, hence $\mathsf{ind}\,(\mathsf{K}\,\Delta) : \mathsf{Sub}\,\Gamma\,\Delta$. We only need to show that this is unique. Assume $\delta : \mathsf{Sub}\,\Gamma\,\Delta$. Now, $\mathsf{ind}\,(\mathsf{Id}\,\delta\,(\mathsf{ind}\,(\mathsf{K}\,\Delta))) : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,\delta\,(\mathsf{ind}\,(\mathsf{K}\,\Delta)))$, and it follows by equality reflection that $\delta \equiv \mathsf{ind}\,(\mathsf{K}\,\Delta)$.

Second, the other direction. We assume that Γ is initial, and also $A : \mathsf{Ty}\,\Gamma$, and aim to inhabit $\mathsf{Tm}\,\Gamma\,A$. By initiality we get a unique $\sigma : \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A)$. Now, $\mathsf{q}[\sigma] : \mathsf{Tm}\,\Gamma\,(A[\mathsf{p} \circ \sigma])$, but since $\mathsf{p} \circ \sigma : \mathsf{Sub}\,\Gamma\,\Gamma$, it must be equal to id by the initiality of Γ. Hence, $\mathsf{q}[\sigma] : \mathsf{Tm}\,\Gamma\,A$.

Lastly: it is well-known that initiality is a mere property, so let us show the same for induction. We assume $\mathsf{ind}, \mathsf{ind}' : \mathsf{Inductive}\,\Gamma$ and $A : \mathsf{Ty}\,\Gamma$. We have $\mathsf{reflect}\,(\mathsf{ind}\,(\mathsf{Id}\,(\mathsf{ind}\,A)\,(\mathsf{ind}'\,A))) : \mathsf{ind}\,A \equiv \mathsf{ind}'\,A$. Since A is arbitrary, by function extensionality we also have $\mathsf{ind} \equiv \mathsf{ind}'$. □

Theorem 2. *$\mathsf{Tm}\,\Gamma\,A$ in an flcwf is propositional when Γ is initial.*

Proof. Assuming $t, u : \mathsf{Tm}\,\Gamma\,A$, we have $\mathsf{reflect}\,(\mathsf{ind}\,(\mathsf{Id}\,t\,u)) : t \equiv u$. □

Note that the above proofs do not rely on Σ-types in the flcwf, so why do we include them in the semantics? One reason is the prior result by Clairmabault and Dybjer [CD14], that a slightly different formulation of flcwfs is biequivalent to finitely complete categories. More concretely, in ibid. there is a 2-category of cwfs with Σ, Id and "democracy", the last of which is equivalent to the weak formulation of constant families. Then, it is shown that this 2-category is biequivalent to the 2-category of finitely complete categories. Thus, including Σ is a good deal, as this allows us to connect our semantics back to finitely complete categories, which are more commonplace in categorical settings.

We recover finite limits in an flcwf as follows. The product of Γ and Δ is given by $\Gamma \triangleright \mathsf{K}\,\Delta$, and we get projection and pairing from context comprehension. The equalizer of σ, $\delta : \mathsf{Sub}\,\Gamma\,\Delta$ is given by $\Gamma \triangleright \mathsf{Id}\,\sigma\,\delta$, which is well-typed because morphisms can be viewed as terms, e.g. $\sigma : \mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta)$. The unique morphism out of the equalizer is $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright \mathsf{Id}\,\sigma\,\delta)\,\Gamma$.

Our flcwf is not exactly the same as in [CD14] because our constant families are strict. But we certainly do not lose anything by having stricter semantics, since the weak version can be trivially recovered.

In the following we present some concepts and results in flcwfs.

Definition 44 (Type categories, c.f. [CD14, Section 2.2])**.** For each $\Gamma : \mathsf{Con}$, there is a category whose objects are types $A : \mathsf{Ty}\,\Gamma$, and morphisms from A to B are terms $t : \mathsf{Tm}\,(\Gamma \triangleright A)\,(B[\mathsf{p}])$. Identity morphisms are given by $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])$, and composition $t \circ u$ by $t[\mathsf{p}, u]$. The assignment of type categories to contexts extends to a split indexed category. For each $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$, there is a functor from $\mathsf{Ty}\,\Delta$ to $\mathsf{Ty}\,\Gamma$, which sends A to $A[\sigma]$ and $t : \mathsf{Tm}\,(\Gamma \triangleright A)\,(B[\mathsf{p}])$ to $t[\sigma \circ \mathsf{p}, \mathsf{q}]$.

Notation 15.

- In any cwf, we use $\sigma : \Gamma \simeq \Delta$ to indicate that $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$ is an isomorphism with inverse σ^{-1}.
- A *type isomorphism*, written as $t : A \simeq B$ is an isomorphism in a type category, with inverse as t^{-1}.

Theorem 3 (Equivalence of types and slices, c.f. [CD14, Section 2.2])**.** *Assume that we work in an flcwf* $\mathbf{\Gamma}$*. For each* $\Gamma : \mathsf{Con}$*, the type category* $\mathsf{Ty}\,\Gamma$ *is equivalent to the slice category* $\mathbf{\Gamma}/\Gamma$*.*

Remark. In the flcwf of sets where types are $A \to \mathsf{Set}$ families, the above theorem yields the equivalence of $A \to \mathsf{Set}$ and $(B : \mathsf{Set}) \times (B \to A)$. This is sometimes called the "family-fibration" equivalence. It is also a notable motivating example for univalence in type theory: it is not an isomorphism of sets, but only an equivalence up to isomorphism of sets. So this is an example for an equivalence which quite naturally arises even if we only care about sets, but one which is not covered by set-level univalence, and actually requires univalence for groupoids, if we want to prove it as a propositional equality.

4.2.4 The Cwf of Finite Limit Cwfs

The next task is to define the cwf part of $\mathbf{M}$. We already know that objects are flcwfs.

Category

A **morphism** $\boldsymbol{\sigma} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\boldsymbol{\Delta}$ is an algebra homomorphism, viewing flcwfs as algebraic structures. Hence, $\boldsymbol{\sigma}$ includes a functor between underlying categories, but it also maps types to types and terms to terms, and strictly preserves all structure.

Notation 16. We may implicitly project out the underlying maps from $\boldsymbol{\sigma}$. Hence, we have the following four maps:

$$\begin{aligned}
&\boldsymbol{\sigma} : \mathsf{Con}_{\boldsymbol{\Gamma}} \to \mathsf{Con}_{\boldsymbol{\Delta}} \\
&\boldsymbol{\sigma} : \mathsf{Sub}_{\boldsymbol{\Gamma}}\,\Gamma\,\Delta \to \mathsf{Sub}_{\boldsymbol{\Delta}}\,(\boldsymbol{\sigma}\,\Gamma)\,(\boldsymbol{\sigma}\,\Delta) \\
&\boldsymbol{\sigma} : \mathsf{Ty}_{\boldsymbol{\Gamma}}\,\Gamma \to \mathsf{Ty}_{\boldsymbol{\Delta}}\,(\boldsymbol{\sigma}\,\Gamma) \\
&\boldsymbol{\sigma} : \mathsf{Tm}_{\boldsymbol{\Gamma}}\,\Gamma\,A \to \mathsf{Tm}_{\boldsymbol{\Delta}}\,(\boldsymbol{\sigma}\,\Gamma)\,(\boldsymbol{\sigma}\,A)
\end{aligned}$$

We list some of the preservation equations as examples of usage:

$$\begin{aligned}
\boldsymbol{\sigma}\,\bullet &\equiv \bullet \\
\boldsymbol{\sigma}\,(\Gamma \triangleright A) &\equiv \boldsymbol{\sigma}\,\Gamma \triangleright \boldsymbol{\sigma}\,A \\
\boldsymbol{\sigma}\,(A[\sigma]) &\equiv (\boldsymbol{\sigma}\,A)[\boldsymbol{\sigma}\,\sigma] \\
\boldsymbol{\sigma}\,(t[\sigma]) &\equiv (\boldsymbol{\sigma}\,t)[\boldsymbol{\sigma}\,\sigma] \\
\boldsymbol{\sigma}\,(\Sigma\,A\,B) &\equiv \Sigma\,(\boldsymbol{\sigma}\,A)\,(\boldsymbol{\sigma}\,B) \\
\boldsymbol{\sigma}\,(\mathsf{proj}_1\,t) &\equiv \mathsf{proj}_1\,(\boldsymbol{\sigma}\,t)
\end{aligned}$$

Above, we could have also included subscripts indicating the $\boldsymbol{\Gamma}$ or $\boldsymbol{\Delta}$ flcwf, as in $\boldsymbol{\sigma}\,\bullet_{\boldsymbol{\Gamma}} \equiv \bullet_{\boldsymbol{\Delta}}$; but these are quite easily inferable, so we omit them.

Identity morphisms and **composition** are defined in the evident way using identity functions and function composition in underlying maps, and they satisfy the category laws.

The terminal object $\bullet : \mathbf{Con}$ is given by having $\mathsf{Con}_\bullet :\equiv \top$, $\mathsf{Sub}_\bullet\, \Gamma\, \Delta :\equiv \top$, $\mathsf{Ty}_\bullet\, \Gamma :\equiv \top$ and $\mathsf{Tm}_\bullet\, \Gamma\, A :\equiv \top$, and all structure and equations are defined trivially.

Family structure

A type $\boldsymbol{A} : \mathbf{Ty}\, \boldsymbol{\Gamma}$ is a displayed flcwf over $\boldsymbol{\Gamma}$. As we have seen before, displayed algebras can be computed as logical predicate interpretations of algebraic signatures. Every $\boldsymbol{A}$ component lies over the corresponding $\boldsymbol{\Gamma}$ component. Also note that a displayed flcwf includes a displayed category, for which some results have been worked out in [AL19].

Notation 17. In situations where we need to refer to both "base" and displayed things, we give underlined names to contexts, substitutions, types and terms in a base flcwf. For example, we may have $\underline{\Gamma} : \mathsf{Con}_{\boldsymbol{\Gamma}}$ living in $\boldsymbol{\Gamma} : \mathbf{Con}$, and $\Gamma : \mathsf{Con}_{\boldsymbol{A}}\, \underline{\Gamma}$ living in a displayed flcwf over $\boldsymbol{\Gamma}$. We only use underlining on 2LTT variable names, and overload flcwf component names for displayed counterparts. For example, a Con component is named the same in a base flcwf and a displayed one.

Concretely, a displayed flcwf $\boldsymbol{A}$ over $\boldsymbol{\Gamma}$ has the following underlying sets, which we call displayed contexts, substitutions, types and terms respectively.

$$
\begin{aligned}
&\mathsf{Con}_{\boldsymbol{A}} : \mathsf{Con}_{\boldsymbol{\Gamma}} \to \mathsf{Set}\\
&\mathsf{Sub}_{\boldsymbol{A}} : \mathsf{Con}_{\boldsymbol{A}}\, \underline{\Gamma} \to \mathsf{Con}_{\boldsymbol{A}}\, \underline{\Delta} \to \mathsf{Sub}_{\boldsymbol{\Gamma}}\, \underline{\Gamma}\, \underline{\Delta} \to \mathsf{Set}\\
&\mathsf{Ty}_{\boldsymbol{A}} \;\; : \mathsf{Con}_{\boldsymbol{A}}\, \underline{\Gamma} \to \mathsf{Ty}_{\boldsymbol{\Gamma}}\, \underline{\Gamma} \to \mathsf{Set}\\
&\mathsf{Tm}_{\boldsymbol{A}} \;\; : (\Gamma : \mathsf{Con}_{\boldsymbol{A}}\, \underline{\Gamma}) \to \mathsf{Ty}_{\boldsymbol{A}}\, \Gamma\, \underline{A} \to \mathsf{Tm}_{\boldsymbol{\Gamma}}\, \underline{\Gamma}\, \underline{A} \to \mathsf{Set}
\end{aligned}
$$

We list select components of $\boldsymbol{A}$ below; note how every $\boldsymbol{A}$ operation lies over the corresponding $\boldsymbol{\Gamma}$ operation. In our notation with implicit arguments, equations in $\boldsymbol{A}$ can be written the same way as in $\boldsymbol{\Gamma}$, but of course there is extra indexing involved, and the displayed equations are well-typed because of their counterparts in the base.

$$
\begin{aligned}
&\mathsf{id}_{\boldsymbol{A}} \;\; : \mathsf{Sub}\, \Gamma\, \Gamma\, \underline{\mathsf{id}_{\boldsymbol{\Gamma}}}\\
&- \circ_{\boldsymbol{A}} - : \mathsf{Sub}\, \Delta\, \Xi\, \underline{\sigma} \to \mathsf{Sub}\, \Gamma\, \Delta\, \underline{\delta} \to \mathsf{Sub}\, \Gamma\, \Xi\, (\underline{\sigma} \circ_{\boldsymbol{\Gamma}} \underline{\delta})\\
&\mathsf{idl}_{\boldsymbol{A}} \;\; : \mathsf{id}_{\boldsymbol{A}} \circ_{\boldsymbol{A}} \sigma \equiv \sigma
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{idr}_{\boldsymbol{A}} && : \sigma \circ_{\boldsymbol{A}} \mathsf{id}_{\boldsymbol{A}} \equiv \sigma\\
&\bullet_{\boldsymbol{A}} && : \mathsf{Con}_{\boldsymbol{A}}\, \bullet_{\boldsymbol{\Gamma}}\\
&- \triangleright_{\boldsymbol{A}} - && : (\Gamma : \mathsf{Con}_{\boldsymbol{A}}\, \underline{\Gamma}) \to \mathsf{Ty}_{\boldsymbol{A}}\, \Gamma\, \underline{A} \to \mathsf{Con}_{\boldsymbol{A}}\, \Gamma\, (\underline{\Gamma} \triangleright_{\boldsymbol{\Gamma}} \underline{A})\\
&-[-]_{\boldsymbol{A}} && : \mathsf{Ty}_{\boldsymbol{A}}\, \Delta\, \underline{A} \to \mathsf{Sub}_{\boldsymbol{A}}\, \Gamma\, \Delta\, \underline{\sigma} \to \mathsf{Ty}_{\boldsymbol{A}}\, \Gamma\, (\underline{A}[\underline{\sigma}]_{\boldsymbol{\Gamma}})\\
&-[-]_{\boldsymbol{A}} && : \mathsf{Tm}_{\boldsymbol{A}}\, \Delta\, A\, \underline{t} \to (\sigma : \mathsf{Sub}_{\boldsymbol{A}}\, \Gamma\, \Delta\, \underline{\sigma}) \to \mathsf{Tm}_{\boldsymbol{A}}\, \Gamma\, (A[\sigma]_{\boldsymbol{A}})\, (\underline{t}[\underline{\sigma}]_{\boldsymbol{\Gamma}})\\
&\mathsf{Id}_{\boldsymbol{A}} && : \mathsf{Tm}_{\boldsymbol{A}}\, \Gamma\, A\, \underline{t} \to \mathsf{Tm}_{\boldsymbol{A}}\, \Gamma\, A\, \underline{u} \to \mathsf{Ty}_{\boldsymbol{A}}\, \Gamma\, (\mathsf{Id}_{\boldsymbol{\Gamma}}\, \underline{t}\, \underline{u})\\
&\mathsf{K}_{\boldsymbol{A}} && : \mathsf{Con}_{\boldsymbol{A}}\, \underline{\Delta} \to \{\Gamma : \mathsf{Con}_{\boldsymbol{A}}\, \underline{\Gamma}\} \to \mathsf{Ty}_{\boldsymbol{A}}\, \Gamma\, (\mathsf{K}_{\boldsymbol{\Gamma}}\, \underline{\Delta})\\
&\Sigma_{\boldsymbol{A}} && : (A : \mathsf{Ty}_{\boldsymbol{A}}\, \Gamma\, \underline{A}) \to \mathsf{Ty}_{\boldsymbol{A}}\, (\Gamma \triangleright_{\boldsymbol{A}} A)\, \underline{B} \to \mathsf{Ty}_{\boldsymbol{A}}\, \Gamma\, (\Sigma_{\boldsymbol{\Gamma}}\, \underline{A}\, \underline{B})
\end{aligned}
$$

In the following we will often omit $_{\boldsymbol{\Gamma}}$ and $_{\boldsymbol{A}}$ subscripts on components; for example, in the type $\mathsf{Con}_{\boldsymbol{A}}\, \bullet$, the $\bullet$ is clearly a base component in $\boldsymbol{\Gamma}$.

A substituted type $\boldsymbol{A}[\boldsymbol{\sigma}]$ **: Ty** $\boldsymbol{\Gamma}$ is defined as follows, for $\boldsymbol{A}$ **: Ty** $\boldsymbol{\Delta}$ and $\boldsymbol{\sigma}$ **: Sub** $\boldsymbol{\Gamma}\, \boldsymbol{\Delta}$. We simply compose underlying functions in $\boldsymbol{\sigma}$ with the underlying predicates in $\boldsymbol{A}$:

$$
\begin{aligned}
\mathsf{Con}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\, \underline{\Gamma} &:\equiv \mathsf{Con}_{\boldsymbol{A}}\, (\boldsymbol{\sigma}\, \underline{\Gamma})\\
\mathsf{Sub}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\, \Gamma\, \Delta\, \underline{\sigma} &:\equiv \mathsf{Sub}_{\boldsymbol{A}}\, \Gamma\, \Delta\, (\boldsymbol{\sigma}\, \underline{\sigma})\\
\mathsf{Ty}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\, \Gamma\, \underline{A} &:\equiv \mathsf{Ty}_{\boldsymbol{A}}\, \Gamma\, (\boldsymbol{\sigma}\, \underline{A})\\
\mathsf{Tm}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\, \Gamma\, A\, \underline{t} &:\equiv \mathsf{Tm}_{\boldsymbol{A}}\, \Gamma\, A\, (\boldsymbol{\sigma}\, \underline{t})
\end{aligned}
$$

It should be clear that $\boldsymbol{A}[\boldsymbol{\sigma}]$ thus defined still supports all displayed flcwf structure. For example, the displayed contexts in $\boldsymbol{A}[\boldsymbol{\sigma}]$ are elements of $\mathsf{Con}_{\boldsymbol{A}}\, (\boldsymbol{\sigma}\, \underline{\Gamma})$, but since $\boldsymbol{\sigma}$ preserves all $\boldsymbol{\Gamma}$-structure, we can also recover all displayed structure. For example, if $\underline{\Gamma}$ is $\underline{\bullet}$, we have $\boldsymbol{\sigma}\, \underline{\bullet} \equiv \underline{\bullet}$, and we can reuse $\bullet_{\boldsymbol{A}} : \mathsf{Con}_{\boldsymbol{A}}\, \underline{\bullet}$ to define the displayed empty context in $\boldsymbol{A}[\boldsymbol{\sigma}]$, and we can proceed analogously for all other structure in $\boldsymbol{A}[\boldsymbol{\sigma}]$.

Additionally, type substitution is functorial, i.e. $\boldsymbol{A}[\mathbf{id}] \equiv \boldsymbol{A}$ and $\boldsymbol{A}[\boldsymbol{\sigma} \circ \boldsymbol{\delta}] \equiv \boldsymbol{A}[\boldsymbol{\sigma}][\boldsymbol{\delta}]$. This holds because the underlying set families are defined by function composition.

Remark. Types could be equivalently defined as slices in the category of flcws, and type substitution could be given as pullback, but in that case we would run into the well-known strictness issue, that type substitution is functorial only up to isomorphism. This is not a critical issue, as there are standard solutions for recovering strict substitutions from weak ones [KLV12, LW15, CD14]. But if we ever need to look inside the definitions in the model, using displayed algebras yields a lot less encoding overhead than strictifying pullbacks.

A term $\boldsymbol{t} : \mathbf{Tm}\,\boldsymbol{\Gamma}\,\boldsymbol{A}$ is a displayed flcwf section, which again strictly preserves all structure. We use the same notation for the action of $\boldsymbol{t}$ that we use for **Sub**. We have the following underlying maps:

$$\begin{aligned}
&\boldsymbol{t} : (\underline{\Gamma} : \mathsf{Con}_{\boldsymbol{\Gamma}}) \to \mathsf{Con}_{\boldsymbol{A}}\,\underline{\Gamma}\\
&\boldsymbol{t} : (\underline{\sigma} : \mathsf{Sub}_{\boldsymbol{\Gamma}}\,\Gamma\,\Delta) \to \mathsf{Sub}_{\boldsymbol{A}}\,(\boldsymbol{t}\,\Gamma)\,(\boldsymbol{t}\,\Delta)\,\underline{\sigma}\\
&\boldsymbol{t} : (\underline{A} : \mathsf{Ty}_{\boldsymbol{\Gamma}}\,\Gamma) \to \mathsf{Ty}_{\boldsymbol{A}}\,(\boldsymbol{t}\,\Gamma)\,\underline{A}\\
&\boldsymbol{t} : (\underline{t} : \mathsf{Tm}_{\boldsymbol{\Gamma}}\,\Gamma\,A) \to \mathsf{Tm}_{\boldsymbol{A}}\,(\boldsymbol{t}\,\Gamma)\,(\boldsymbol{t}\,A)\,\underline{t}
\end{aligned}$$

A substituted term $\boldsymbol{t}[\boldsymbol{\sigma}]$ for $\boldsymbol{t} : \mathbf{Tm}\,\boldsymbol{\Delta}\,\boldsymbol{A}$ and $\boldsymbol{\sigma} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\boldsymbol{\Delta}$ is again given by component-wise function composition.

An extended context $\boldsymbol{\Gamma} \triangleright \boldsymbol{A}$ is the *total flcwf* of $\boldsymbol{A}$. This is defined by combining corresponding underlying sets with Σ-types:

$$\begin{aligned}
\mathsf{Con}_{\boldsymbol{\Gamma}\triangleright\boldsymbol{A}} &:\equiv (\underline{\Gamma} : \mathsf{Con}_{\boldsymbol{\Gamma}}) \times \mathsf{Con}_{\boldsymbol{A}}\,\underline{\Gamma}\\
\mathsf{Sub}_{\boldsymbol{\Gamma}\triangleright\boldsymbol{A}}\,(\underline{\Gamma},\,\Gamma)\,(\underline{\Delta},\,\Delta) &:\equiv (\underline{\sigma} : \mathsf{Sub}_{\boldsymbol{\Gamma}}\,\underline{\Gamma}\,\underline{\Delta}) \times \mathsf{Sub}_{\boldsymbol{A}}\,\Gamma\,\Delta\,\underline{\sigma}\\
\mathsf{Ty}_{\boldsymbol{\Gamma}\triangleright\boldsymbol{A}}\,(\underline{\Gamma},\,\Gamma) &:\equiv (\underline{A} : \mathsf{Ty}_{\boldsymbol{\Gamma}}\,\underline{\Gamma}) \times \mathsf{Ty}_{\boldsymbol{A}}\,\Gamma\,\underline{A}\\
\mathsf{Tm}_{\boldsymbol{\Gamma}\triangleright\boldsymbol{A}}\,(\underline{\Gamma},\,\Gamma)\,(\underline{A},\,A) &:\equiv (\underline{t} : \mathsf{Tm}_{\boldsymbol{\Gamma}}\,\underline{\Gamma}\,\underline{A}) \times \mathsf{Tm}_{\boldsymbol{A}}\,\Gamma\,A\,\underline{t}
\end{aligned}$$

All structure is defined pointwise, using $\boldsymbol{\Gamma}$-structure for first projections and $\boldsymbol{A}$-structure for second projections. $\boldsymbol{\Gamma} \triangleright \boldsymbol{A}$ may be viewed as a dependent generalization of direct products of flcwfs.

Comprehension structure follows from the above definition: $\mathbf{p}$ is component-wise first projection, $\mathbf{q}$ is second projection and substitution extension $-\mathbf{,}-$ is pairing.

With this, we have a cwf of flcws. *Remark:* flcwf itself is algebraic, and has a finitary QII signature. Hence, if we succeed building semantics for finitary QII signatures, we get "for free" an flcwf of flcwfs. Of course, we cannot rely on this when we are in the process of defining the $\mathbf{M}$ model in the first place. Checking that the $\mathbf{M}$ model indeed works, is the somewhat tedious task that we have to perform *once*, in order to get semantics for any other finitary QII theory.

4.2.5 Type Formers

Strict constant families

This was not included in the ToS specification, but it is quite useful, so we shall define it. $\mathbf{K}\,\boldsymbol{\Delta} : \mathbf{Ty}\,\boldsymbol{\Gamma}$ is defined by ignoring $\boldsymbol{\Gamma}$ inhabitants in all underlying sets:

$$\begin{aligned}
\mathsf{Con}_{\mathbf{K}\,\boldsymbol{\Delta}}\,\underline{\Gamma} &:\equiv \mathsf{Con}_{\boldsymbol{\Delta}} \\
\mathsf{Sub}_{\mathbf{K}\,\boldsymbol{\Delta}}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv \mathsf{Sub}_{\boldsymbol{\Delta}}\,\Gamma\,\Delta \\
\mathsf{Ty}_{\mathbf{K}\,\boldsymbol{\Delta}}\,\Gamma\,\underline{A} &:\equiv \mathsf{Ty}_{\boldsymbol{\Delta}}\,\Gamma \\
\mathsf{Tm}_{\mathbf{K}\,\boldsymbol{\Delta}}\,\Gamma\,A\,\underline{t} &:\equiv \mathsf{Tm}_{\boldsymbol{\Delta}}\,\Gamma\,A
\end{aligned}$$

All structure is inherited from $\boldsymbol{\Delta}$. There is also a type substitution rule, expressing that for $\boldsymbol{\sigma} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\boldsymbol{\Xi}$, we have $(\mathbf{K}\,\{\boldsymbol{\Xi}\}\,\boldsymbol{\Delta})[\boldsymbol{\sigma}] \equiv \mathbf{K}\,\{\boldsymbol{\Gamma}\}\,\boldsymbol{\Delta}$. This follows immediately from the above definition and the definition of type substitution, since the base inhabitants are ignored the same way on both sides of the equation. We also need to show $\mathbf{Tm}\,\boldsymbol{\Gamma}\,(\mathbf{K}\,\boldsymbol{\Delta}) \equiv \mathbf{Sub}\,\boldsymbol{\Gamma}\,\boldsymbol{\Delta}$. This again follows directly from the $\mathbf{K}$ definition. From $\mathbf{K}$, we get

- The unit type, defined as $\mathbf{K}\,\bullet : \mathbf{Ty}\,\boldsymbol{\Gamma}$.
- Direct products of $\boldsymbol{\Gamma}$ and $\boldsymbol{\Delta}$, defined as $\boldsymbol{\Gamma} \triangleright \mathbf{K}\,\boldsymbol{\Delta}$.
- The ability to define closed type formers as elements of $\mathbf{Con}$.

Universe

Similarly to what we did in Definition 32, we define $\mathbf{U}$ as a context, and use $\mathbf{K}$ later to get the universe as a type. $\mathbf{U} : \mathbf{Con}$ is defined to be the flcwf where objects are inner types, and morphisms are outer functions between them:

$$\begin{aligned}
\mathsf{Con}_{\mathbf{U}} &:\equiv \mathsf{Ty}_0 \\
\mathsf{Sub}_{\mathbf{U}}\,\Gamma\,\Delta &:\equiv \mathsf{Tm}_0\,\Gamma \to \mathsf{Tm}_0\,\Delta \\
\mathsf{Ty}_{\mathbf{U}}\,\Gamma &:\equiv \mathsf{Tm}_0\,\Gamma \to \mathsf{Ty}_0 \\
\mathsf{Tm}_{\mathbf{U}}\,\Gamma\,A &:\equiv (\gamma : \mathsf{Tm}_0\,\Gamma) \to \mathsf{Tm}_0\,(A\,\gamma)
\end{aligned}$$

Substitution for types and terms is defined by function composition. The empty context is defined as the inner unit type $\top_0$, and context extension $\Gamma \triangleright_{\mathbf{U}} A$ is defined

as $(\gamma : \Gamma) \times A\,\gamma$ using inner Σ. We can also define Σ_{U} and Id_{U} using inner Σ and identity.

For constant families, we do not need any additional assumption in the inner theory, since it can be defined as $\mathsf{K}_{\mathsf{U}}\,\{\Gamma\}\,\Delta :\equiv \Delta$, and $\mathsf{Sub}_{\mathsf{U}}\,\Gamma\,\Delta \equiv \mathsf{Tm}_{\mathsf{U}}\,\Gamma\,(\mathsf{K}_{\mathsf{U}}\,\Delta)$ follows immediately.

For $\boldsymbol{a} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\mathbf{U}$, we have to define $\mathbf{El}\,\boldsymbol{a} : \mathbf{Ty}\,\boldsymbol{\Gamma}$. This is given as the *displayed flcwf of elements* of $\boldsymbol{a}$.

Background: from any functor $F : \mathbb{C} \to \mathbf{Set}$ we can construct the category of elements $\int F$, where objects are in $(i : |\mathbb{C}|) \times F\,i$ and morphisms between $(i,\, x)$ and $(j,\, y)$ are in $(f : \mathbb{C}(i,\, j)) \times (F\,f\,x \equiv y)$. If we take the second projections of components in $\int F$, we get the displayed category of elements, which lies over $\mathbb{C}$. We may also call this a *discrete displayed category*, in analogy to discrete categories, whose objects are elements of sets.

We extend this to flcwfs in the definition of $\mathbf{El}\,\boldsymbol{a}$. With this definition, $\boldsymbol{\Gamma} \triangleright \mathbf{El}\,\boldsymbol{a}$ will yield the flcwf of elements of $\boldsymbol{a}$.

$$\begin{aligned}
\mathsf{Con}_{\mathbf{El}\,\boldsymbol{a}}\,\underline{\Gamma} &:\equiv \mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{\Gamma}) \\
\mathsf{Sub}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv \boldsymbol{a}\,\underline{\sigma}\,\Gamma \equiv \Delta \\
\mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,\underline{A} &:\equiv \mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{A}\,\Gamma) \\
\mathsf{Tm}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,A\,\underline{t} &:\equiv \boldsymbol{a}\,\underline{t}\,\Gamma \equiv A
\end{aligned}$$

Let us check that we have all other structure as well.

- For contexts and types, the task is to exhibit elements of $\boldsymbol{a}$ lying over specific base contexts and types.
- For terms and substitutions, the task is to exhibit equations which specify the action of $\boldsymbol{a}$.
- Equations between terms and substitutions are trivial because of UIP (we need to show equations between equality proofs).

We summarize below the additional structure on top of the displayed category part of $\mathbf{El}\,\boldsymbol{a}$.

- For $\bullet_{\mathbf{El}\,\boldsymbol{a}} : \mathsf{Con}_{\mathbf{El}\,\boldsymbol{a}}\,\bullet$, the type can be simplified along the definition of $\mathsf{Con}_{\mathbf{El}\,\boldsymbol{a}}$ and structure-preservation by $\boldsymbol{a}$ to $\mathsf{Tm}_0\,\top_0$. Hence, $\bullet_{\mathbf{El}\,\boldsymbol{a}} :\equiv \mathsf{tt}_0$ is the unique

definition. For $\epsilon : \mathsf{Sub}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,\bullet_{\mathbf{El}\,\boldsymbol{a}}\,\underline{\epsilon}$, we have to show $\boldsymbol{a}\,\underline{\epsilon}\,\Gamma \equiv \mathsf{tt}_0$, which holds by the uniqueness of tt_0.

- For $\Gamma \triangleright_{\mathbf{El}\,\boldsymbol{a}} A : \mathsf{Con}_{\mathbf{El}\,\boldsymbol{a}}\,(\underline{\Gamma} \triangleright \underline{A})$, the target type unfolds to $\mathsf{Tm}_0\,(\boldsymbol{a}\,(\underline{\Gamma} \triangleright \underline{A}))$, which in turn simplifies to $\mathsf{Tm}_0\,((\gamma : \boldsymbol{a}\,\underline{\Gamma}) \times \boldsymbol{a}\,\underline{A}\,\gamma)$. Since $\Gamma : \mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{\Gamma})$ and $A : \mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{A}\,\Gamma)$, we define $\Gamma \triangleright_{\mathbf{El}\,\boldsymbol{a}} A$ as $(\Gamma,\, A)$.

- For comprehension, we have to show the following, after simplifying types:

$$\begin{aligned}\mathsf{p} \quad &: \boldsymbol{a}\,\underline{\mathsf{p}}\,(\Gamma,\, A) \equiv \Gamma\\ \mathsf{q} \quad &: \boldsymbol{a}\,\underline{\mathsf{q}}\,(\Gamma,\, A) \equiv A\\ (\sigma,\, t) &: \boldsymbol{a}\,(\underline{\sigma},\, \underline{t})\,\Gamma \equiv (\Delta,\, A)\end{aligned}$$

  For p and q, equations follow from preservation by $\boldsymbol{a}$. For pairing, the goal further simplifies to $(\boldsymbol{a}\,\underline{\sigma}\,\Gamma,\, \boldsymbol{a}\,\underline{t}\,\Gamma) \equiv (\Delta,\, A)$. Then, the first and second components are equal by the σ and t hypotheses.

- Assuming $A : \mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,\Delta\,\underline{A}$ and $\sigma : \mathsf{Sub}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,\Delta\,\underline{\sigma}$, we aim to define $A[\sigma]_{\mathbf{El}\,\boldsymbol{a}} : \mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,(\underline{A}[\underline{\sigma}])$. Simplifying types, $A : \mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{A}\,\Delta)$, $\sigma : \boldsymbol{a}\,\underline{\sigma}\,\Gamma \equiv \Delta$ and the target type is $\mathsf{Tm}_0\,(\boldsymbol{a}\,(\underline{A}[\underline{\sigma}])\,\Gamma)$, which is the same as $\mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{A}\,(\boldsymbol{a}\,\underline{\sigma}\,\Gamma))$, by the preservation of $-[-]$ by $\boldsymbol{a}$. Hence, by the σ assumption, the target type is $\mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{A}\,\Delta)$, so we give the following definition:

$$A[\sigma]_{\mathbf{El}\,\boldsymbol{a}} :\equiv A$$

  This is clearly functorial; moreover, substitution rules for the other type formers hold trivially.

- Term substitution is given by transitivity of equality.

- For $\mathsf{Id}_{\mathbf{El}\,\boldsymbol{a}}\,t\,u : \mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,(\mathsf{Id}\,\underline{t}\,\underline{u})$, the goal type is $\mathsf{Tm}_0\,(\boldsymbol{a}\,(\mathsf{Id}\,\underline{t}\,\underline{u})\,\Gamma)$, hence $\mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{t}\,\Gamma = \boldsymbol{a}\,\underline{u}\,\Gamma)$. This holds by $t : \boldsymbol{a}\,\underline{t}\,\Gamma \equiv A$ and $u : \boldsymbol{a}\,\underline{t}\,\Gamma \equiv A$. Reflexivity and equality reflection are trivial by UIP.

- For $A : \mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,\underline{A}$ and $B : \mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,(\Gamma \triangleright A)\,\underline{B}$, we aim to define $\Sigma_{\mathbf{El}\,\boldsymbol{a}}\,A\,B : \mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,(\Sigma\,\underline{A}\,\underline{B})$, hence

$$\begin{aligned}&\Sigma_{\mathbf{El}\,\boldsymbol{a}}\,A\,B : \mathsf{Tm}_0\,(\boldsymbol{a}\,(\Sigma\,\underline{A}\,\underline{B})\,\Gamma)\\ &\Sigma_{\mathbf{El}\,\boldsymbol{a}}\,A\,B : \mathsf{Tm}_0\,((A : \boldsymbol{a}\,\underline{A}\,\Gamma) \times \boldsymbol{a}\,\underline{B}\,(\Gamma,\, A))\\ &\Sigma_{\mathbf{El}\,\boldsymbol{a}}\,A\,B :\equiv (A,\, B)\end{aligned}$$

Projections and pairing proceed analogously to what we did for comprehension.

- For $\mathsf{K}_{\mathbf{El}\,\boldsymbol{a}}\,\Delta : \mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,(\mathsf{K}\,\underline{\Delta})$, the target type simplifies to $\mathsf{Tm}_0\,(\boldsymbol{a}\,\underline{\Delta})$, hence we have $\mathsf{K}_{\mathbf{El}\,\boldsymbol{a}}\,\Delta :\equiv \Delta$. For the specifying sort equation of K, we have to show

$$\mathsf{Sub}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,\Delta\,\underline{\sigma} \equiv \mathsf{Tm}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,(\mathsf{K}_{\mathbf{El}\,\boldsymbol{a}}\,\Delta)\,\underline{\sigma}$$

where $\underline{\sigma} : \mathsf{Sub}\,\underline{\Gamma}\,\underline{\Delta}$ but at the same time $\underline{\sigma} : \mathsf{Tm}\,\underline{\Gamma}\,(\mathsf{K}\,\underline{\Delta})$ because of the K sort equation in the base. Fortunately, both sides simplify to $\boldsymbol{a}\,\underline{\sigma}\,\Gamma \equiv \Delta$.

We still have to check $(\mathbf{El}\,\boldsymbol{a})[\boldsymbol{\sigma}] \equiv \mathbf{El}\,(\boldsymbol{a} \circ \boldsymbol{\sigma})$, the naturality rule for $\mathbf{El}$. We only have to check equality of underlying sets, Con and Ty formers, since terms and substitutions are equal by UIP. For underlying sets, both sides compute to the following:

$$\begin{aligned}
\mathsf{Con}\,\underline{\Gamma} &:\equiv \mathsf{Tm}_0\,(\boldsymbol{a}\,(\boldsymbol{\sigma}\,\underline{\Gamma})) \\
\mathsf{Sub}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv \boldsymbol{a}\,(\boldsymbol{\sigma}\,\underline{\sigma})\,\Gamma \equiv \Delta \\
\mathsf{Ty}\,\Gamma\,\underline{A} &:\equiv \mathsf{Tm}_0\,(\boldsymbol{a}\,(\boldsymbol{\sigma}\,\underline{A})\,\Gamma) \\
\mathsf{Tm}\,\Gamma\,A\,\underline{t} &:\equiv \boldsymbol{a}\,(\boldsymbol{\sigma}\,\underline{t})\,\Gamma \equiv A
\end{aligned}$$

Since $\boldsymbol{\sigma}$ also strictly preserves all structure, and we simply replace $\boldsymbol{a}$ action by the composite $\boldsymbol{a} \circ \boldsymbol{\sigma}$ action, it is straightforward to check that Con and Ty formers are also the same on both sides.

At this point, we have $\mathbf{U} : \mathbf{Con}$ and $\mathbf{El} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\mathbf{U}$. Let us rename them to $\mathbf{U}'$ and $\mathbf{El}'$ respectively, and define the usual "open" versions:

$$\begin{aligned}
\mathbf{U} &: \mathbf{Ty}\,\boldsymbol{\Gamma} & \mathbf{El} &: \mathbf{Tm}\,\boldsymbol{\Gamma}\,\mathbf{U} \to \mathbf{Ty}\,\boldsymbol{\Gamma} \\
\mathbf{U} &:\equiv \mathbf{K}\,\mathbf{U}' & \mathbf{El}\,\boldsymbol{a} &:\equiv \mathbf{El}'\,\boldsymbol{a}
\end{aligned}$$

Identity

Assuming $\boldsymbol{t}, \boldsymbol{u} : \mathbf{Tm}\,\boldsymbol{\Gamma}\,\boldsymbol{A}$, extensional identity $\mathbf{Id}\,\boldsymbol{t}\,\boldsymbol{u}$ is defined as component-wise equality:

$$\begin{aligned}
\mathsf{Con}_{\mathbf{Id}\,\boldsymbol{t}\,\boldsymbol{u}}\,\underline{\Gamma} &:\equiv \boldsymbol{t}\,\underline{\Gamma} \equiv \boldsymbol{u}\,\underline{\Gamma} \\
\mathsf{Sub}_{\mathbf{Id}\,\boldsymbol{t}\,\boldsymbol{u}}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv \boldsymbol{t}\,\underline{\sigma} \equiv \boldsymbol{u}\,\underline{\sigma} \\
\mathsf{Ty}_{\mathbf{Id}\,\boldsymbol{t}\,\boldsymbol{u}}\,\Gamma\,\underline{A} &:\equiv \boldsymbol{t}\,\underline{A} \equiv \boldsymbol{u}\,\underline{A} \\
\mathsf{Tm}_{\mathbf{Id}\,\boldsymbol{t}\,\boldsymbol{u}}\,\Gamma\,A\,\underline{t} &:\equiv \boldsymbol{t}\,\underline{t} \equiv \boldsymbol{u}\,\underline{t}
\end{aligned}$$

All other structure follows from structure-preservation of $\boldsymbol{t}$ and $\boldsymbol{u}$. For the simplest example, $\bullet_{\mathsf{Id}\,\boldsymbol{t}\,\boldsymbol{u}} : \boldsymbol{t}\,\underline{\bullet} \equiv \boldsymbol{u}\,\underline{\bullet}$ holds because $\boldsymbol{t}$ and $\boldsymbol{u}$ both preserve $\underline{\bullet}$. The rule $(\mathsf{Id}\,\boldsymbol{t}\,\boldsymbol{u})[\boldsymbol{\sigma}] \equiv \mathsf{Id}\,(\boldsymbol{t}[\boldsymbol{\sigma}])\,(\boldsymbol{u}[\boldsymbol{\sigma}])$ is straightforward to check: we only have to look at the underlying sets, where e.g. both sides have $\mathsf{Con}\,\underline{\Gamma} \equiv (\boldsymbol{t}\,(\boldsymbol{\sigma}\,\underline{\Gamma}) \equiv \boldsymbol{u}\,(\boldsymbol{\sigma}\,\underline{\Gamma}))$. It is also evident that $\mathsf{Tm}\,\boldsymbol{\Gamma}\,(\mathsf{Id}\,\boldsymbol{t}\,\boldsymbol{u})$ is equivalent to $\boldsymbol{t} \equiv \boldsymbol{u}$, that is, we have reflexivity and equality reflection.

Inductive function type

For $\boldsymbol{a} : \mathsf{Tm}\,\boldsymbol{\Gamma}\,\mathsf{U}$ and $\boldsymbol{B} : \mathsf{Ty}\,(\boldsymbol{\Gamma} \triangleright \mathsf{El}\,\boldsymbol{a})$, we aim to define $\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B} : \mathsf{Ty}\,\boldsymbol{\Gamma}$. This is a dependent product of displayed flcwfs, indexed over a *discrete* domain. Discreteness is critical: since morphisms in $\mathsf{El}\,\boldsymbol{a}$ are proof-irrelevant and invertible (because they are equations), we avoid the variance issues that preclude general Π-types in the cwf of categories [Joh02, Secion A1.5].

The direct definition would be to define underlying sets as products, indexed over corresponding components in $\mathsf{El}\,\boldsymbol{a}$:

$$
\begin{aligned}
\mathsf{Con}_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\underline{\Gamma} &:\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Con}_{\boldsymbol{B}}\,(\underline{\Gamma},\,\gamma)\\
\mathsf{Sub}_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv \{\gamma : \boldsymbol{a}\,\underline{\Gamma}\}\{\delta : \boldsymbol{a}\,\underline{\Delta}\}(\sigma : \mathsf{Sub}_{\mathsf{El}\,\boldsymbol{a}}\,\gamma\,\delta\,\underline{\sigma}) \to \mathsf{Sub}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\Delta\,\delta)\,(\underline{\sigma},\,\sigma)\\
\mathsf{Ty}_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,\underline{A} &:\equiv \{\gamma : \boldsymbol{a}\,\underline{\Gamma}\}(\alpha : \boldsymbol{a}\,\underline{A}\,\gamma) \to \mathsf{Ty}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\underline{A},\,\alpha)\\
\mathsf{Tm}_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,A\,\underline{t} &:\equiv \{\gamma : \boldsymbol{a}\,\underline{\Gamma}\}\{\alpha : \boldsymbol{a}\,\underline{A}\,\gamma\}(t : \mathsf{Tm}_{\mathsf{El}\,\boldsymbol{a}}\,\gamma\,\delta\,\underline{t}) \to \mathsf{Tm}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\mathsf{A}\,\alpha)\,(\underline{t},\,t)
\end{aligned}
$$

But just like in Definitions 4 and 6, we can contract the Sub and Tm definitions, since $\mathsf{Sub}_{\mathsf{El}\,\boldsymbol{a}}\,\gamma\,\delta\,\underline{\sigma} \equiv (\boldsymbol{a}\,\underline{\sigma}\,\gamma \equiv \delta)$ and $\mathsf{Tm}_{\mathsf{El}\,\boldsymbol{a}}\,\gamma\,\alpha\,\underline{t} \equiv (\boldsymbol{a}\,\underline{t}\,\gamma \equiv \alpha)$.

$$
\begin{aligned}
\mathsf{Con}_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\underline{\Gamma} &:\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Con}_{\boldsymbol{B}}\,(\underline{\Gamma},\,\gamma)\\
\mathsf{Sub}_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Sub}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\Delta\,(\boldsymbol{a}\,\underline{\sigma}\,\gamma))\,(\underline{\sigma},\,\mathsf{refl})\\
\mathsf{Ty}_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,\underline{A} &:\equiv \{\gamma : \boldsymbol{a}\,\underline{\Gamma}\}(\alpha : \boldsymbol{a}\,\underline{A}\,\gamma) \to \mathsf{Ty}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\underline{A},\,\alpha)\\
\mathsf{Tm}_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,A\,\underline{t} &:\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Tm}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\mathsf{A}\,(\boldsymbol{a}\,\underline{t}\,\gamma))\,(\underline{t},\,\mathsf{refl})
\end{aligned}
$$

With the contracted definition, Sub and Tm are only indexed over displayed objects and types, but not over displayed morphisms or terms anymore. So it is apparent that we cannot have issues with indexing variance. All structure in $\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}$ is pointwise inherited from $\boldsymbol{B}$. We list some examples below for definitions.

$$
\begin{aligned}
\bullet_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\gamma &:\equiv \bullet_{\boldsymbol{B}}\\
(\Gamma \triangleright_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}} A)\,(\gamma,\,\alpha) &:\equiv (\Gamma\,\gamma \triangleright_{\boldsymbol{B}} A\,\alpha)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{id}_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\gamma &:\equiv \mathsf{id}_{\boldsymbol{B}} \\
(\sigma \circ_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}} \delta)\,\gamma &:\equiv \sigma\,\gamma \circ_{\boldsymbol{B}} \delta\,\gamma \\
A[\sigma]_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\{\gamma\}\,\alpha &:\equiv (A\,\alpha)[\sigma\,\gamma]_{\boldsymbol{B}} \\
\mathsf{K}_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Delta\,\alpha &:\equiv \mathsf{K}_{\boldsymbol{B}}\,(\Delta\,\alpha)
\end{aligned}
$$

For the specifying isomorphism $\mathbf{(app,\ lam)} : \mathbf{Tm}\,\mathbf{\Gamma}\,(\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}) \simeq \mathbf{Tm}\,(\mathbf{\Gamma} \triangleright \mathbf{El}\,\boldsymbol{a})\,\boldsymbol{B}$, note that the difference in presentation is exactly component-wise currying and uncurrying. For instance, in $\boldsymbol{t} : \mathbf{Tm}\,\mathbf{\Gamma}\,(\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B})$, the underlying action on contexts has the following type:

$$(\underline{\Gamma} : \mathsf{Con}_{\mathbf{\Gamma}})(\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Con}_{\boldsymbol{B}}\,(\underline{\Gamma},\,\gamma)$$

While in $\boldsymbol{t} : \mathbf{Tm}\,(\mathbf{\Gamma} \triangleright \mathbf{El}\,\boldsymbol{a})\,\boldsymbol{B}$, we have

$$((\underline{\Gamma},\,\gamma) : (\underline{\Gamma} : \mathsf{Con}_{\mathbf{\Gamma}}) \times \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Con}_{\boldsymbol{B}}\,(\underline{\Gamma},\,\gamma)$$

So **app** and **lam** are defined as component-wise uncurrying and currying respectively. Naturality of $\mathbf{\Pi}$ and **app** again follows from the fact that flcwf morphisms strictly preserve all structure, and substitution is component-wise function composition.

External function type

For $Ix : \mathsf{Ty}_0$ and $\boldsymbol{B} : \mathsf{Tm}_0\,Ix \to \mathbf{Ty}\,\mathbf{\Gamma}$, we define $\mathbf{\Pi}^{\mathbf{ext}}\,Ix\,\boldsymbol{B} : \mathbf{Ty}\,\mathbf{\Gamma}$ as the Ix-indexed direct product of a family of displayed flcwfs.

$$
\begin{aligned}
\mathsf{Con}_{\mathbf{\Pi}^{\mathbf{ext}}\,Ix\,\boldsymbol{B}}\,\underline{\Gamma} &:\equiv (i : \mathsf{Tm}_0\,Ix) \to \mathsf{Con}_{\boldsymbol{B}\,i}\,\underline{\Gamma} \\
\mathsf{Sub}_{\mathbf{\Pi}^{\mathbf{ext}}\,Ix\,\boldsymbol{B}}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv (i : \mathsf{Tm}_0\,Ix) \to \mathsf{Sub}_{\boldsymbol{B}\,i}\,(\Gamma\,i)\,(\Delta\,i)\,\underline{\sigma} \\
\mathsf{Ty}_{\mathbf{\Pi}^{\mathbf{ext}}\,Ix\,\boldsymbol{B}}\,\Gamma\,\underline{A} &:\equiv (i : \mathsf{Tm}_0\,Ix) \to \mathsf{Ty}_{\boldsymbol{B}\,i}\,(\Gamma\,i)\,\underline{A} \\
\mathsf{Tm}_{\mathbf{\Pi}^{\mathbf{ext}}\,Ix\,\boldsymbol{B}}\,\Gamma\,A\,\underline{t} &:\equiv (i : \mathsf{Tm}_0\,Ix) \to \mathsf{Tm}_{\boldsymbol{B}\,i}\,(\Gamma\,i)\,(A\,i)\,\underline{t}
\end{aligned}
$$

All structure is defined in the evident pointwise way. $\mathbf{app}^{\mathbf{ext}}$ and $\mathbf{lam}^{\mathbf{ext}}$ are defined by component-wise flipping of function arguments. This concludes the definition of the $\mathbf{M}$ model.

Example 16. We look at the computation of a semantic flcwf, in the simple case of the flcwf of Nat-algebras. Recall that the signature is

$$\mathsf{NatSig} :\equiv \bullet \triangleright (N : \mathsf{U}) \triangleright (\mathsf{zero} : \mathsf{El}\,N) \triangleright (\mathsf{suc} : N \Rightarrow \mathsf{El}\,N)$$

We evaluate NatSig in $\mathbf{M}$ entry-wise. We start from $\bullet$, the terminal flcwf where algebras are elements of $\top$. Then, moving left to right, we take the total flcwf of each type in the signature. From U, we get the product of $\top$ and the flcwf of sets, which is equivalent to simply the flcwf of sets. Second, we extend this with the semantic $\mathsf{El}\,N$, which is the *displayed flcwf of points of sets*, to get the flcwf of pointed sets. Finally, by extension with $N \Rightarrow \mathsf{El}\,N$, we get the flcwf of Nat-algebras.

Let us also look at some components of the resulting flcwf. Algebras, displayed algebras, morphisms and sections have been already discussed before, so we look at other components. We omit the leading $\top$ components everywhere in the following.

$\bullet$ is the terminal Nat-algebra, i.e. $\bullet \equiv (\top,\, \mathsf{tt},\, \lambda\,\_.\,\mathsf{tt})$. Context extension $- \triangleright -$: $(\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$ constructs the total algebra of a displayed algebra.

$$\begin{aligned}&(N,\, z,\, s) \triangleright (N^D,\, z^D,\, s^D) \equiv\\ &\quad(((n : N) \times N^D\,n),\, (z,\, z^D),\, (\lambda\,(n,\, n^D).\,(s\,n,\, s^D\,n\,n^D)))\end{aligned}$$

p and q respectively project first and second components from a total algebra. For $t,\, u : \mathsf{Tm}\,(N,\, z,\, s)\,(N^D,\, z^D,\, s^D)$, $\mathsf{Id}\,t\,u$ is the displayed Nat-algebra which expresses equality of Nat-algebra sections. Let us review the definition of sections:

$$\begin{aligned}&\mathsf{Tm}\,(N,\, z,\, s)\,(N^D,\, z^D,\, s^D) \equiv\\ &\quad(N^S : (n : N) \to N^D\,n)\\ &\times (z^S\ \ : N^S\,z \equiv z^D)\\ &\times (s^S\ \ : (n : N) \to N^S\,(s\,n) \equiv s^D\,n\,(N^S\,n))\end{aligned}$$

We have that

$$\begin{aligned}&\mathsf{Id}\,(N_0^S,\, z_0^S,\, s_0^S)\,(N_1^S,\, z_1^S,\, s_1^S) \equiv\\ &\quad((\lambda\,n.\,N_0^S\,n \equiv N_1^S\,n),\, (\_ : N_0^S\,z \equiv N_1^S\,z),\, (\lambda\,n.\,(\_ : N_0^S\,(s\,n) \equiv N_1^S\,(s\,n))))\end{aligned}$$

The underscores denote omitted equality proofs; they follow from the z^S and s^S components. It should be apparent that $\mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,t\,u)$ is isomorphic to $t \equiv u$; this follows from function extensionality and decomposition of equalities of pairs. Thus, equality reflection holds in the flcwf of Nat-algebras. Note that we do not need to use equality reflection for $- \equiv -$ to show this; it is simply a reshuffling of components along funext.

$\mathsf{K} : \mathsf{Con} \to \{\Delta : \mathsf{Con}\} \to \mathsf{Ty}\,\Delta$ yields a non-dependent displayed algebra:

$$\mathsf{K}\,(N,\, z,\, s)\,\{N',\, z',\, s'\} \equiv (\lambda\,\_.\,N,\, z,\, \lambda\,n\,\_.\,s\,n)$$

With this definition, we indeed have that $\mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta) \equiv \mathsf{Sub}\,\Gamma\,\Delta$.

$\Sigma : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$ is the evident parameterized variant of $- \triangleright -$:

$$\begin{aligned}
&\Sigma\,(N^D,\, z^D,\, s^D)\,(N^{D'},\, z^{D'},\, s^{D'}) :\equiv \\
&\quad((\lambda\, n.\,(n^D : N^D\, n) \times N^{D'}\,(n,\, n^D)),\\
&\quad(z^D,\, z^{D'}),\\
&\quad(\lambda\, n\,(n^D,\, n^{D'}).\,(s^D\, n\, n^D,\, s^{D'}\,(n,\, s^D\, n\, n^D)\, n^{D'})))
\end{aligned}$$

4.3 Left Adjoints of Substitutions

In this section we show that if all signatures have initial algebras, then the semantic interpretation of each $\nu : \mathsf{Sub}\,\Omega\,\Delta$ has a left adjoint functor. We have the following setup.

- We write $[\![-]\!]$ for the interpretation into the flcwf model M.
- We close types in ToS under $\top$ and Σ, that is, we have $\top : \mathsf{Ty}\,\Gamma$ and $\Sigma : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$. The flcwf semantics can be immediately extended with these type formers: since flcwfs are given by an FQII signature, they form an flcwf themselves and support $\top$ (as $\mathsf{K}\,\bullet$) and Σ. In the following we will need to talk about signatures depending on signatures, and $\top$ and Σ are more convenient for this purpose than telescopes.

Given $\nu : \mathsf{Sub}\,\Omega\,\Delta$ in the ToS syntax, we get $[\![\nu]\!] : [\![\Omega]\!] \to [\![\Delta]\!]$ as a functor between $[\![\Omega]\!]$ and $[\![\Delta]\!]$ categories of algebras. We seek to construct some $\mathsf{L} : [\![\Delta]\!] \to [\![\Omega]\!]$ such that $\mathsf{L} \dashv [\![\nu]\!]$.

The basic idea is the following: the existence of left adjoints is equivalently characterized by having an initial object in the comma category $\delta/[\![\nu]\!]$ for each $\delta : \Delta^A$ [ML98, Section IV]. Thus, it is enough to find some signature Ω such that $[\![\Omega]\!]$ is equivalent to $\delta/[\![\nu]\!]$, and by assumption we get an initial object. The objects of $\delta/[\![\nu]\!]$ consist of the following:

$$(\omega : \Omega^A) \times (\eta : \Delta^M\,\delta\,(\nu^A\,\omega))$$

Of the two components, $\omega : \Omega^A$ can be clearly represented as the Ω signature. The η component is a bit more complicated. We need to represent a Δ-morphism, but

whose domain is an external algebra, and whose codomain is an algebra internal to the ToS syntax. In other words, we need a notion of "heterogeneous" morphism, where the domain lives in the usual flcwf semantics, but the codomain lives in the syntactic slice model ToS/Ω.

Definition 45 (**Heterogeneous morphisms**)**.** Fixing $\Omega : \mathsf{Con}$, we define $-^{HM}$ by induction on the ToS.

$$
\begin{aligned}
&-^{HM} : (\Gamma : \mathsf{Con}) \to \Gamma^A \to \mathsf{Sub}\,\Omega\,\Gamma \to \mathsf{Ty}\,\Omega\\
&-^{HM} : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Omega\,(\Gamma^{HM}\,\gamma_0\,\gamma_1) \to \mathsf{Tm}\,\Omega\,(\Delta^{HM}\,(\sigma^A\,\gamma_0)\,(\sigma \circ \gamma_1))\\
&-^{HM} : (A : \mathsf{Ty}\,\Gamma) \to A^A\,\gamma_0 \to \mathsf{Tm}\,\Omega\,(A[\gamma_1]) \to \mathsf{Tm}\,\Omega\,(\Gamma^{HM}\,\gamma_0\,\gamma_1) \to \mathsf{Ty}\,\Omega\\
&-^{HM} : (t : \mathsf{Tm}\,\Gamma\,A)(\gamma^{HM} : \mathsf{Tm}\,\Omega\,(\Gamma^{HM}\,\gamma_0\,\gamma_1)) \to \mathsf{Tm}\,\Omega\,(A^{HM}\,(t^A\,\gamma_0)\,(t[\gamma_1])\,\gamma^{HM})
\end{aligned}
$$

The interpretation on contexts sums up the difference between the "homogeneous" $-^{HM}$ and the current one. In the homogeneous interpretation, we have $\Gamma^{HM} : \Gamma^A \to \Gamma^A \to \mathsf{Set}$, in the heterogeneous one the codomain of the relation is syntactic, and the return type as well. We use $\top$ and Σ in ToS to interpret contexts:

$$
\begin{aligned}
&\bullet^{HM}\,\gamma_0\,\gamma_1 &&:\equiv \top\\
&(\Gamma \triangleright A)^{HM}\,(\gamma_0,\,\alpha_0)\,(\gamma_1\,\alpha_1) &&:\equiv \Sigma\,(\gamma^{HM} : \Gamma^{HM}\,\gamma_0\,\gamma_1)\,(A^{HM}\,\alpha_0\,\alpha_1\,\gamma^{HM})
\end{aligned}
$$

We use a nameful notation for Σ-binding on the right hand side. In the cwf interpretation we similarly reuse ToS type formers in a mechanical way, following the definitions of the homogeneous $-^{HM}$.
U is interpreted using external function types:

$$
\begin{aligned}
&\mathsf{U}^{HM} : (a_0 : \mathsf{Ty}_0)(a_1 : \mathsf{Tm}\,\Omega\,\mathsf{U}) \to \mathsf{Tm}\,\Omega\,(\Gamma^{HM}\,\gamma_0\,\gamma_1) \to \mathsf{Ty}\,\Omega\\
&\mathsf{U}^{HM}\,a_0\,a_1\,\gamma^{HM} :\equiv a_0 \to^{\mathsf{ext}} \mathsf{El}\,a_1
\end{aligned}
$$

Note that this does not work if a_0 is syntactic and a_1 is external, as we have no function type in ToS with external codomain; so $-^{HM}$ would not work with an external second parameter. El^{HM} uses the Id type in ToS:

$$
\begin{aligned}
&(\mathsf{El}\,a)^{HM} : a^A\,\gamma_0 \to \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\gamma_1])) \to \mathsf{Ty}\,\Omega\\
&(\mathsf{El}\,a)^{HM}\,\alpha_0\,\alpha_1\,\gamma^{HM} :\equiv \mathsf{Id}\,(a^{HM}\,\gamma^{HM}\,\alpha_0)\,\alpha_1
\end{aligned}
$$

In Π we give the usual pointwise definition, using the external function type:

$$
(\Pi\,a\,B)^{HM}\,t_0\,t_1\,\gamma^{HM} :\equiv (\alpha : a^A\,\gamma_0) \to^{\mathsf{ext}} B^{HM}\,(t_0\,\alpha)\,(t_1\,(a^{HM}\,\gamma^{HM}\,\alpha))\,(\gamma^{HM},\,\mathsf{refl})
$$

In Id, we reuse the Id in ToS:

$$(\mathsf{Id}\, t\, u)^{HM} : t^A\, \gamma_0 \equiv u^A\, \gamma_0 \to \mathsf{Tm}\, \Omega\, (\mathsf{Id}\, (t[\gamma_1])\, (u[\gamma_1])) \to \mathsf{Tm}\, \Omega\, (\Gamma^{HM}\, \gamma_0\, \gamma_1) \to \mathsf{Ty}\, \Omega$$
$$(\mathsf{Id}\, t\, u)^{HM}\, p_0\, p_1\, \gamma^{HM} :\equiv \mathsf{Id}\, (t^{HM}\, \gamma^{HM})\, (u^{HM}\, \gamma^{HM})$$

External functions are again external functions.

$$(\Pi^{\mathsf{ext}}\, \mathsf{Ix}\, B)^{HM}\, t_0\, t_1\, \gamma^{HM} :\equiv (i : \mathsf{Ix}) \to^{\mathsf{ext}} (B\, i)^{HM}\, (t_0\, i)\, (t_1\, i)\, \gamma^{HM}$$

The newly added $\top$ and Σ type formers are evident:

$$\begin{aligned}
&\top^{HM}\, \mathsf{tt}\, \mathsf{tt}\, \gamma^M :\equiv \top \\
&(\Sigma\, A\, B)^{HM}\, (\alpha_0,\, \beta_0)\, (\alpha_1,\, \beta_1) :\equiv \\
&\qquad \Sigma\, (\alpha^{HM} : A^{HM}\, \alpha_0\, \alpha_1\, \gamma^M)\, (B^{HM}\, \beta_0\, \beta_1\, (\gamma^{HM},\, \alpha^{HM}))
\end{aligned}$$

Definition 46 (**Representing signature**)**.** For $\nu : \mathsf{Sub}\, \Omega\, \Delta$ and $\delta : \Delta^A$, we define the signature which represents $\delta/[\![\nu]\!]$:

$$\mathsf{Sig}_{\delta/[\![\nu]\!]} :\equiv \Omega \triangleright \Delta^{HM}\, \delta\, \nu$$

Now, we have that

$$\begin{aligned}
(\mathsf{Sig}_{\delta/[\![\nu]\!]})^A &\equiv (\omega : \Omega^A) \times ((\Delta^{HM}\, \delta\, \nu)^A\, \omega) \\
(\mathsf{Sig}_{\delta/[\![\nu]\!]})^M\, (\omega_0,\, \eta_0)\, (\omega_1,\, \eta_1) &\equiv (\omega^M : \Omega^M\, \omega_0\, \omega_1) \times ((\Delta^{HM}\, \delta\, \nu)^M\, \eta_0\, \eta_1\, \omega^M)
\end{aligned}$$

It remains to show that $[\![\mathsf{Sig}_{\delta/[\![\nu]\!]}]\!]$ is indeed equivalent to $\delta/[\![\nu]\!]$. It suffices to show that sets of objects and morphisms are isomorphic. We need the following:

$$\begin{aligned}
(\Delta^{HM}\, \delta\, \nu)^A\, \omega &\simeq \Delta^M\, \delta\, (\nu^A\, \omega) \\
(\Delta^{HM}\, \delta\, \nu)^M\, \eta_0\, \eta_1\, \omega^M &\simeq (\nu^M\, \omega^M \circ \eta_0 \equiv \eta_1)
\end{aligned}$$

These can be shown by induction on ToS again; we omit describing this here.

Theorem 4. *If every FQII signature has an initial algebra, then for every* $\nu : \mathsf{Sub}\, \Omega\, \Delta$*, there exists a left adjoint of* $[\![\nu]\!] : [\![\Omega]\!] \to [\![\Delta]\!]$*.*

Proof. For each $\delta : \Delta^A$, the comma category $\delta/[\![\nu]\!]$ can be specified with $\mathsf{Sig}_{\delta/[\![\nu]\!]}$ by Definition 46, hence it has an initial object. The left adjoint $\mathsf{L} : [\![\Delta]\!] \to [\![\Omega]\!]$ sends each $\delta : \Delta^A$ to the $\omega : \Omega^A$ component of the initial algebra of $\mathsf{Sig}_{\delta/[\![\nu]\!]}$. □

4.4 Discussion of Semantics

4.4.1 Flcwfs For Free

We give a quick summary for using the semantics of FQII signatures. As input we pick a) a signature Γ b) a cwf $\mathbb{C}$ with Σ, $\top$ and extensional Id. Then, we interpret the signature in $\mathbf{M}$, thereby getting an flcwf in 2LTT. Then, we interpret that in presheaves over $\mathbb{C}$, and we get the flcwf whose objects are internal Γ-algebras in $\mathbb{C}$.

One use case is in building models of certain type theories. Usually, this starts with constructing the base cwf. But if the objects can be specified using an FQII signature, we get an flcwf for free. In some cases, we get exactly what is needed. For example, the flcwf of presheaves can be used as it is in the presheaf models of type theories.

In other cases, the flcwf that we get has to be extended in some ways. This often happens if the objects in the model have an internal notion of "equivalence" which has to be respected by types.

- In the setoid model, objects are setoids and types are displayed setoids with additional fibrancy structure [ABKT19].
- The groupoid model [HS96] is analogous; again types are displayed groupoids with fibrancy structure.
- Likewise, in the cubical set model [BCH14], types are displayed presheaves together with fibrancy structure (Kan composition).

In all these cases, the semantic objects have FQII signatures. We can interpret their flwcfs in $\mathbf{Set}$ and add fibrancy conditions. The cubical set model is presented exactly in this way in [BCH14], using displayed algebras. The groupoid model in [HS96] instead presents types as $\mathbf{\Gamma} \to \mathbf{Gpd}$ functors, i.e. uses an indexed style instead of the displayed style.

In the indexed-style groupoid model, we get strictly functorial type substitution, just like in the displayed style. However, the displayed style appears to be a more general way to get strict substitution, as it works for every FQII theory. Again, although finitely complete categories can be always strictified to cwfs, if we ever need to perform calculations with the internal definitions of a model, the displayed style is much more direct.

4.4.2 Variations of the Semantics

In Section 4.1, we required that the inner theory has Σ, $\top$ and extensional Id, and then used the assumed type formers in the definition of $\mathbf{U}$. Hence, when we interpret the semantic flcwf of a signature in the presheaf model, we again need to assume these type formers in the base cwf $\mathbb{C}$.

However, we can drop Id from the requirements on the inner theory, and likewise drop the identity type from flcwfs, and the model still works. In this case we have a somewhat more general semantics. In particular, like in Section 3.5.2, we can interpret signatures in finite product categories, because $\top$ and Σ can be derived from finite products in the constructed "simply typed" cwf. On the other hand, we get less out of the semantics. For instance, we cannot show equivalence of initiality and induction without Id.

If we want to trim down the assumptions on the inner theory to the minimum, we can make do with simply an inner cwf with no type formers at all. This implies that for each signature we can build a category of algebras, plus extra structure which does not require Σ or $\top$ in the $\mathbf{U}$ definition. So we may have displayed algebras, sections, and also functorial substitution for these, but we do not have terminal algebras and total algebras.

We could also add more type formers to the semantics. We may add *small limits* via the external function type Π^{ext} that we already have in signatures. Extending flcwfs with Π^{ext} requires Π-types in the inner theory of 2LTT, hence in $\mathbb{C}$ as well. The reason is that indexed products of algebras require functions in the underlying sorts. More concretely, in the definition of $\mathbf{U}$, we have to interpret

$$\Pi^{\mathsf{ext}}_{\mathbf{U}} : (Ix : \mathsf{Ty}_0) \to (\mathsf{Tm}_0\, Ix \to \mathsf{Ty}_{\mathbf{U}}\, \Gamma) \to \mathsf{Ty}_{\mathbf{U}}\, \Gamma$$

hence

$$\Pi^{\mathsf{ext}}_{\mathbf{U}} : (Ix : \mathsf{Ty}_0) \to (\mathsf{Tm}_0\, Ix \to \mathsf{Tm}_0\, \Gamma \to \mathsf{Ty}_0) \to \mathsf{Tm}_0\, \Gamma \to \mathsf{Ty}_0$$

This works if we can return an inner Π in the definition:

$$\Pi^{\mathsf{ext}}_{\mathbf{U}}\, Ix\, B\, \gamma :\equiv (i : Ix) \to B\, i\, \gamma$$

In this case, the "small limit" cwf semantics can be completed. We omit checking the details here. From Π^{ext}, we also recover indexed products, by using $\Pi^{\mathsf{ext}}\, Ix\, (\lambda\, i.\, \mathsf{K}\, \Gamma_i)$, where Γ_i is an indexed family of objects.

With the small limit semantics, if we want to have a simply typed interpretation, we can start with a cartesian closed $\mathbb{C}$.

4.4.3 Substitutions

Interpreting signatures is not the only potentially useful thing that we get out of the semantics. Each $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$ can be viewed as a free interpretation of the Δ theory in Γ, and we get a strict flcwf morphism from the semantics.

Ornaments

One use case of Sub is to specify *ornaments* [Dag17], i.e. ways to decorate structures with additional information, or dually, to erase parts of some structure. Ornaments differ from the usual forgetful maps, because they forget structure in *negative* position, i.e. in assumptions of construction rules.

Example 17. We assume $A : \mathsf{Ty}_0$. We define the substitution which forgets elements of A-lists.

$$\begin{aligned}\sigma : \mathsf{Sub}\,(\bullet \triangleright (\mathit{Nat} : \mathsf{U}) \triangleright (\mathit{zero} : \mathsf{El}\,\mathit{Nat}) \triangleright (\mathit{suc} : \mathit{Nat}))\\ (\bullet \triangleright (\mathit{List} : \mathsf{U}) \triangleright (\mathit{nil} : \mathsf{El}\,\mathit{List}) \triangleright (\mathit{cons} : A \to^{\mathsf{ext}} \mathit{List} \to \mathit{List}))\end{aligned}$$

The map goes from numbers to lists because of the "contravariant" forgetfulness. We define σ by listing its component definitions.

$$\begin{aligned}\mathsf{List} &:\equiv \mathsf{Nat}\\ \mathsf{nil} &:\equiv \mathsf{zero}\\ \mathsf{cons} &:\equiv \lambda^{ext}\,\_.\,\lambda\, n.\,\mathsf{suc}\, n\end{aligned}$$

Example 18. We assume $\mathsf{Nat}_0 : \mathsf{Ty}_0$ with zero_0 and suc_0, and define $\sigma : \mathsf{Sub}\,\mathsf{NatSig}\,\mathsf{FinSig}$, where FinSig is as follows:

$$\begin{aligned}\mathsf{Fin} &: \mathsf{Nat}_0 \to^{\mathsf{ext}} \mathsf{U}\\ \mathsf{zero} &: (n : \mathsf{Nat}_0) \to^{\mathsf{ext}} \mathsf{El}\,(\mathsf{Fin}\,(\mathsf{suc}_0\, n))\\ \mathsf{suc} &: (n : \mathsf{Nat}_0) \to^{\mathsf{ext}} \mathsf{Fin}\, n \to \mathsf{El}\,(\mathsf{Fin}\,(\mathsf{suc}_0\, n))\end{aligned}$$

σ is defined as

$$\begin{aligned}\mathsf{Fin} &:\equiv \mathsf{Nat}\\ \mathsf{zero} &:\equiv \lambda^{ext}\,\_.\,\mathsf{zero}\\ \mathsf{suc} &:\equiv \lambda^{ext}\,\_.\,\lambda\, n.\,\mathsf{suc}\, n\end{aligned}$$

For a specific programming use case, if we have any recursive function defined on an "erased" type, we can convert that to a recursive function which acts on an "ornamented" type. For example, if we have some Nat-algebra Γ, the recursor yields a morphism from the initial algebra to Γ. We can map Γ to a list-algebra or a Fin-algebra, and then we can also use recursors for lists or Fin. Equivalently, we can map the unique morphism to Γ directly to a morphism between ornamented algebras.

Note though that a number of features and concepts from prior work on ornaments are not yet reproduced. For example, we do not yet have an analogue of *algebraic ornaments*, which would allow us produce an ornamented signature as an *output* of a generic operation, instead of assuming it to begin with. Exploring ornaments with QII signatures could be part of future work.

Model constructions

In a broader context, ToS provides a synthetic language for specifying *model constructions*.

Example 19. For a simple example, we might want to show that constant families are equivalent to democracy in cwfs. Democracy means that for each $\Gamma : \mathsf{Con}$ there is a $\overline{\Gamma} : \mathsf{Ty}\,\bullet$ such that $\Gamma \simeq (\bullet \triangleright \overline{\Gamma})$ [CCD17, Section 3.1].

We can define a $\sigma : \mathsf{Sub}\,\mathsf{cwf}^{\mathsf{K}}\,\mathsf{cwf}^{\mathsf{dem}}$ which interprets democracy using constant families. It is the identity morphism on the cwf parts and interprets democracy as $\overline{\Gamma} :\equiv \mathsf{K}\,\Gamma$. The isomorphism $\Gamma \simeq (\bullet \triangleright \mathsf{K}\,\Gamma)$ follows from the specification of K. We can also define a morphism $\sigma^{-1} : \mathsf{Sub}\,\mathsf{cwf}^{\mathsf{K}}\,\mathsf{cwf}^{\mathsf{dem}}$, which interprets $\mathsf{K}\,\Delta$ as $\overline{\Delta}[\epsilon]$. It is easy to check that σ^{-1} is indeed the inverse of σ. Thus we get an isomorphism of categories of models from the ToS semantics.

This construction is very simple, and would not be difficult to check without the ToS semantics. But it is generally not obvious that a certain mapping from models to models extends to an flcwf morphism, so it may be helpful to work inside ToS.

Example 20. There is a simple way to show that if a type theory does not support η for Π, then function extensionality is not provable in the theory [BPT17][1].

[1]It is also possible to show unprovability of function extensionality *assuming* η for functions, but in significantly more complicated ways. To the author's best knowledge, the set-based polynomial model is the easiest solution [VG15].

Assume some type theory with Σ, Π, Id and Bool, and abbreviate its signature as TT. We define a $\sigma : \mathsf{Sub}\,\mathsf{TT}\,\mathsf{TT}$ which has identity action everywhere except on Π. There, we have

$$\begin{aligned}\Pi &:\equiv \lambda\, A\, B.\, \Pi\, A\, B \times \mathsf{Bool}\\ \mathsf{app} &:\equiv \lambda\, t.\, \mathsf{app}\,(\mathsf{proj}_1\, t)\\ \mathsf{lam} &:\equiv \lambda\, t.\, (\mathsf{lam}\, t,\, \mathsf{true})\end{aligned}$$

In short, we tag functions with a Bool value. This equips Π with "intensional" information, contradicting extensionality. If we have two functions which are pointwise equal, that only specifies that the function parts are equal, but does not say anything about the Bool tags. Hence, if we take any model of TT, we get a new model by the semantic action of σ, where function extensionality is false. Note though that the η rule also fails in the new model, so we had to drop η from the TT signature as well.

In [BPT17], this construction is presented for the special case where the starting model is initial. While it is easy to generalize to arbitrary starting models, it is less obvious to extend the construction to a functor of categories of models - which we do get for free here.

Example 21. The gluing construction by Kaposi, Huber and Sattler [KHS19] takes as input two models of some type theory together with a weak cwf-morphism between them, and produces as output a displayed model over the first model. Depending on the choice of the inputs, the gluing construction can yield parametricity translations and canonicity proofs as well.

Let us use $\mathsf{TT} : \mathsf{Ty}\,\bullet$ for the signature of the type theory, given as an iterated large Σ-type. Then, the notion of weak cwf-morphism is also expressible in ToS as $\mathsf{morph} : \mathsf{Ty}\,(\bullet \triangleright (M_0 : \mathsf{TT}) \triangleright (M_1 : \mathsf{TT}))$, and the notion of displayed model as well, as $\mathsf{TT}^D : \mathsf{Ty}\,(\bullet \triangleright (M : \mathsf{TT}))$[2]. Thus, we can give a "type" for the gluing construction, as follows:

$$\begin{aligned}&\mathsf{Tm}\,(\bullet \triangleright (M_0 : \mathsf{TT}) \triangleright (M_1 : \mathsf{TT}) \triangleright (f : \mathsf{morph}[M_0 \mapsto M_0,\, M_1 \mapsto M_1]))\\ &\quad (\mathsf{TT}^D[M \mapsto M_0])\end{aligned}$$

Moreover, the gluing construction itself can be given as an inhabitant of the above type. This construction works in the ToS, because it only reuses structure from M_1 to define the displayed model over M_0.

[2]We will be also able to automatically derive TT^D from TT, in Section 5.4.

Limitations. In the finitary ToS syntax, when defining substitutions we can only ever use assumed type constructors. If we assume a Σ and $\top$ type formers in the domain signature of a construction, we might be able to work around the lack of Σ and $\top$ in U in the ToS itself. This does not always work though; for example, take the substitution with type $\mathsf{Sub}\,\mathsf{MonoidSig}\,\mathsf{CatSig}$ which maps a monoid to a single-object category. Assuming $\mathsf{M} : \mathsf{U}$ is the carrier set in $\mathsf{MonoidSig}$, we would need to have the following:

$$\begin{aligned}\mathsf{Obj} &:\equiv \top\\ \mathsf{Hom} &:\equiv \lambda \_\,\_.\,\mathsf{M}\end{aligned}$$

But we have $\mathsf{Obj} : \mathsf{U}$ in CatSig, so we would need to have $\top : \mathsf{U}$. In Chapter 5, we present a more expressive ToS which does include $\top : \mathsf{U}$.

4.4.4 Using Signatures in Implementations

We may ask whether the current ToS is suitable for implementations of type theories. The answer is not wholly straightforward.

Note that we must choose a concrete surface syntax in an implementation, and there are many design choices. The surface syntax would be almost certainly nameful, and may or may not leave El-s implicit, since they are not difficult to insert by bidirectional elaboration. Besides the elaboration of surface syntax, we should have at least the computation of induction principles.

Equality reflection in the ToS is a complication. If we have "silent" transports along equality reflection, that makes elaboration of surface signatures undecidable. We might make transports explicit, which restores decidable checking, but that requires the ToS to be deeply embedded in some ambient theory[3].

Alternatively, we may just drop equality reflection from the ToS, and use transport and UIP as primitives. This recovers decidable surface syntax, but now we have to cover transport and UIP in the semantics, to be able to compute induction principles. This is not too difficult; in Chapter 6 we do the same for path induction J in the ToS. In that case, we even have a Haskell implementation of signature elaboration and computation of induction principles [Kov20].

Hence, handling signatures and computing induction principles is not difficult. Instead, the real gap between our ToS and practical implementations is that we

[3]Equality reflection is simply an equality constructor in the embedded syntax, and has no bearing on decidability of type checking in the metalanguage.

need to have computationally adequate treatment of quotients. In plain Martin-Löf type theories, computation gets stuck on quotients. We need to use more recent systems, such as a cubical type theories [VMA21, SAG20], or some flavor of observational [AMS07] or setoid [ABKT19] type theory. In each of these systems, the signatures and their semantics would need to be adapted, and we would need to work out additional details. For example, we would need to produce extra computation rules which explain the behavior of coercion or transport on QIIT constructors.

4.4.5 Recovering AMDS Interpretations

We have defined the M model in a "bundled" fashion, but sometimes we will also need to refer to pieces of it. On Figure 4.1 we have a summary of the model. On the left, the rows are labeled with components of ToS, while on the top we have components of flcwf. The individual rows can be further unfolded, as each of them contains multiple components. Likewise the Σ, Id and K columns can be unfolded. We get the whole model by filling every cell of the unfolded table with a definition. Of course, many of these cells are equations between equations, hence trivial by UIP.

This setup is very regular and convenient, because we can extract a displayed ToS model from any column, which may depend on columns to the left. The whole model is the total model of all columns. For example, the Con column does not depend on anything, so it is a plain model. The Ty column is displayed over Con. The Tm column depends on Con and Ty, but it does not depend on Sub.

See also Appendix A for a tabular specification of the AMDS interpretations.

From each displayed model, we get an eliminator, i.e. a family of interpretation functions. We note $-^A$, $-^M$, $-^D$ and $-^S$ in the table, but in principle we could refer to the eliminators of other columns as well. The interpretation functions can be defined in two ways:

- By separately taking the eliminators of each column, and referring to previous eliminators in each displayed model; e.g. referring to the eliminator functions $-^A$ in the definition of the Ty column.
- By taking the recursor for the entire model, and projecting out components from the result. E.g. we get $-^A$ by projecting out the first components of the interpretations of ToS objects.

	cwf					fl		
	Con	Sub	Ty	Tm	...	Σ	Id	K
cwf U Id Π Π^{ext}	$\_^A$	$\_^M$	$\_^D$	$\_^S$				

Figure 4.1: The flcwf model of the theory of signatures

However, the two versions coincide because of the initiality of ToS syntax.

4.5 Term Algebras

In this section we proceed with the construction of term algebras for FQII signatures, together with their recursors and eliminators. We make two significant modifications to the setup.

First, **we drop the outer theory**, and work exclusively inside an extensional type theory. The reason is the following. The main purpose of 2LTT is to generalize the semantics of signatures. In the previous section, we presented semantics for signatures, where algebras are internal to arbitrary cwfs with Σ, $\top$ and extensional Id. This is quite general; in particular we can interpret signatures in any finitely complete category. We also described dropping assumptions in Section 4.4.2, thereby getting semantics in yet more general settings.

In contrast, we make a lot more assumptions in the inner theory when we develop initial term algebras; we essentially have to replicate the outer features verbatim. Thus, we gain nothing by using 2LTT, compared to working in a model of an extensional TT.

What about the term model construction for simple signatures in Section 3.5.6, why did we use 2LTT there? In that case, the inner theory was intensional, i.e. lacked equality reflection. So there remained an interesting distinction between the inner and outer layer, which allowed us to prove definitional β-rules for recursors. In contrast, here we assume inner equality reflection, so we have no distinction between propositional and definitional inner equality.

$$\frac{i \leq j}{\Gamma \vdash \mathsf{Set}_i \leq \mathsf{Set}_j} \qquad \frac{\Gamma,\, x : A \vdash B \leq B'}{\Gamma \vdash (x : A) \to B \leq (x : A) \to B'}$$

$$\frac{\Gamma \vdash A \leq A' \qquad \Gamma,\, x : A \vdash B \leq B'}{\Gamma \vdash (x : A) \times B \leq (x : A') \times B'} \qquad \frac{}{\Gamma \vdash A \leq A}$$

$$\frac{\Gamma \vdash A \leq B \qquad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C} \qquad \frac{\Gamma \vdash A \leq A' \qquad \Gamma \vdash t : A}{\Gamma \vdash t : A'}$$

Figure 4.2: Rules for cumulative subtyping

Second, **we make universe levels explicit** in the semantics and constructions. So far, we have been consistently ignoring universe levels. Now, size questions are less obvious, and quite relevant to a) ensuring the consistency of assumed induction principles b) laying groundwork for bootstrapped semantics and self-describing signatures in Section 4.6.

Universe levels are a fairly bureaucratic detail in type theories. In the following we try to be as informal as possible, while still representing the essential sizing aspects. In the following, we describe the new universe setup, and adapt the previously described signatures and semantics to it.

4.5.1 Universes & Metatheory

We have countable Russell-style Set_i universes, which are *cumulative*, meaning that any type in Set_i is also an element of Set_{i+1}. We use a surface syntax which is similar to Coq, where cumulativity is implicit. This contrasts the formal ("algebraic") specification of cumulativity [Ste19, Kov21], which involves rather heavy explicit annotation.

Also following Coq, we have implicit *cumulative subtyping* [TS18]. In our case, this means that cumulativity distributes through basic type formers. We have a $- \leq -$ subtyping relation on types, specified on Figure 4.2. This is subtyping for *surface syntax*; it is expected that surface syntax can be elaborated to *coercions* in a formal syntax with algebraic cumulativity.

Note that we have an invariant rule for function domain types. This is to

match Coq and [TS18], and also because we will not need a contravariant rule in any case.

We assume that Π and Σ types return in least upper bounds of levels. For instance, assuming $A : \mathsf{Set}_i$ and $B : A \to \mathsf{Set}_j$, we have $(x : A) \to B : \mathsf{Set}_{i \sqcup j}$.

4.5.2 Signatures & Semantics

First, we parameterize the notion of ToS-model with levels.

Definition 47. For levels i and j, $\mathsf{ToS}_{i,j} : \mathsf{Set}_{i+1 \sqcup j+1}$ is the type of ToS models, defined as before, but where Con, Sub, Ty and Tm all return in Set_i, and Π^{ext} abstracts over Set_j.

We have that $\mathsf{ToS}_{i,j} \leq \mathsf{ToS}_{i+1,j}$. This follows from the rules in Figure 4.2. All underlying sets return in Set_i, which can be bumped to Set_{i+1}. Th j level does not change, which is as expected, since Set_j appears in a negative position in the type of Π^{ext}, and has to be invariant.

Assumption. We assume that for all j, there exists $\mathsf{syn}_j : \mathsf{ToS}_{j+1,j}$ which supports induction. Note the level bump in the first index; this is to avoid inconsistency from type-in-type:

$$\begin{aligned} \mathsf{Ty} &: \mathsf{Con} \to \mathsf{Set}_{j+1} \\ \Pi^{\mathsf{ext}} &: (A : \mathsf{Set}_j) \to (A \to \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,\Gamma \end{aligned}$$

With Ty returning in Set_j, Π^{ext} would "contain" a Set_j, but at the same time return in a type in Set_j, and by induction we would be able to derive a Russell-like paradox. Likewise, all other underlying sets must be bumped to Set_{j+1}, because of their mutual nature: contexts, terms and substitutions all "contain" types through some of their constructors.

Definition 48 (Signatures). We define $\mathsf{Sig}_j : \mathsf{Set}_{j+1}$ as the type of signatures where Π^{ext} may abstract over Set_j, so we have $\mathsf{Sig}_j :\equiv \mathsf{Con}_{\mathsf{syn}_j}$.

Definition 49 (Flwcf model). For levels i and j, we have $\mathbf{M}_{i,j} : \mathsf{ToS}_{(i+1 \sqcup j)+1,j}$ as the model where contexts are flcwfs, and objects in the flcwf are algebras. The model is defined in essentially the same way as in Section 4.2. The algebras have underlying sets in Set_i and (semantic) external functions have types in Set_j as domain. Hence, every algebra in $\mathbf{M}_{i,j}$ is in $\mathsf{Set}_{i+1 \sqcup j}$.

Example 22. We may define NatSig as an element of Sig_0. Then, by interpreting the signature in $\mathbf{M}_{i,0}$, we get $\mathsf{NatSig}^A \equiv (N : \mathsf{Set}_i) \times (N \to N) \times N$, hence $\mathsf{NatSig}^A : \mathsf{Set}_{i+1 \sqcup 0}$.

Notation 18. For a signature $\Gamma : \mathsf{Sig}_j$ and level i, we may write Γ_i^A for the type of Γ-algebras with underlying sets in Set_i, which is computed by interpreting Γ in $\mathbf{M}_{i,j}$. We may use similar notation for $-^M$, $-^D$ and $-^S$.

Cumulativity of algebras. In the following, we shall assume that for $\Gamma : \mathsf{Sig}_j$ and $i \leq i'$, we have $\Gamma_i^A \leq \Gamma_{i'}^A$. For any concrete signature Γ, this is clearly the case, but $- \leq -$ is not subject to propositional reasoning, so we cannot prove this by internal induction on signatures. We can prove by induction on signatures that there exists a *lifting*, a $\mathsf{Lift}\,\Gamma_i^A : \mathsf{Set}_{i+1 \sqcup j}$ which is isomorphic to Γ_i^A. Instead, we take liberties, and work as if we had actual cumulative subtyping. This seems acceptable, since by using implicit cumulativity, we are already taking the same liberty everywhere, by omitting formal lifts and isomorphisms.

4.5.3 Term Algebra Construction

We fix $\Omega : \mathsf{Sig}_j$ for some j level. We define $-^T$ by induction on syn_j. In the following we write $-^A$ for $-^A_{j+1}$, i.e. the algebra interpretation where underlying sets are in Set_{j+1}. Formally, we need a displayed model over syn_j, but we instead present the resulting eliminator, which is perhaps easier to read. The underlying functions have the following types.

$$
\begin{aligned}
&-^T : (\Gamma : \mathsf{Con}) && (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^A \\
&-^T : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) && (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^T\,(\sigma \circ \nu) \equiv \sigma^A\,(\Gamma^T\,\nu) \\
&-^T : (A : \mathsf{Ty}\,\Gamma) && (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,(A[\nu]) \to A^A\,(\Gamma^T\,\nu) \\
&-^T : (t : \mathsf{Tm}\,\Gamma\,A) && (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^T\,\nu\,(A[\nu]) \equiv t^A\,(\Gamma^T\,\nu)
\end{aligned}
$$

We review the idea of term algebras. In any model of ToS, we might think of a $\mathsf{Sub}\,\bullet\,\Gamma$ as a Γ-algebra internal to the model. In the $-^T$ interpretation we can assume $\Omega \equiv \bullet$; this means that from any internal Γ-algebra we can extract an "external" Γ-algebra. This is possible because every sort $a : \mathsf{Tm}\,\Gamma\,\mathsf{U}$ in ToS induces an external type of terms as $\mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)$.

We can view the generalization from $\bullet$ to arbitrary Ω as switching from working in the syntactic model syn_j, to working in the *slice model* syn_j/Ω, where contexts

are given as Ω extended with zero or more entries. And in syn_j/Ω, we have an Ω-algebra quite trivially, by taking the identity morphism $\mathsf{id} : \mathsf{Sub}\,\Omega\,\Omega$[4]. Hence, term algebras arise by first taking the trivial internal algebra id in syn_j/Ω, then converting it to an external algebra as $\Omega^T\,\mathsf{id} : \Omega^A$.

Remark. We could have presented $-^T$ and slice models separately. We instead chose to merge them into the current $-^T$, since we do not use slice models elsewhere, and we can skip their definition this way. Slice models would require the specification of *telescopes*, used to extend the base context, but this entails a fair amount of bureaucratic detail.

We explain the $-^T$ specification in the following. Term and substitution equations are given by UIP. We omit cases for substitutions and terms.

For contexts, we simply recurse on the entries. We use a pattern matching notation for $\mathsf{Sub}\,\Omega\,(\Gamma \triangleright A)$, since any ν with this type is uniquely determined by its first and second projections $\mathsf{p} \circ \nu$ and $\mathsf{q}[\nu]$.

$$\begin{aligned} \bullet^T\,\nu &:\equiv \mathsf{tt} \\ (\Gamma \triangleright A)^T(\nu,\, t) &:\equiv (\Gamma^T\,\nu,\, A^T\,\nu\,t) \end{aligned}$$

Type substitution with $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$ is as follows. This is well-typed by $\sigma^T\,\nu : \Delta^T\,(\sigma \circ \nu) \equiv \sigma^A\,(\Gamma^T\,\nu)$.

$$(A[\sigma])^T\,\nu\,t :\equiv A^T\,(\sigma \circ \nu)\,t$$

Universe

For the universe, note that $\mathsf{U}^A_{j+1}\,\gamma \equiv \mathsf{Set}_{j+1}$. As we mentioned before, this is the key part when we map from internal sorts to external sets. The levels line up, since in syn_j we have Tm returning in Set_{j+1}.

$$\begin{aligned} &\mathsf{U}^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,\mathsf{U} \to \mathsf{Ty} \\ &\mathsf{U}^T\,\nu\,a :\equiv \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a) \end{aligned}$$

For El, we have to define

$$(\mathsf{El}\,a)^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu])) \to a^A\,(\Gamma^T\,\nu)$$

[4]Writing $-^{\mathsf{syn}_j/\Omega}$ for the interpretation of syntax in the slice model, $\mathsf{Sub}_{\mathsf{syn}_j/\Omega}\bullet(\Omega^{\mathsf{syn}_j/\Omega})$ is isomorphic to, but not strictly the same as $\mathsf{Sub}_{\mathsf{syn}_j}\,\Omega\,\Omega$.

but since $a^T\,\nu : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu])) \equiv a^A\,(\Gamma^T\,\nu)$, we have

$$(\mathsf{El}\,a)^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu])) \to \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu]))$$
$$(\mathsf{El}\,a)^T\,\nu\,t :\equiv t$$

The $a^T\,\nu$ equation is worth noting. If we have $\nu \equiv \mathsf{id}$, the equation is $a^T\,\mathsf{id} : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a) \equiv a^A\,(\Omega^T\,\mathsf{id})$, that is, if we evaluate a signature sort in the term model $\Omega^T\,\mathsf{id}$, we get a type of inner terms.

Identity

We have to show that provably equal terms are evaluated to the same value in the term model.

$$(\mathsf{Id}\,t\,u)^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,(\mathsf{Id}\,(t[\nu])\,(u[\nu])) \to t^A\,(\Gamma^T\,\nu) \equiv u^A\,(\Gamma^T\,\nu)$$

We know by equality reflection that $t[\nu] \equiv u[\nu]$, and we also get

$$t^T\,\nu\ : A^T\,\nu\,(t[\nu])\ \equiv t^A\,(\Gamma^T\,\nu)$$
$$u^T\,\nu : A^T\,\nu\,(u[\nu]) \equiv u^A\,(\Gamma^T\,\nu)$$

from which the target equality follows. Equality reflection for inner Id is crucial here. It is the reason why $-^T$ works for *quotient signatures*; equality reflection is in fact the "quotient" rule which identifies provably equal terms. For a simple example, terms with type

$$\mathsf{Tm}\,(\bullet \triangleright (I : \mathsf{U}) \triangleright (\mathit{left} : \mathsf{El}\,I) \triangleright (\mathit{right} : \mathsf{El}\,I) \triangleright (\mathit{seg} : \mathsf{Id}\,l\,r))\,(\mathsf{El}\,I)$$

are quotiented by *seg*, which is a provable equation in the context.

Inductive function type

Here we have to convert an inner term with Π type to an outer function.

$$(\Pi\,a\,B)^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,(\Pi\,(a[\nu])\,(B[\nu \circ \mathsf{p},\ \mathsf{q}]))$$
$$\to (\alpha : a^A\,(\Gamma^T\,\nu)) \to B^A\,(\Gamma^T\,\nu,\ \alpha)$$
$$(\Pi\,a\,B)^T\,\nu\,t :\equiv \lambda\,\alpha.\,B^T\,(\nu,\ \alpha)\,(t\,\alpha)$$

This is well-typed by $a^T\,\nu : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu])) \equiv a^A\,(\Gamma^T\,\nu)$, which allows us to consider α to be an inner term in $\lambda\,\alpha.\,B^T\,(\nu,\ \alpha)\,(t\,\alpha)$.

External function type

In this case we just recurse through the specifying isomorphism:

$$\begin{aligned}(\Pi^{\mathsf{ext}}\, A\, B)^T &: (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,(\Pi^{\mathsf{ext}}\, A\,(\lambda\,\alpha.\,(B\,\alpha)[\nu])) \\ &\qquad \to (\alpha : A) \to (B\,\alpha)^A\,(\Gamma^T\,\nu) \\ (\Pi^{\mathsf{ext}}\, A\, B)^T\,\nu\,t &:\equiv \lambda\,\alpha.\,(B\,\alpha)^T\,(\nu,\,\alpha)\end{aligned}$$

This concludes the definition of $-^T$.

Definition 50. For an $\Omega : \mathsf{Con}_{\mathsf{syn}_j}$ signature, the corresponding **term algebra** is given as $\Omega^T\,\mathsf{id} : \Omega^A_{j+1}$.

Remark. If we start with a signature in syn_j, then the underlying sets in the term algebra are all in Set_{j+1}. Hence, the term algebra for $\mathsf{NatSig} : \mathsf{Sig}_0$ has an underlying set in Set_1. This is perhaps inconvenient, since normally we would have natural numbers in Set_0. However, we argue that this is no issue, because we are free to specify Set_0 as we like. In particular, we can say that Set_0 is an *empty universe*, closed under no type formers at all (or explicitly isomorphic to $\bot$) in which case Sig_0 stands for *closed* signatures (since Π^{ext} cannot be constructed), and it is expected that any closed inductive type would be placed in Set_1. Alternatively, we could name the bottom-most universe Set_{empty} or Set_{-1}, and start counting non-empty universes from Set_0.

4.5.4 Recursor Construction

We continue with the construction of recursors. This is not necessary, strictly speaking, since recursion is derivable from elimination, so it would suffice to only construct eliminators. We still present recursors, for the sake of matching the presentation in Chapter 2.

The goal is to construct a morphism from a term algebra to any other $\omega : \Omega^A$ algebra. However, we have to handle universe levels as well. We want to be able to eliminate from the term algebra, which was constructed at the lowest possible level, to any other universe. So far we have not introduced a "heterogeneous" notion of morphism, between algebras at different levels. We get this from cumulativity.

- We assume $\Omega : \mathsf{Sig}_j$, for which we already have the term algebra $\Omega^T\,\mathsf{id} : \Omega^A_{j+1}$.
- We assume some $k \geq j+1$, and an $\omega : \Omega^A_k$, the target of recursion.

- We implicitly lift $\Omega^T\,\mathsf{id}$ from level $j+1$ to level k by cumulativity, and construct a "homogeneous" morphism from the lifted term algebra to ω.

This allows us to eliminate from $\Omega^T\,\mathsf{id}$ to any level. If we want to eliminate to $k \geq j+1$, we can lift the term algebra, and use a constructed recursor. On the other hand, if we want to eliminate to $k < j+1$, we can instead lift the target $\omega : \Omega_k^A$ algebra to $j+1$, and again use a constructed recursor.

In general, for any $\omega : \Omega_i^A$ and $\omega' : \Omega_j^A$, the notion of heterogeneous morphism between them arises by lifting both algebras to $i \sqcup j$, and taking homogeneous morphisms between these.

Example 23. The $\mathsf{NatSig} : \mathsf{Sig}_0$ signature gives rise to $\mathsf{NatSig}^T\,\mathsf{id} : \mathsf{NatSig}_1^A$. This consist of $\mathsf{Nat} : \mathsf{Set}_1$ together with zero and suc. Assuming a recursion principle as described above, and $\mathsf{Bool} : \mathsf{Set}_0$, we may define an $\mathsf{isZero} : \mathsf{Nat} \to \mathsf{Bool}$ function by "downwards" elimination. We have that $(\mathsf{Bool}, \mathsf{true}, \lambda\,\_.\,\mathsf{false}) : \mathsf{NatSig}_0^A$, so by cumulativity we also have $(\mathsf{Bool}, \mathsf{true}, \lambda\,\_.\,\mathsf{false}) : \mathsf{NatSig}_1^A$, hence by recursion we get the desired morphism from $\mathsf{NatSig}^T\,\mathsf{id}$ to this model, which contains the $\mathsf{Nat} \to \mathsf{Bool}$ function. We can also eliminate "upwards" by lifting $\mathsf{NatSig}^T\,\mathsf{id}$ to any NatSig_i^A for $i > 1$.

We define $-^R$ by induction on syn_j. From this, we will obtain the recursor as $\Omega^R\,\mathsf{id}$.

$$
\begin{aligned}
&-^R : (\Gamma : \mathsf{Con}) \qquad (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^M\,(\nu^A\,(\Omega^T\,\mathsf{id}))\,(\nu^A\,\omega)\\
&-^R : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta)(\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^R\,(\sigma \circ \nu) \equiv \sigma^M\,(\Gamma^R\,\nu)\\
&-^R : (A : \mathsf{Ty}\,\Gamma) \qquad (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(A[\nu])) \to A^M\,(t^A\,(\Omega^T\,\mathsf{id}))\,(t^A\,\omega)\,(\Gamma^R\,\nu)\\
&-^R : (t : \mathsf{Tm}\,\Gamma\,A)\ (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^R\,\nu\,(t[\nu]) \equiv t^A\,(\Gamma^R\,\nu)
\end{aligned}
$$

Let us refresh some details about the involved operations. The reader may also refer to Appendix A for definitions of the AMDS interpretations.

- For $\nu : \mathsf{Sub}\,\Gamma\,\Delta$, we get $\nu^A : \Gamma^A \to \Delta^A$. In the semantics, ν is a functor, and ν^A is its action on objects. Analogously, for a term $t : \mathsf{Tm}\,\Gamma\,A$, we have $t^A : (\gamma : \Gamma^A) \to A^A\,\gamma$, also an action on objects.

- Γ^M is the set of Γ-morphisms. $A : \mathsf{Ty}\,\Gamma$ is a displayed flcwf in the semantics. A^M yields sets of displayed morphisms, corresponding to the semantic Sub component. So we have

$$A^M : A^A\,\gamma_0 \to A^A\,\gamma_1 \to \Gamma^M\,\gamma_0\,\gamma_1 \to \mathsf{Set}_k$$

- t^M and σ^M yield actions on morphisms. For $t : \mathsf{Tm}\,\Gamma\,A$ and $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$, we have

$$\begin{aligned} t^M &: (\gamma^M : \Gamma^M\,\gamma_0\,\gamma_1) \to A^M\,(t^A\,\gamma_0)\,(t^A\,\gamma_1)\,\gamma^M \\ \sigma^M &: (\gamma^M : \Gamma^M\,\gamma_0\,\gamma_1) \to \Delta^M\,(\sigma^A\,\gamma_0)\,(\sigma^A\,\gamma_1) \end{aligned}$$

Again, we follow the "sliced" pattern that we have seen in the term model construction. Another way to view this, is that getting term algebras or recursors by direct induction on signatures is futile, since in the construction we have to refer to the *whole* Ω signature, but when we recurse inside Ω we necessarily get *smaller* signatures.

Hence, the sliced induction can be viewed as induction on arbitrary Γ signatures which are smaller than Ω, in the sense that there is a $\mathsf{Sub}\,\Omega\,\Gamma$. Of course, $\mathsf{Sub}\,\Omega\,\Gamma$ includes "being smaller", but it is more general.

We look at the interpretation of type formers. Again, term and substitution equations are given by UIP, and we omit term and substitution formers. For contexts, we again simply recurse:

$$\begin{aligned} \bullet^R\,\nu &:\equiv \mathsf{tt} \\ (\Gamma \triangleright A)^R(\nu,\, t) &:\equiv (\Gamma^R\,\nu,\, A^R\,\nu\,t) \end{aligned}$$

Type substitutions with $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$ also follows the same pattern. The following is well-typed by $\sigma^R\,\nu : \Delta^R\,(\sigma \circ \nu) \equiv \sigma^M\,(\Gamma^R\,\nu)$.

$$(A[\sigma])^R\,\nu\,t :\equiv A^R\,(\sigma \circ \nu)\,t$$

Universe

We need to define

$$\mathsf{U}^R : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(a : \mathsf{Tm}\,\Omega\,\mathsf{U}) \to \mathsf{U}^M\,(a^A\,(\Omega^T\,\mathsf{id}))\,(a^A\,\omega)\,(\Gamma^R\,\nu)$$

Morphisms in the semantics of U are simply functions. Moreover, we have $a^T\,\mathsf{id} : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a) \equiv a^A\,(\Omega^T\,\mathsf{id})$.

$$\begin{aligned} &\mathsf{U}^R : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(a : \mathsf{Tm}\,\Omega\,\mathsf{U}) \to \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a) \to a^A\,\omega \\ &\mathsf{U}^R\,\nu\,a\,t :\equiv t^A\,\omega \end{aligned}$$

Thus, we evaluate t in the ω algebra, the same way as we did in Chapter 2.

For El, we need to show

$$(\mathsf{El}\, a)^R : (\nu : \mathsf{Sub}\, \Omega\, \Gamma)(t : \mathsf{Tm}\, \Gamma\, (\mathsf{El}\, (a[\nu]))) \to a^M\, (\Gamma^R\, \nu)\, (t^A\, (\Omega^T\, \mathsf{id})) \equiv t^A\, \omega$$

We have $a^R\, \nu : U^R\, \nu\, (a[\nu]) \equiv a^M\, (\Gamma^R\, \nu)$. Hence, $U^R\, \nu\, (a[\nu])\, t \equiv a^M\, (\Gamma^R\, \nu)\, t$, and by computing U^R we have $t^A\, \omega \equiv a^M\, (\Gamma^R\, \nu)\, t$. The target equation then follows by $t^T\, \mathsf{id} : t^A\, (\Omega^T\, \mathsf{id}) \equiv t$.

Identity

We need to show:

$$(\mathsf{Id}\, t\, u)^R : (\nu : \mathsf{Sub}\, \Omega\, \Gamma)(e : \mathsf{Tm}\, \Gamma\, (\mathsf{Id}\, (t[\nu])\, (u[\nu]))) \to t^M\, (\Gamma^R\, \nu) \equiv u^M\, (\Gamma^R\, \nu)$$

This follows from equality reflection on e, together with

$$\begin{aligned} t^R\, \nu &: A^R\, \nu\, (t[\nu]) \equiv t^M\, (\Gamma^R\, \nu) \\ u^R\, \nu &: A^R\, \nu\, (u[\nu]) \equiv u^M\, (\Gamma^R\, \nu) \end{aligned}$$

Inductive function type

We get the following target type after unfolding $(\Pi\, a\, B)^M$:

$$\begin{aligned} &(\Pi\, a\, B)^R : (\nu : \mathsf{Sub}\, \Omega\, \Gamma)(t : \mathsf{Tm}\, \Omega\, (\Pi\, (a[\nu])\, (B[\nu \circ \mathsf{p},\, \mathsf{q}]))) \\ &\quad \to (\alpha : a^A\, (\nu^A\, (\Omega^T\, \mathsf{id}))) \to B^M\, (t^A\, (\Omega^T\, \mathsf{id})\, \alpha)\, (t^A\, \omega\, (a^M\, (\Gamma^R\, \nu)\, \alpha))\, (\Gamma^R\, \nu,\, \mathsf{refl}) \end{aligned}$$

We have

$$\begin{aligned} \nu^T\, \mathsf{id} &: \Gamma^T\, \nu \equiv \nu^A\, (\Omega^T\, \mathsf{id}) \\ a^T\, \nu &: a^A\, (\Gamma^T\, \mathsf{id}) \equiv \mathsf{Tm}\, \Omega\, (\mathsf{El}\, (a[\nu])) \end{aligned}$$

Hence, $a^A\, (\nu^A\, (\Omega^T\, \mathsf{id})) \equiv \mathsf{Tm}\, \Omega\, (\mathsf{El}\, (a[\nu]))$. We also have $a^R\, \nu : (\lambda\, \alpha.\, \alpha^A\, \omega) \equiv a^M\, (\Gamma^R\, \nu)$, therefore $\alpha^A\, \omega \equiv a^M\, (\Gamma^R\, \nu)$. With this in mind, the goal type can be rewritten as

$$\begin{aligned} &(\Pi\, a\, B)^R : (\nu : \mathsf{Sub}\, \Omega\, \Gamma)(t : \mathsf{Tm}\, \Omega\, (\Pi\, (a[\nu])\, (B[\nu \circ \mathsf{p},\, \mathsf{q}]))) \\ &\quad \to (\alpha : \mathsf{Tm}\, \Omega\, (\mathsf{El}\, (a[\nu]))) \to B^M\, (t^A\, (\Omega^T\, \mathsf{id})\, \alpha)\, (t^A\, \omega\, (\alpha^A\, \omega))\, (\Gamma^R\, \nu,\, \mathsf{refl}) \end{aligned}$$

We have the following typing now:

$$B^R\,(\nu,\,\alpha)\,(t\,\alpha) : B^M\,((t\,\alpha)^A\,(\Omega^T\,\mathsf{id}))\,((t\,\alpha)^A\,\omega)\,(\Gamma^T\,\nu,\,\mathsf{refl})$$

By the action of $-^A$ on inductive function application, we have

$$B^R\,(\nu,\,\alpha)\,(t\,\alpha) : B^M\,(t^A\,(\Omega^T\,\mathsf{id})\,(\alpha^A\,(\Omega^T\,\mathsf{id})))\,(t^A\,\omega\,(\alpha^A\,\omega))\,(\Gamma^T\,\nu,\,\mathsf{refl})$$

But since $\alpha^T\,\mathsf{id} : \alpha^A\,(\Omega^T\,\mathsf{id}) \equiv \alpha$, this is exactly the target type. Therefore the definition is:

$$(\Pi\,a\,B)^R\,\nu\,t :\equiv \lambda\,\alpha.\,B^R\,(\nu,\,\alpha)\,(t\,\alpha)$$

External function type

We again simply recurse through the indexing:

$$\begin{aligned}(\Pi^{\mathsf{ext}}\,A\,B)^R : {}&(\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(\Pi^{\mathsf{ext}}\,A\,(\lambda\,\alpha.\,(B\,\alpha)[\nu])))\\ &\to (\alpha : A) \to (B\,\alpha)^M\,(t^A\,(\Omega^T\,\mathsf{id})\,\alpha)\,(t^A\,\omega\,\alpha)\,(\Gamma^R\,\nu)\\ (\Pi^{\mathsf{ext}}\,A\,B)^R\,\nu\,t :{}&\equiv \lambda\,\alpha.\,(B\,\alpha)^R\,\nu\,(t\,\alpha)\end{aligned}$$

This concludes the definition of $-^R$.

Definition 51 (Recursors). Assuming $\Omega : \mathsf{Sig}_j$, a k level such that $k \geq j+1$ and $\omega : \Omega^A_k$, we have $\Omega^R\,\mathsf{id} : \Omega^M\,(\Omega^T\,\mathsf{id})\,\omega$ as the recursor for the term algebra.

4.5.5 Eliminator Construction

We assume $\Omega : \mathsf{Sig}_j$ and $\omega^D : \Omega^D_k\,(\Omega^T\,\mathsf{id})$, where $k \geq j+1$. Again we implicitly lift the term algebra from level $j+1$ to k. Here, ω^D is a displayed algebra over the term algebra. We seek to construct an inhabitant of $\Omega^S\,(\Omega^T\,\mathsf{id})\,\omega^D$. We define $-^E$ by induction.

Constructing eliminators is on the whole quite similar to the recursor construction. The switch from morphisms to sections is mechanical. We shall only look at U, El and Π here.

$$\begin{aligned}
&-^E : (\Gamma : \mathsf{Con}) && (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^S\,(\nu^A\,(\Omega^T\,\mathsf{id}))\,(\nu^D\,\omega^D)\\
&-^E : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) && (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^E\,(\sigma \circ \nu) \equiv \sigma^S\,(\Gamma^E\,\nu)\\
&-^E : (A : \mathsf{Ty}\,\Gamma) && (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(A[\nu])) \to A^S\,(t^A\,(\Omega^T\,\mathsf{id}))\,(t^D\,\omega^D)\,(\Gamma^E\,\nu)\\
&-^E : (t : \mathsf{Tm}\,\Gamma\,A) && (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^E\,\nu\,(t[\nu]) \equiv t^S\,(\Gamma^E\,\nu)
\end{aligned}$$

For the **universe**, we have the following.

$$\mathsf{U}^E : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(a : \mathsf{Tm}\,\Omega\,\mathsf{U}) \to (\alpha : a^A\,(\Omega^T\,\mathsf{id})) \to a^D\,\omega^D\,\alpha$$

By $a^T\,\mathsf{id} : a^A\,(\Omega^T\,\mathsf{id}) \equiv \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a)$, we can give the following definition:

$$\begin{aligned}&\mathsf{U}^E : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(a : \mathsf{Tm}\,\Omega\,\mathsf{U}) \to (\alpha : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a)) \to a^D\,\omega^D\,\alpha\\&\mathsf{U}^E\,\nu\,a\,\alpha :\equiv \alpha^D\,\omega^D\end{aligned}$$

In other words, we evaluate α in the ω^D displayed algebra. Let us check that this is well-typed:

$$\begin{aligned}\alpha^D &: \{\omega : \Omega^A\}(\omega^D : \Omega^D\,\omega) \to a^D\,\omega^D\,(\alpha^A\,\omega)\\\alpha^D\,\omega^D &: a^D\,\omega^D\,(\alpha^A\,(\Omega^T\,\mathsf{id})\\\alpha^T\,\mathsf{id} &: \alpha^A\,(\Omega^T\,\mathsf{id}) \equiv \alpha\end{aligned}$$

Thus $\alpha^D\,\omega^D : a^D\,\omega^D\,\alpha$. Recall that α^D can be viewed as the logical predicate interpretation of α, which expresses that α^A preserves $-^D$ predicates.

For El, we need to show

$$(\mathsf{El}\,a)^S : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(a[\nu]))) \to a^S\,(\Gamma^E\,\nu)\,(t^A\,(\Omega^T\,\mathsf{id})) \equiv t^D\,\omega^D$$

This follows from $t^T\,\mathsf{id} : t^A\,(\Omega^T\,\mathsf{id}) \equiv t$ and $a^E\,\nu : (\lambda\,t.\,t^D\,\omega^D) \equiv a^S\,(\Gamma^E\,\nu)$.

The **inductive function** interpretation is defined similarly as before:

$$\begin{aligned}&(\Pi\,a\,B)^E : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(\Pi\,(a[\nu])\,(B[\nu \circ \mathsf{p}, \mathsf{q}])))\\&\qquad \to (\alpha : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu]))) \to B^S\,(t^A\,(\Omega^T\,\mathsf{id})\,\alpha)\,(t^D\,\omega^D\,(\alpha^D\,\omega^D))\,(\Gamma^E\,\nu, \mathsf{refl})\\&(\Pi\,a\,B)^E\,\nu\,t :\equiv \lambda\,\alpha.\,B^E\,(\nu, \alpha)\,(t\,\alpha)\end{aligned}$$

We make use of $\nu^T\,\mathsf{id}$, $u^T\,\mathsf{id}$, $a^E\,\nu$ and $a^T\,\nu$ to type-check the definition.

Interpretations for contexts and other type formers are also essentially the same as with recursors.

Definition 52 (**Eliminators**)**.** Assuming $\Omega : \mathsf{Sig}_j$, a k level such that $k \geq j+1$ and $\omega^D : \Omega^D_k\,(\Omega^T\,\mathsf{id})$, we have $\Omega^E\,\mathsf{id} : \Omega^S\,(\Omega^T\,\mathsf{id})\,\omega^D$ as the eliminator.

Theorem 5. *$\Omega^T\,\mathsf{id}$ is initial when lifted to any $k \geq j+1$ level.*

Proof. $\Omega^T\,\mathsf{id} : \Omega^A_k$ supports elimination by Definition 52, and elimination is equivalent to initiality by Theorem 1. □

4.6 Levitation and Bootstrapping for Closed Signatures

When we previously introduced the ToS, we only specified the notion of model, and simply assumed that there is an evident notion of model morphism and also a notion of induction. For the theory of *closed* signatures, we can do better, because that ToS is itself a closed FQII theory. This is *levitation* [CDMM10], i.e. the situation where a ToS contains a signature for itself. Levitation is useful for bootstrapping: it shall be sufficient to specify only the notion of model for ToS, and notions of ToS-morphisms, initiality and induction can be computed from that. This bootstrapping process eliminates the need for either

- Assuming that the syntax of ToS already exists as a QIIT. This is not too elegant, since we are in the process of building metatheory for QII theories.
- Bootstrapping the ToS syntax as "raw" syntax, using simple inductive types, typing/conversion relations and quotients. This is very tedious and should be avoided if possible. See Section 4.7 for a discussion of this approach, although used for slightly different purposes.

In this section we describe levitation for closed signatures. The theory of closed signatures does not have Π^{ext}, but is otherwise the same as before. As we have seen, the inclusion of Π^{ext} yields a ToS which is itself infinitary, which breaks levitation. Moving to a theory of infinitary signatures will restore levitation; we revisit this is Section 5.7.

4.6.1 Models & Signatures

Since we do not have Π^{ext}, we only need a single universe level for indexing models.

Definition 53. For some i level, we have $\mathsf{ToS}_i : \mathsf{Set}_{i+1}$ as the type of models of ToS, where all underlying sets return in Set_i.

Definition 54 (Flcwf model). For i, we have $\mathbf{M}_i : \mathsf{ToS}_{i+2}$ as the model where contexts are flcwfs of algebras, and algebras have underlying sets in Set_i. To see how $i+2$ checks out: if algebras contain Set_i-s, the category of algebras has a Set_{i+1} for a set of objects, and $\mathbf{M}_i$ itself includes a category of these categories.

So far, this can be defined while only using the notion of model for ToS. What about signatures though? Previously we had that signatures are contexts in ToS *syntax*, and to talk about syntax, we need to know at least the notion of ToS model morphism.

Actually, if we only want to be able to write down signatures and interpret them in the semantics, we do not need a ToS syntax. A functional encoding suffices.

Definition 55. A **bootstrap signature** is a function which for every ToS model yields a context in that model. The type of bootstrap signatures is:

$$\mathsf{BootSig} :\equiv (i : \mathsf{Level}) \to (M : \mathsf{ToS}_i) \to \mathsf{Con}_M$$

Note that this is a universe-polymorphic type. This is not an issue; universe polymorphism is a sensible feature in type theories, or alternatively we may assume that quantification over levels takes place in some outer theory.

We do not get induction on bootstrap signatures, nor do we automatically get any naturality or parametricity property.

Example 24. For NatSig, we define the expected signature, but we specify it in an arbitrary M model instead of the syntax.

$$\mathsf{NatSig} : \mathsf{BootSig}$$
$$\mathsf{NatSig} :\equiv \lambda(i : \mathsf{Level})(M : \mathsf{ToS}_i).$$
$$(\bullet_M \triangleright_M (N : \mathsf{U}_M) \triangleright_M (\mathit{zero} : \mathsf{El}_M\, N) \triangleright_M (\mathit{suc} : N \Rightarrow_M \mathsf{El}_M\, N))$$

We might as well use the same notations for signatures as in Section 4.1, as every signature from before can be unambiguously rewritten as a bootstrap signature.

With this, we can interpret each signature in an arbitrary ToS model, by applying a signature to a model. $\mathsf{BootSig}_j$ can be viewed as a precursor to a Böhm-Berarducci encoding [BB85] for the theory of signatures, but we only need contexts encoded in this way, and not other ToS components. In functional programming, this style of encoding is sometimes called "finally tagless" [CKS07].

If we only want to build the 2LTT-based semantics of signatures, we are done with bootstrapping right now. In the 2LTT semantics, we never needed induction on ToS, we only needed to be able to write down signatures and interpret them in models - which we can do. Going forward, we only need to assume an inner

(Ty_0, Tm_0) layer with appropriate type formers, and define the flcwf model the same way as before.

On the other hand, if we want to consider term models, we do need a notion of induction on ToS.

Definition 56 (**Signature for ToS**)**.** We define $\mathsf{ToSSig} : \mathsf{BootSig}$ as the bootstrap signature for the theory of signatures. We present an excerpt from ToSSig below using internal notation; it should be clear that every component can be reproduced. We use SigU and SigEl to disambiguate components inside the signature from ToS components.

$$
\begin{aligned}
&\mathsf{Con} && : \mathsf{U}\\
&\mathsf{Sub} && : \mathsf{Con} \to \mathsf{Con} \to \mathsf{U}\\
&\mathsf{Ty} && : \mathsf{Con} \to \mathsf{U}\\
&\mathsf{Tm} && : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{U}\\
&\ldots\\
&\mathsf{SigU} && : \{\Gamma : \mathsf{Con}\} \to \mathsf{El}\,(\mathsf{Ty}\,\Gamma)\\
&\mathsf{SigEl} && : \{\Gamma : \mathsf{Con}\} \to \mathsf{Tm}\,\Gamma\,\mathsf{SigU} \to \mathsf{El}\,(\mathsf{Ty}\,\Gamma)\\
&\Pi && : \{\Gamma : \mathsf{Con}\}(a : \mathsf{Tm}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright \mathsf{SigEl}\,a) \to \mathsf{El}\,(\mathsf{Ty}\,\Gamma)\\
&\ldots
\end{aligned}
$$

For each i, the interpretation of ToSSig in $\mathbf{M}_i$ yields an flcwf $\boldsymbol{\Gamma}$ such that $\mathsf{Con}_{\boldsymbol{\Gamma}} \equiv \mathsf{ToS}_i$, that is, objects are models of ToS at level i. This yields a model theory for ToS, which includes the notion of induction at level i.

We also know by the definition of ToS_i that we have cumulativity, i.e. $\mathsf{ToS}_i \leq \mathsf{ToS}_{i+1}$[5]. Hence, we can make the following definition:

Definition 57. $M : \mathsf{ToS}_0$ supports elimination into any universe if it supports elimination when lifted by cumulativity to any ToS_i.

This notion of (large) elimination is sufficient for the term algebra and eliminator constructions in Section 4.5. Thus, we were able to derive all required concepts just from the notion of model of ToS.

[5]For concrete bootstrap signatures we may conclude cumulativity of algebras, but we cannot conclude this universally for all bootstrap signatures, since we cannot do induction on them, and we do not even assume that they are parametric in levels.

4.7 Reductions to Basic Type Formers

From the construction of term algebras and eliminators, we get a reduction of all QIITs to a single infinitary QIIT, namely the syntax of ToS. We spell this out:

Theorem 6. *If an extensional type theory supports syntax for* $\mathsf{ToS}_{j+1,j}$*, it supports initial algebras for each signature in* Sig_j*.*

Ideally, we would like to reduce QIITs to some collection of basic type formers. The ToS syntax is far from being a basic type former, it is rather large and complicated. Therefore, the remaining job is to reduce the ToS syntax to basic type formers.

We do not attempt here to construct the entire ToS syntax as specified. The reason is the following: [LS, Section 9] shows that infinitary QIITs are not constructible from inductive types and simple quotienting with relations. We do not actually know what is the basic type former, the magic ingredient from which infinitary QIITs are actually constructible, and which is simple enough. Simplicity is key, of course; in Chapter 5 we construct infinitary QIITs from the infinitary ToS syntax, but that is certainly not a simple structure.

In [AK16, Section 2.2], there is an argument that infinitary quotient inductive types extend the base type theory with constructive choice principles. We conjecture that the magic ingredient would be an infinitary QIT which expresses some kind of generic choice principle. We leave this to future work.

What we can do is to show constructions of certain fragments of the full ToS syntax. In the following, we first give a general description of QIIT constructions, then describe two specific constructions, for a) finitary inductive-inductive signatures b) closed QII signatures.

4.7.1 Finitary QIIT Constructions

The general recipe of constructing finitary QIITs from basic type formers is the following. This is more or less adapted from Streicher [Str93] and Brunerie et al. [Bru19].

1. We define the *raw* syntax, using at most inductive families, but no induction-induction. These definitions include all value constructors of the goal QIIT, but there is no indexing involved, constructors only store the raw inductive

data. For example, the raw syntax of closed ToS would include the following:

$$
\begin{aligned}
&\mathsf{Con} && : \mathsf{Set}\\
&\mathsf{Sub} && : \mathsf{Set}\\
&\mathsf{Ty} && : \mathsf{Set}\\
&\mathsf{Tm} && : \mathsf{Set}\\
&\bullet && : \mathsf{Con}\\
&- \triangleright - && : \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Ty}\\
&\mathsf{id} && : \mathsf{Con} \to \mathsf{Sub}\\
&- \circ - && : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Con} \to \mathsf{Sub} \to \mathsf{Sub} \to \mathsf{Sub}\\
&\ldots
\end{aligned}
$$

This can be given by a simple mutual inductive definition, which can be represented as an indexed inductive family. Indexed families can be reduced to indexed W-types [KvR20], which can be reduced in turn to W-types and the identity type.

2. We define typing and conversion relations on the raw syntax. For dependent type theories, the two are usually mutual: typing includes the rule which coerces terms along type conversion, and conversion is usually defined only on well-typed terms. However, it is still possible to define everything using only indexed inductive families.

3. The underlying sets are given as follows: we take raw syntactic objects which are *merely* well-formed (i.e. proofs of well-formedness are propositionally truncated, or defined in a universe of irrelevant propositions to begin with), and quotient them by conversion.

4. We show that these underlying sets support all constructors of the target QIIT: value constructors are defined using raw constructors, while equality constructors follow from conversion rules and quotienting.

5. We construct a unique morphism from the above term model to an arbitrary model of the QII theory. This usually requires several steps. One approach is to first define by induction on raw syntax a family of partial functions into the assumed model, then separately show that these functions are total on well-typed input. The separation is necessary because the induction principle

for the raw syntax is too weak: it cannot express the inductive-inductive indexing dependencies which would be required to construct the morphism in one go. For instance, if we have the QIIT syntax for ToS, and we have some displayed model A over the syntax, the eliminator contains the following:

$$\mathsf{Con}^S : (\Gamma : \mathsf{Con}) \to \mathsf{Con}_A\,\Gamma$$
$$\mathsf{Sub}^S : (\Gamma\,\Delta : \mathsf{Con})(\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Sub}_A\,(\mathsf{Con}^S\,\Gamma)\,(\mathsf{Con}^S\,\Delta)\,\sigma$$

But with the raw syntax, we can only eliminate using a displayed model of the raw syntax, and the eliminator contains the following:

$$\mathsf{Con}^S : (\Gamma : \mathsf{Con}) \to \mathsf{Con}_A\,\Gamma$$
$$\mathsf{Sub}^S : (\sigma : \mathsf{Sub}) \to \mathsf{Sub}_A\,\sigma$$

Lastly, we show that the constructed morphism is unique. This is done by induction on raw syntax, and is generally possible in just one elimination.

Note that the above recipe permits a large number of design variations. Some examples:

- We may omit fields from raw syntax which are fully determined by type indices. This may make subsequent work easier or harder depending on particulars.

- We may start from a *well-scoped* raw syntax, if there is a notion of scoping in the goal QIIT. In general, we may start from some kind of partially raw syntax, which is well-typed to some extent. This extent is bounded by what is expressible only using indexed inductive families but not induction-induction.

- We may move along a spectrum of "paranoia" in the specification of well-typing [Win20, Section 9.2]. A paranoid typing rule assumes the well-formedness of everything involved, for example assumes the well-formedness of a context Γ before it assumes well-formedness of a type in Γ. In contrast, an "economic" specification tries to make the minimum necessary assumptions, relying on admissibility properties. It is possible that well-formedness of Γ is derivable from the well-formedness of a type in Γ, so the assumption can be dropped.

  However, if we omit too much, then some other admissibility properties may break! Design decisions along the paranoia spectrum are often all tangled up

like this; hence the name "paranoid", which probably stems from the anxiety of breaking things by making too much shortcuts.

- Instead of using partial maps from raw syntax to the the assumed model in step 5, we may define *relations* between *well-formed* raw syntax and the given model, and later show that these relations are functional. This seems to be a technically easier approach. The reason is that we do not have decidable definedness of the partial maps, which makes them more complicated. A decidably defined partial function has type $A \to \mathsf{Maybe}\, B$. For any $a : A$ we can look at whether the function is defined on it. A more general partial function has type $A \to ((P : \mathsf{Prop}) \times (P \to B))$. If we forget about the Prop-ness of P for the time being, we can equivalently have a relation $A \to B \to \mathsf{Set}$ instead. This is a more "indexed" definition compared to the "fibered" presentation with $P : \mathsf{Prop}$, and indexed presentations in type theory usually enjoy more definitional computation rules - this is also the reason why displayed algebras are better-behaved computationally than fibered algebras.

It should be apparent that constructing QIITs is tedious, and especially so for large QIITs like type theories. Hence, it is best if we do it just once, for a theory of signatures from which every other QIIT can be constructed.

Connection to the initiality conjecture

The initiality conjecture was made by Voevodsky [Voe17], and it is essentially the conjecture that the above construction ("initiality construction") can be carried out in sufficient formal detail for "usual" type theories.

There has been much debate about the merits of initiality constructions. See [Con19] for a hub of such discussions. On one hand, some people believed that the initiality construction is essential for reconciling the usage of raw syntax and categorical notions of models. On the other hand, some people dismissed the initiality construction as a pointless exercise, considering the categorical syntax to be the actual syntax, and raw syntax as merely notation for that. The author of this thesis is of a somewhat different opinion than either of the above.

First, as a moral justification for the usage of raw syntax, the initiality construction is indeed mostly pointless. That is because *elaboration* comprises the true justification for that. Elaboration is the effective algorithm which converts

raw syntax to "core syntax", i.e. typed categorical syntax. Given a piece of raw syntax, even if I have done the initiality construction, I have no effective way of learning which core syntactic object it corresponds to! The elaboration literature is mainly about practical justifications for using certain raw syntaxes, and it comes with established ways to talk about strength and correctness of elaboration algorithms.

Second, there is a different motivation for the initiality construction: *foundational minimalism*, the reduction of a complicated QIIT to basic type formers. Elaboration merely assumes that a categorical core syntax already exists, as the target of elaboration, but it is orthogonal to the construction of the core syntax. If we have elaboration, we may still want to show a reduction of the core syntax, but now we are free to perform this construction in whatever way is the easiest. We do not have to construct the QIIT out of a raw syntax which is intentionally close to the raw syntax that we use in practice! In the author's opinion, a great deal of confusion arises from the conflation of the two different motivations for the initiality construction.

As to which way of construction is easiest: the author does not know of any truly easy way, but this thesis shows that we only have to do it once, for a theory of signatures, and then we can construct all other QIITs from that in a generic way. In particular, almost all type theories in the wild are finitary closed QII theories (with the notable exceptions of our ToS-es), so if we can construct closed signatures, we can construct initial models of almost all type theories.

What about generic ways to formalize elaboration algorithms? This seems to be a lot more difficult. To the author's knowledge there have not been notable research in this area. Decidability of conversion is already very hard to analyze in a generic way, and the simplest possible bidirectional elaboration algorithms rely on decidable conversion. To formalize practically realistic elaboration (i.e. elaboration which includes unification) is yet more difficult.

4.7.2 Reduction of Finitary Inductive-Inductive Types

This section is based on the author's joint work with Kaposi and Lafont [KKL19]. The core idea is the following: a certain fragment of ToS can be constructed in a far simpler way than what we described in Section 4.7.1, with fewer assumptions in the ambient theory. We call this fragment the theory of *finitary inductive-inductive* signatures. This theory has the following type formers (on the top of

the base cwf):

- The U universe with El.
- Inductive function type Π, but without lam, and thus without $\beta\eta$-rules.
- External function type Π^{ext}, but again without $\mathsf{lam}^{\mathsf{ext}}$.

This ToS is tuned so that

1. No quotients are required in its construction.
2. The generic term model construction still goes through for every signature in the ToS.

We explain in the following. First, the equational theory of ToS only specifies substitution, but it contains no computation rules for type formers. Thus, ToS is a theory of *neutral terms* and substitutions. This allows us to define a raw syntax which only includes normal forms, and to define substitution as recursive functions acting on normal forms. This trivializes the conversion relation: conversion is simply propositional equality of raw terms. Thus, there is no need to quotient by conversion. Note that our raw syntax is infinitary, because we have to represent the branching in Π^{inf}. This is fine though: we only run into the issue of the missing "choice" principle if we try to mix quotients and infinite branching. Without quotients, infinite branching is not an issue.

Second, we do not include an identity type in ToS. This blocks the other way for quotients to enter the picture. With identity types, the generic term model construction relies on equality reflection in ToS. But when we construct ToS syntax, the only way to show equality reflection is to quotient raw syntax by internally provable equalities.

Third, it remains to check that the generic term model construction works with the pared-down ToS. We only need to check that the omission of lam and $\mathsf{lam}^{\mathsf{ext}}$ does not mess things up. Looking at Sections 4.5.3 and 4.5.5, we see that it does not: the interpretations of Π and Π^{ext} only require applications in ToS, not abstractions.

Remark. Although we have not yet talked about infinitary signatures, we can give a short summary why the current construction fails to work for their ToS. The generic term algebra construction in Section 5.6.1 for infinitary signatures relies on there being both lam and app for "infinitary" function types, with $\beta\eta$-rules. This

makes the equational theory of ToS non-trivial, so quotients are necessary in the construction of the syntax. However, this requires mixing quotients and infinite branching, which we cannot yet handle.

We summarize the construction of the ToS syntax below. We refer the reader to [KKL19] for details.

1. We define raw syntax by mutual induction. Substitution are in normal form: they are just lists of raw terms. Variables are also normalized as de Bruijn indices. We define the action of substitution by recursion on raw syntax. In [KKL19], raw syntax is not well-scoped, and substitution is partial, but it would be also possible to start from well-scoped raw syntax.

2. We inductively define well-formedness relations for contexts, substitutions, types and terms, and show by induction on raw syntax that well-formedness is propositional, i.e. proof-irrelevant. Alternatively, we could have defined well-formedness by recursion on raw syntax.

3. We construct a term model of ToS from well-formed raw syntax. All equations in the model are provable from the properties of recursive substitution on raw terms.

4. We pick a ToS model, and inductively define a family of relations between the term model and the given model, which define the function graphs of the model morphism that we aim to define. Then we show in order:

   (a) Right-uniqueness of the relation, by induction on well-formedness derivations.

   (b) Stability of the relation under substitution.

   (c) Left-totality of the relation, by induction on well-formedness derivations.

   We then define the actual model morphism using the functionality of the relation.

5. For the uniqueness of the constructed morphism, we exploit right-uniqueness of the relation: it is enough to show that any other model morphism maps syntactic input to related semantic output.

This construction is formalized in Agda; see [KKL19]. It uses indexed inductive families, UIP, function extensionality, and equality reflection in the form of Agda rewrite rules, although the latter could be in principle omitted from the formalization. Thus, it follows that any model of ETT with inductive families supports finitary inductive-inductive types.

4.7.3 Reduction of Closed QIITs

For closed QIITs, there is unfortunately no direct formalization which constructs the ToS. There is one though which is *close enough*, by Menno de Boer and Guillaume Brunerie [BdB20], see also De Boer's thesis [dB20]. This constructs a type theory with the following features:

- A contextual category for base (instead of a cwf).
- Countable predicative universes.
- $\mathbb{N}$, Σ, Π, $\top$, $\bot$, $-+-$ and intensional Id.

The construction follows the 1-5 steps that we described previously in Section 4.7.1. It makes the following assumptions:

- A universe of strict propositions Prop. Every type in this universe enjoys definitional proof-irrelevance. This Prop is used to define partial functions and well-formedness relations.
- Function extensionality.
- Propositional extensionality for Prop.
- Quotients by relations valued in Prop.
- Indexed inductive families returning in Set or in Prop.

Notably, UIP is not assumed. It appears that the irrelevance of equalities in Prop in sufficient to obviate UIP.

It is very plausible that this construction can be adapted to our theory of closed QII signatures. De Boer and Brunerie construct a complicated open finitary QIIT, while ours is a fairly similar closed QIIT, with fewer and more restricted type formers. The openness comes from the use of contextual categories, which involve indexing by external natural numbers. Contextuality does not make much

difference in the construction though, since raw syntax is always contextual by the inductive nature of raw contexts.

Hence, it is safe to say that any model of a type theory which supports the assumptions of De Boer and Brunerie, also supports all closed QIITs.

4.8 Related Work

This chapter is based on the following publications, all coauthored by the current thesis' author.

1. "Constructing Quotient Inductive-Inductive Types" [KKA19]
2. "Large and Infinitary Quotient Inductive-Inductive Types" [KK20b]
3. "For Finitary Induction-Induction, Induction is Enough" [KKL19]

We summarize the differences and enhancements in this chapter, in comparison to the above (1)-(3) sources.

The theory of signatures is similar to that in (1), except (1) does not include eliminators for Π and Π^{ext}, and it has $\mathsf{Id} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma$, i.e. it cannot equate terms with arbitrary types.

The usage of 2LTT is novel compared to (1)-(3). In (1), the semantics had a cwf with Id and K for each signature; this was extended with Σ in (2) to get the notion of flcwf that we also use in this chapter.

The construction of left adjoints of substitutions is novel.

The current term algebra construction is the same as in (1), but universe levels were not treated rigorously in (1); instead we adapt the more precise universe treatment from (2). Notions of bootstrapping and levitation are also "backported" from (2) to closed finitary signatures.

(3) is summarized in the current chapter without any notable change.

Generalized Algebraic Theories

FQII signatures and Cartmell's generalized algebraic theories [Car86] are close in expressive power, but they do not appear to be equivalent.

GATs may contain an infinite number of rules, while FQII signatures are finitely long. On the other hand, FQII signatures have Π^{ext} and GATs do not. It appears that infinite signatures are stronger than Π^{ext}: it is possible to recover Π^{ext} by

adding a rule for every value of the external index, but it is not possible to recover infinite signatures with Π^{ext}. The reason is that in $\Pi^{\mathsf{ext}} : (Ix : \mathsf{Ty}_0) \to (Ix \to \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,\Gamma$, the Γ context is fixed, so it is not possible to represent a family of signature entries where each entry may refer to the previous entry within the same family. For example, the following (pseudo)-GAT has no corresponding FQII signature:

$$
\begin{aligned}
&A_0 : \mathsf{U}\\
&A_1 : A_0 \to \mathsf{U}\\
&A_2 : (a_0 : A_0) \to A_1\, a_1 \to \mathsf{U}\\
&A_3 : (a_0 : A_0)(a_1 : A_1\, a_0) \to A_2\, a_0\, a_1 \to \mathsf{U}\\
&\ldots
\end{aligned}
$$

Could we somehow include these? The most convenient way would be to define signatures coinductively. However, the would cause a mismatch, that described theories are inductive, while the ToS itself is coinductive, which rules out levitation and bootstrapping. It is potential future work to investigate such coinductive signatures.

This leads us to the main difference in formalization between GATs and FQII signatures: the theory of GATs itself is not presented as a GAT, instead it has a low-level presentation with raw syntax and well-formedness relations. As a result, the immediate metatheory of GATs is roughly as tedious as we can expect from raw syntaxes.

This is a motivation for formally getting away from GATs, by showing their equivalence to contextual categories. Contextual categories are algebraic and more convenient to handle than GATs. In [Car78] one leg of this equivalence is the construction of a classifying contextual category for each GAT, which is essentially a term model construction from quotiented raw syntax. A downside of this setup is that classifying contextual categories cannot be easily written out by hand like GATs. Thus, GATs necessarily remain the practical way for specifying the classifying categories.

In contrast, the theory of FQII signatures is itself algebraic, possesses a nice model theory (as an infinitary QII theory), and it is only mildly more complex than the theory of contextual categories. Since the immediate theory of signatures is already quite nice, we do not feel as much pressure to look for nicer presentations.

Nevertheless, compact alternative presentations would be still interesting to research.

- We could look for for an analogue of the GAT-contextual-category correspondence for our signatures. This would send each signature to its classifying category.
- We could also look for an analogue of the Gabriel-Ulmer duality [GU06]. This would send each signature to its category of models in **Set**. In the other direction, we would need a way to restrict categories to those which are categories of algebras.

Essentially algebraic theories

Essentially algebraic theories (EATs) [Fre72] are categories with finite limits. This is a more semantic notion of an algebraic signature, much like how contextual categories are a more semantic presentation of "syntactic" GATs. For EATs Γ and Δ, the Γ-algebras internal to Δ are simply the finite limit preserving functors from Γ to Δ, while algebra morphisms are natural transformations.

We have more syntactic notions of essentially algebraic signatures as well. For example, the signatures of Adámek and Rosicky [AAR$^+$94, Section 3.D] or the Partial Horn theories of Palmgren and Vickers [PV07] are such. These signatures are also specified using raw syntax, but they are significantly easier to formalize than GATs, as the syntax of signatures admits fewer dependencies. However, the lack of dependency also causes a significant encoding overhead on comparison to GATs or FQII signatures. For a classic example, the theory of transitive directed graphs is given with an FQII signature as

$$\begin{aligned}
&\mathsf{V} && : \mathsf{U} \\
&\mathsf{E} && : \mathsf{V} \to \mathsf{V} \to \mathsf{U} \\
&- \circ - && : (i\, j\, k : \mathsf{V}) \to \mathsf{E}\, i\, j \to \mathsf{E}\, j\, k \to \mathsf{El}\,(\mathsf{E}\, i\, k)
\end{aligned}$$

The same in a pseudo-EA notation could be:

$$\begin{aligned}
&\mathsf{V} && : \mathsf{Set} \\
&\mathsf{E} && : \mathsf{Set} \\
&\mathsf{src} && : \mathsf{E} \to \mathsf{V} \\
&\mathsf{tgt} && : \mathsf{E} \to \mathsf{V}
\end{aligned}$$

$$- \circ - : (f\ g : \mathsf{E}) \to \mathsf{tgt}\, f = \mathsf{src}\, g \to (h : \mathsf{E}) \times (\mathsf{src}\, h = \mathsf{src}\, f) \times (\mathsf{tgt}\, h = \mathsf{tgt}\, g)$$

In short, the FQII notation is "indexed", while the EA is "fibered". Also recall Theorem 3. While this example is not wildly different in the two cases, if we move to more complex theories, such as type theories, the encoding overhead of EA signatures is much greater. In informal mathematics, this is still not an issue, but in mechanized mathematics, it is. Type dependencies are a formal complication, but in proof assistants they enable more compact definitions. They also often force indices to be particular values, which enables inference and unification to fill in more details in surface syntaxes.

Sketches (see e.g. [Bar85, Section 4]) are another way to specify EATs. They lie somewhere between the syntactic/logical styles of specification, and just taking EATs to be finitely complete categories. They support an elegant metatheory, but they involve an encoding overhead which is likely unworkable in mechanized settings.

All in all, there is a rich literature on EATs, sketches and related topics, and it would be interesting to try to connect our signatures to any of these, or try to reproduce the numerous related results in categorical universal algebra. This remains future work for now.

Prior work on (quotient) inductive types

The current work grew out of a line of research in the field of type theory. This involved working out more and more expressive classes of inductive types.

Martin-Löf's W-types [ML84] are an early example for a scheme for inductive types. In fact, it is better viewed as a single parameterized inductive type, which allows construction of a remarkable range of inductive types [Hug21], although with some encoding overheads.

Inductive families [Dyb94] allow indexing the inductive sort with external types. This directly supports only single-sorted signatures, but some form of mutual induction can be easily modeled through the indexing. Inductive families have become a core feature in all major implementations of type theories, such as Coq, Idris, Lean or Agda.

Inductive-recursive types [DS99] allow mutual definition of an inductive sort and a function which acts on the sort. These types are absent from this thesis, they are not representable with any of our theories of signatures. Induction-recursion

is notable for tremendously boosting the proof-theoretic strength of a type theory; a primary motivation for it was to explore the limits of predicative constructive mathematics. It is useful for modeling a wide variety of universe features internally to a type theory [Kov21].

Induction-induction was described in [AMFS11] and in [NF13]. This notion allowed two inductive sorts, where the second one may be indexed over the first. As we mentioned previously, this notion is more restricted than what was covered in this chapter.

[ACD+18] investigated QIITs. The notion of signature here is more of a semantic nature than ours. Signatures are defined simultaneously with their categories of algebras. A signature is a inductive list of functors: at each signature entry, we extend the category of algebras with a functor whose domain is the current category of algebras. This can be viewed as a generalization of F-algebras as a form of specification. However, there is no strict positivity restriction in signatures, hence no attempt at constructing initial algebras either.

We will look at work related to infinitary QITs in Section 5.8 and at work related to higher inductive types in Section 6.3.2.

CHAPTER 5

Infinitary Quotient Inductive-Inductive Signatures

In this chapter we present another theory of signatures, for *infinitary quotient inductive-inductive* signatures. As we will see, the reason for considering the finitary and infinitary cases separately is that they support different semantics.

First, we specify signatures and define semantics in 2LTT. Then, like in the previous chapter, we switch to a extensional TT setting and look at term algebras and related constructions.

5.1 Theory of Signatures

Metatheory. We work in 2LTT. We assume the following type formers in the inner theory: $\top$, Σ, extensional identity $- = -$ and Π. Note that Π is an extra assumption compared to what we had in the finitary case.

Definition 58. A **model of the theory of signatures** consists of the following.

- A **cwf** with underlying sets Con, Sub, Ty and Tm, all returning in the outer Set universe of 2LTT.
- A **Tarski-style universe** U with decoding El. U is closed under the following type formers:
  - The **unit type** $\top$.
  - **Σ-types** $\Sigma : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,\mathsf{U} \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$, with specifying isomorphism

$$(\mathsf{proj},\ -,-) : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Sigma\,a\,b)) \simeq (t : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)) \times \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b[\mathsf{id},\ t]))$$

  - **Extensional identity** $\mathsf{Id} : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a) \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$, specified by $(\mathsf{reflect}, \mathsf{refl}) : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\mathsf{Id}\,t\,u)) \simeq (t \equiv u)$.
  - **Infinitary functions** $\Pi^{\mathsf{inf}} : (Ix : \mathsf{Ty}_0) \to (Ix \to \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$, specified by $(\mathsf{app}^{\mathsf{inf}}, \mathsf{lam}^{\mathsf{inf}}) : \mathsf{Tm}\,\Gamma\,(\Pi^{\mathsf{inf}}\,Ix\,b) \simeq ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b\,i)))$.

- An **inductive function type** $\Pi : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Ty}\,(\Gamma \triangleright \mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma$, specified by $(\mathsf{app}, \mathsf{lam}) : \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) \simeq \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B$.
- An **external function type** $\Pi^{\mathsf{ext}} : (Ix : \mathsf{Ty}_0) \to (Ix \to \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,\Gamma$, specified by $(\mathsf{app}^{\mathsf{ext}}, \mathsf{lam}^{\mathsf{ext}}) : \mathsf{Tm}\,\Gamma\,(\Pi^{\mathsf{ext}}\,Ix\,B) \simeq ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(B\,i))$.

Once again we assume that an initial model for ToS exists, and a signature is a context in the initial model.

Notation 19. We employ the same notations for signatures as in Section 4.1. In addition to that, we have the usual internal notation for $\top$ and Σ, and we write $(x : A) \to^{\mathsf{inf}} B$ for Π^{inf} and λ^{inf} for $\mathsf{lam}^{\mathsf{inf}}$.

Let us do a comparison to the finitary case. First, the new signatures do not support sort equations, since there is no identity type for arbitrary terms, only for terms with types in U. Second, the universe is not empty anymore, it supports $\top$, Σ and the infinitary function type Π^{inf}, which can be viewed as an analogue of Π^{ext} inside U. We look at example signatures.

Example 25. Infinitary constructors can be given with Π^{inf}. A classic example is W-types. Assuming $S : \mathsf{Ty}_0$ and $P : S \to \mathsf{Ty}_0$, we have the following signature for P-branching well-founded trees:

$$\begin{aligned} &W &&: \mathsf{U} \\ &\mathsf{sup} &&: (s : S) \to^{\mathsf{ext}} (P\,s \to^{\mathsf{inf}} W) \to \mathsf{El}\,W \end{aligned}$$

Note that since $P\,s \to^{\mathsf{inf}} W$ is in U, it can appear on the left side of $\to$. Hence the naming "infinitary": if $P\,s$ is an infinite type, sup branches with an infinite number of inductive subtrees. Of course, finitary branching can be also expressed with Π^{inf}, but that use case was already possible with finitary signatures, by iterating $\to$ finite times.

Example 26. Equations can appear as assumptions now. The simplest example

would be set truncation for some $A : \mathsf{Ty}_0$:

$$\begin{aligned} |A|_0 \quad &: \mathsf{U} \\ \mathsf{embed} &: A \to^{\mathsf{ext}} \mathsf{El}\,|A|_0 \\ \mathsf{trunc} \quad &: (x\,y : |A|_0)(p\,q : \mathsf{Id}\,x\,y) \to \mathsf{El}\,(\mathsf{Id}\,p\,q) \end{aligned}$$

However, this ends up being redundant in our semantics, since we assume UIP, and every semantic underlying type will be a set. Does this mean that recursive equations are useless? We do not think so. In the specification of cubical type theories, there are *boundary conditions* which can be given as Id assumptions [CCHM17, AHW16, AHH18]. Also, it seems that these conditions cannot be easily contracted away, or expressed equivalently without recursive equation. For an example of contraction, the signature

$$\bullet \triangleright (A : \mathsf{U}) \triangleright (c_1 : \mathsf{El}\,A) \triangleright (c_2 : (x : A) \to \mathsf{Id}\,x\,c_1 \to \mathsf{El}\,A)$$

can be rewritten to the equivalent

$$\bullet \triangleright (A : \mathsf{U}) \triangleright (c_1 : \mathsf{El}\,A) \triangleright (c_2 : \mathsf{El}\,A)$$

signature. However, we cannot mechanically eliminate the Id from the following signature.

$$\begin{aligned} \mathsf{A} \quad &: \mathsf{U} \\ \mathsf{B} \quad &: \mathsf{A} \to \mathsf{U} \\ \mathsf{b}_1 &: \mathsf{A} \to \mathsf{El}\,\mathsf{B} \\ \mathsf{b}_2 &: \mathsf{A} \to \mathsf{El}\,\mathsf{B} \\ &\ldots \\ \mathsf{a} \quad &: (x\,y : \mathsf{A}) \to \mathsf{Id}\,(\mathsf{b}_1\,x)\,(\mathsf{b}_2\,y) \to \mathsf{El}\,\mathsf{A} \end{aligned}$$

Whether we can reformulate a without the Id condition depends on what kind of equational theory we specify for B in the omitted parts of the signature.

Example 27. All theories of signatures that we discussed so far, have (infinitary) signatures.

For finitary signatures, the ToS is itself infinitary because of Π^{ext}. We assume

an universe U_0 in Ty_0. In the signature, we have

$$\begin{aligned}
&\mathsf{Con} : \mathsf{U}\\
&\mathsf{Ty} \quad : \mathsf{Con} \to \mathsf{U}\\
&\Pi^{\mathsf{ext}} : \{\Gamma : \mathsf{Con}\} \to (A : \mathsf{U}_0) \to^{\mathsf{ext}} (A \to^{\mathsf{inf}} \mathsf{Ty}\,\Gamma) \to \mathsf{El}\,(\mathsf{Ty}\,\Gamma)
\end{aligned}$$

In the signature for infinitary ToS, we have

$$\begin{aligned}
&\mathsf{Univ} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Ty}\,\Gamma\\
&\Pi^{\mathsf{inf}} \ : \{\Gamma : \mathsf{Con}\} \to (A : \mathsf{U}_0) \to^{\mathsf{ext}} (A \to^{\mathsf{inf}} \mathsf{Tm}\,\Gamma\,\mathsf{Univ}) \to \mathsf{Tm}\,\Gamma\,\mathsf{Univ}
\end{aligned}$$

Remark. When we will take the semantics of the above signature, we will not exactly get back the theory of signatures that we are using right now. We have ToS in 2LTT now, but the semantics is in the inner theory. What we can do though, is to assume that the inner theory is also a 2LTT. Then we might assume that the inner theory of *that* is again a 2LTT, and so on. This is a possible (and quite natural) generalization of 2LTT to n-level type theory. In this setting, one round of self-description requires a bumping of levels in the sense of n-level TT. In this thesis we do not explore this, instead we use a more conventional universe hierarchy in an extensional TT, to investigate self-description.

Example 28. We have seen in Example 15 that Ty_0-valued presheaves have finitary signatures. With infinitary signatures, we can also cover monads on Ty_0. We assume a universe $\mathsf{U}_0 : \mathsf{Ty}_0$.

$$\begin{aligned}
&\mathsf{M} \quad\quad : \mathsf{U}_0 \to \mathsf{U}\\
&\mathsf{map} \quad : (A \to B) \to^{\mathsf{ext}} \mathsf{M}\,A \to \mathsf{El}\,(\mathsf{M}\,B)\\
&\mathsf{map}_{\mathsf{id}} \ : \mathsf{El}\,(\mathsf{Id}\,(\mathsf{map}\,\mathsf{id}\,x)\,x)\\
&\mathsf{map}_{\circ} \ : \mathsf{El}\,(\mathsf{Id}\,(\mathsf{map}\,(f \circ g)\,x)\,(\mathsf{map}\,f\,(\mathsf{map}\,g\,x)))\\
&\mathsf{return} : A \to^{\mathsf{ext}} \mathsf{El}\,(\mathsf{M}\,A)\\
&\mathsf{bind} \quad : M\,A \to (A \to^{\mathsf{inf}} \mathsf{M}\,B) \to \mathsf{El}\,(\mathsf{M}\,B)
\end{aligned}$$

We rely on $\to^{\mathsf{inf}}$ to specify binding. The *join*-based specification would not work, since $\mathsf{M}\,(\mathsf{M}\,A)$ is not valid in signatures. The above signature can be helpful for deriving some of the metatheory of Dijkstra monads [MAA+19, Section 5].

In the 2LTT-based semantics, we will get $\mathsf{M} : \mathsf{U}_0 \to \mathsf{Ty}_0$, which is not quite an endofunctor, but it would be straightforward to refine the semantics so that

we can pick universes for underlying types of algebras, and have $\mathsf{M} : \mathsf{U}_0 \to \mathsf{U}_0$. Alternatively, we will also have the ETT-based semantics in Section 5.6, where universe levels are explicitly handled.

Example 29. It is worth to note that every set-truncated higher inductive type from the Homotopy Type Theory book [Uni13] is covered. This includes

- The cumulative hierarchy of sets [Uni13, Section 10.5].
- Cauchy real numbers [Uni13, Section 11.3].
- Surreal numbers [Uni13, Section 11.6].

5.2 Semantics

5.2.1 Overview

As we mentioned, we need a different semantics for infinitary signatures. First, we look at why the previous semantics fails. We try to model signatures again as flcwfs, and morphisms as strict flcwf-morphisms. The simplest point of failure is the interpretation of the unit type $\top : \mathbf{Tm}\,\boldsymbol{\Gamma}\,\mathbf{U}$.

In the semantics, this is the same as defining $\top : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\mathbf{Ty_0}$, where $\mathbf{Ty_0}$ is the flcwf of inner types. The only sensible definition here is the functor which is constantly $\top_0$. But this does not strictly preserve context comprehension or the finite limit type formers. If we have

$$\top : \mathsf{Con}_{\boldsymbol{\Gamma}} \to \mathsf{Ty}_0$$
$$\top\,\Gamma :\equiv \top_0$$

then we have $\top\,(\Gamma \triangleright_{\boldsymbol{\Gamma}} A) \equiv \top_0$, but $\top\,\Gamma \triangleright_{\mathbf{Ty_0}} \top\,A \equiv \top_0 \times \top_0$. Thus, $\top_0 \not\equiv \top_0 \times \top_0$, but of course $\top_0 \simeq \top_0 \times \top_0$.

Let us look at $\boldsymbol{\Pi}^{\mathbf{inf}} : (A : \mathbf{Ty_0}) \to (A \to \mathbf{Tm}\,\boldsymbol{\Gamma}\,\mathbf{U}) \to \mathbf{Tm}\,\boldsymbol{\Gamma}\,\mathbf{U}$ as well, since that is a more interesting new feature than the unit type. The only viable definition is to take the A-indexed product of $\mathbf{Sub}\,\boldsymbol{\Gamma}\,\mathbf{Ty_0}$ morphisms, so we map objects of $\boldsymbol{\Gamma}$ to function types:

$$\mathsf{Con}_{\boldsymbol{\Pi}^{\mathbf{inf}}\,\boldsymbol{A}\,\boldsymbol{b}}\,\Gamma :\equiv (\alpha : A) \to \mathsf{Con}_{\boldsymbol{b}\,\alpha}\,\Gamma$$

But now we have

$$(\mathbf{\Pi}^{\mathbf{inf}}\ \boldsymbol{A}\ \boldsymbol{b})\bullet_{\boldsymbol{\Gamma}} \equiv (\alpha : A) \to \mathsf{Con}_{\boldsymbol{b}\,\alpha}\bullet_{\boldsymbol{\Gamma}} \equiv A \to \top_0$$

Hence, $(\mathbf{\Pi}^{\mathbf{inf}}\ \boldsymbol{A}\ \boldsymbol{b})\bullet_{\boldsymbol{\Gamma}} \not\equiv \top_0$, although $(\mathbf{\Pi}^{\mathbf{inf}}\ \boldsymbol{A}\ \boldsymbol{b})\bullet_{\boldsymbol{\Gamma}} \simeq \top_0$.

Intuitively, if $\mathbf{U}$ has no type formers, the terms in $\mathbf{U}$ are neutral, i.e. variables applied to zero or more neutral terms. But variables in the semantics simply project out components from iterated Σ-types. For example, the action of $\mathbf{q} : \mathbf{Tm}\,(\boldsymbol{\Gamma} \triangleright \boldsymbol{A})\,(\boldsymbol{A}[\mathbf{p}])$ on objects, types, morphisms and terms is given by taking second projections. Since all structure in $\boldsymbol{\Gamma} \triangleright \boldsymbol{A}$ is given by pairing things, $\mathbf{q}$ strictly preserves all structure, and the same goes for all variables.

Substitutions and terms in the finitary ToS are only allowed to freely reshuffle structure. We can forget, duplicate, or permute signature entries, or build neutral expressions from assumptions. In contrast, the infinitary ToS allows us to take small limits of assumptions, using $\top$, $\mathbf{\Sigma}$, $\mathbf{Id}$ and $\mathbf{\Pi}^{\mathbf{inf}}$ to build new inhabitants of $\mathbf{U}$. We summarize the process of getting the new semantics:

1. Strict structure-preservation for type formers in $\mathbf{U}$ generally fails, but they still preserve structure up to isomorphism.

2. Hence, we switch from strict flcwf-morphisms to weak ones, which preserve $\bullet$, comprehension and fl-structure weakly.

3. However, in the finitary case we often relied on transporting along preservation equations. We need to recover transports along isomorphism.

4. Hence, we extend semantic types from displayed flcwfs to *isofibrations*, which support the required transports.

5. However, this rules out sort equations, because they are not stable under isomorphisms. For example, for sets A, B, C such that $A \simeq B$ and $A \simeq C$, it is not necessarily the case that $B \equiv C$.

Univalent semantics

The isofibrant semantics will turn out to be significantly more technical than the strict semantics. Instead of working with isofibrations in an extensional setting, could we work with univalent structures in homotopy type theory? In other words, work with univalent categories of algebras, and univalent displayed categories over

them [AL19]. A major benefit of the univalent setting is that we would get a *structure identity principle* [Acz11] out of the semantics, which says that for algebras, isomorphism is the same as equality.

However, it appears that univalent *cwfs* are overall yet more technical to handle than isofibrations. In an univalent cwf, objects and types are generally h-groupoids, so we would have groupoids of algebras instead of sets of algebras. This implies that *type equalities* are between groupoids, so they need to be coherent, if we want them to be well-behaved. Hence, Ty is not an 1-presheaf over contexts, but rather a $(2,1)$-presheaf.

Alternatively, we could simplify the task by only constructing univalent categories of algebras, and skipping the family structure (and fl-structure). This would be the minimum amount of effort that would yield the structure identity principle.

Both of these would be interesting to check in future work. As a third alternative, instead of stopping at set-truncated algebras in HoTT, we might as well consider types at arbitrary h-levels, and construct $(\omega, 1)$-categories of algebras. This comprises a semantics of higher inductive-inductive signatures. We do not present a full higher-categorical semantics in this thesis; we only present a fragment of it in Chapter 6.

5.2.2 Model of the Theory of Signatures

In the following we present a model of ToS. We call it $\mathbf{M}$, and like before, we use **bold** font to refer to components of $\mathbf{M}$.

Contexts

$\boldsymbol{\Gamma} : \mathbf{Con}$ is again an flcwf, but with a minor change: K is not strict anymore, so we have $(\mathsf{app}_\mathsf{K}, \mathsf{lam}_\mathsf{K}) : \mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta) \simeq \mathsf{Sub}\,\Gamma\,\Delta$. As we will see shortly, $\boldsymbol{A}[\boldsymbol{\sigma}]$ does not support strict displayed K anymore, hence the change.

Substitutions

$\boldsymbol{\sigma} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\boldsymbol{\Delta}$ is a *weak flcwf-morphism*, that is, a functor between underlying categories, which also maps types to types and terms to terms, and satisfies the following mere properties:

1. $\boldsymbol{\sigma}\,(A[\sigma]) \equiv (\boldsymbol{\sigma}\,A)\,[\boldsymbol{\sigma}\,\sigma]$

2. $\boldsymbol{\sigma}\,(t[\sigma]) \equiv (\boldsymbol{\sigma}\,t)\,[\boldsymbol{\sigma}\,\sigma]$

3. The unique map $\epsilon : \mathsf{Sub}\,(\boldsymbol{\sigma}\,\bullet)\,\bullet$ has a retraction.

4. Each $(\boldsymbol{\sigma}\,\mathsf{p},\, \boldsymbol{\sigma}\,\mathsf{q}) : \mathsf{Sub}\,(\boldsymbol{\sigma}\,(\Gamma \triangleright A))\,(\boldsymbol{\sigma}\,\Gamma \triangleright \boldsymbol{\sigma}\,A)$ has an inverse.

In short, $\boldsymbol{\sigma}$ preserves substitution strictly and preserves empty context and context extension up to isomorphism. We notate the evident isomorphisms as $\boldsymbol{\sigma}_\bullet : \boldsymbol{\sigma}\,\bullet \simeq \bullet$ and $\boldsymbol{\sigma}_\triangleright : \boldsymbol{\sigma}\,(\Gamma \triangleright A) \simeq \boldsymbol{\sigma}\,\Gamma \triangleright \boldsymbol{\sigma}\,A$. Our notion of weak morphism is the same as in [BCM+20], when restricted to cwfs.

Theorem 7. *Every* $\boldsymbol{\sigma} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\boldsymbol{\Delta}$ *preserves fl-structure up to type isomorphism. That is, we have*

$$\boldsymbol{\sigma}_\Sigma : \boldsymbol{\sigma}\,(\Sigma\,A\,B) \simeq \Sigma\,(\boldsymbol{\sigma}\,A)\,((\boldsymbol{\sigma}\,B)[\boldsymbol{\sigma}_\triangleright^{-1}])$$
$$\boldsymbol{\sigma}_\mathsf{K} : \boldsymbol{\sigma}\,(\mathsf{K}\,\Delta) \simeq \mathsf{K}\,(\boldsymbol{\sigma}\,\Delta)$$
$$\boldsymbol{\sigma}_\mathsf{Id} : \boldsymbol{\sigma}\,(\mathsf{Id}\,t\,u) \simeq \mathsf{Id}\,(\boldsymbol{\sigma}\,t)\,(\boldsymbol{\sigma}\,u)$$

These are all natural in the following sense: for $\sigma : \mathsf{Sub}_{\boldsymbol{\Gamma}}\,\Gamma\,\Delta$*, if we have* $\boldsymbol{\sigma}_\Sigma$ *as a type isomorphism in* $\boldsymbol{\sigma}\,\Delta$*, if we reindex it by* σ*, we get* $\boldsymbol{\sigma}_\Sigma$ *as a type isomorphism in* $\boldsymbol{\sigma}\,\Gamma$*. The same holds for* $\boldsymbol{\sigma}_\mathsf{K}$ *and* $\boldsymbol{\sigma}_\mathsf{Id}$*.*

Moreover, $\boldsymbol{\sigma}$ *preserves all term and substitution formers in the fl-structure. For example,* $\boldsymbol{\sigma}\,(\mathsf{proj1}\,t) \equiv \mathsf{proj1}\,(\boldsymbol{\sigma}_\Sigma[\mathsf{id}, \boldsymbol{\sigma}\,t])$*.*

Proof. For $\boldsymbol{\sigma}_\Sigma$, we construct the following context isomorphism:

$$(\boldsymbol{\sigma}\,\Gamma \triangleright \boldsymbol{\sigma}\,(\Sigma\,A\,B)) \simeq (\boldsymbol{\sigma}\,\Gamma \triangleright \boldsymbol{\sigma}\,A \triangleright (\boldsymbol{\sigma}\,B)[\boldsymbol{\sigma}_\triangleright^{-1}])$$
$$\simeq (\boldsymbol{\sigma}\,\Gamma \triangleright \Sigma\,(\boldsymbol{\sigma}\,A)\,((\boldsymbol{\sigma}\,B)[\boldsymbol{\sigma}_\triangleright^{-1}]))$$

This isomorphism is the identity on $\boldsymbol{\sigma}\,\Gamma$, hence we can extract the desired $\boldsymbol{\sigma}_\Sigma : \boldsymbol{\sigma}\,(\Sigma\,A\,B) \simeq \Sigma\,(\boldsymbol{\sigma}\,A)\,((\boldsymbol{\sigma}\,B)[\boldsymbol{\sigma}_\triangleright^{-1}])$ from it.

For $\boldsymbol{\sigma}_\mathsf{K}$, note the following:

$$(\bullet \triangleright \boldsymbol{\sigma}\,(\mathsf{K}\,\Delta)) \simeq (\boldsymbol{\sigma}\,\bullet \triangleright \boldsymbol{\sigma}\,(\mathsf{K}\,\Delta)) \simeq \boldsymbol{\sigma}\,(\bullet \triangleright \mathsf{K}\,\Delta)$$
$$\simeq \boldsymbol{\sigma}\,\Delta \simeq (\bullet \triangleright K\,(\boldsymbol{\sigma}\,\Delta))$$

This yields a type isomorphism $\boldsymbol{\sigma}\,(\mathsf{K}\,\Delta) \simeq \mathsf{K}\,(\boldsymbol{\sigma}\,\Delta)$ in the empty context, and we can use the functorial action of $\epsilon : \mathsf{Sub}\,\Gamma\,\bullet$ to weaken it to any Γ context.

For $\boldsymbol{\sigma}_\mathsf{Id}$, both component morphisms can be constructed by refl and equality reflection, and the morphisms are inverses by UIP. We omit here the verification of naturality and that $\boldsymbol{\sigma}$ preserves term and substitution formers in the fl-structure. □

Identity and composition

$\mathbf{id} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\boldsymbol{\Gamma}$ is defined in the obvious way, with identities for underlying functions and for preservation morphisms.

For $\boldsymbol{\sigma} \circ \boldsymbol{\delta}$, the underlying functions are given by function composition, and the preservation morphisms are given as follows:

$$(\boldsymbol{\sigma} \circ \boldsymbol{\delta})^{-1}_{\bullet} :\equiv \boldsymbol{\sigma}\,\boldsymbol{\delta}^{-1}_{\bullet} \circ \boldsymbol{\delta}^{-1}_{\bullet}$$
$$(\boldsymbol{\sigma} \circ \boldsymbol{\delta})^{-1}_{\triangleright} :\equiv \boldsymbol{\sigma}\,\boldsymbol{\delta}^{-1}_{\triangleright} \circ \boldsymbol{\delta}^{-1}_{\triangleright}$$

It is easy to verify the left and right identity laws and associativity for $- \circ -$.

Lemma 6. The derived preservation isomorphisms for the fl-structure can be decomposed analogously; all derived isomorphisms in $\mathbf{id}$ are identities, and we have

$$(\boldsymbol{\sigma} \circ \boldsymbol{\delta})_{\Sigma} \equiv \boldsymbol{\sigma}\,\boldsymbol{\delta}_{\Sigma} \circ \boldsymbol{\delta}_{\Sigma}$$
$$(\boldsymbol{\sigma} \circ \boldsymbol{\delta})_{\mathsf{K}} \equiv \boldsymbol{\sigma}\,\boldsymbol{\delta}_{\mathsf{K}} \circ \boldsymbol{\delta}_{\mathsf{K}}$$
$$(\boldsymbol{\sigma} \circ \boldsymbol{\delta})_{\mathsf{Id}} \equiv \boldsymbol{\sigma}\,\boldsymbol{\delta}_{\mathsf{Id}} \circ \boldsymbol{\delta}_{\mathsf{Id}}$$

On the right sides, $- \circ -$ refers to composition of type morphisms.

Proof. In the case of Id, the equations hold immediately by UIP. For Σ and K, we prove by flcwf computation and straightforward unfolding of definitions. □

Empty context

The empty context $\bullet : \mathbf{Con}$ is the same as before, i.e. the terminal flcwf. Since the unique $\boldsymbol{\epsilon} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\bullet$ morphism strictly preserves all structure, it also a weak morphism.

Types

We define $\mathbf{Ty}\,\boldsymbol{\Gamma} : \mathsf{Set}$ as the type of split flcwf-isofibrations over $\boldsymbol{\Gamma}$. This consists of a displayed flcwf together with *iso-cleaving* structure. For the displayed flcwf part, we reuse previous notation from Section 4.2.4. For the iso-cleaving, we make some auxiliary definitions first.

Definition 59 (Displayed type categories)**.** For each $\Gamma : \mathsf{Con}_{\boldsymbol{A}}\,\underline{\Gamma}$, there is a displayed category over the type category $\mathsf{Ty}_{\boldsymbol{\Gamma}}\,\underline{\Gamma}$, whose objects over $\underline{A} : \mathsf{Ty}_{\boldsymbol{\Gamma}}\,\underline{\Gamma}$ are

elements of $\mathsf{Ty}_{\boldsymbol{A}}\,\Gamma\,\underline{A}$, and displayed morphisms over $\underline{t} : \mathsf{Tm}_{\boldsymbol{\Gamma}}\,(\underline{\Gamma} \triangleright \underline{A})\,(\underline{B}[\mathsf{p}])$ are elements of $\mathsf{Tm}_{\boldsymbol{A}}\,(\Gamma \triangleright A)\,(B[\mathsf{p}])\,\underline{t}$. The identity morphism is given by $\mathsf{q}_{\boldsymbol{A}}$, and the composition of t and u is $t[\mathsf{p}_{\boldsymbol{A}}, u]$. Analogously to Definition 44, this extends to a displayed split indexed category.

Definition 60 (Displayed isomorphisms)**.** A *displayed context isomorphism* over $\underline{\sigma} : \underline{\Gamma} \simeq \underline{\Delta}$, notated $\sigma : \Gamma \simeq_{\underline{\sigma}} \Delta$, is an invertible displayed morphism $\sigma : \mathsf{Sub}_{\boldsymbol{A}}\,\Gamma\,\Delta\,\underline{\sigma}$, with inverse $\sigma^{-1} : \mathsf{Sub}_{\boldsymbol{A}}\,\Delta\,\Gamma\,\underline{\sigma}^{-1}$. A *displayed type isomorphism* over $\underline{t} : \underline{A} \simeq \underline{B}$, notated $t : A \simeq_{\underline{t}} B$, is an isomorphism in a displayed type category.

Definition 61. A *vertical morphism* lies over an identity morphism. We use this definition for context morphisms (substitutions) and type morphisms as well.

Definition 62 (Context iso-cleaving)**.** This lifts a base context isomorphism to a displayed one. It consists of

$$
\begin{aligned}
\mathsf{coe} &: \underline{\Gamma} \simeq \underline{\Delta} \to \mathsf{Con}_{\boldsymbol{A}}\,\underline{\Gamma} \to \mathsf{Con}_{\boldsymbol{A}}\,\underline{\Delta} \\
\mathsf{coh} &: (\underline{\sigma} : \underline{\Gamma} \simeq \underline{\Delta})(\Gamma : \mathsf{Con}_{\boldsymbol{A}}\,\underline{\Gamma}) \to \Gamma \simeq_{\underline{\sigma}} \mathsf{coe}\,\underline{\sigma}\,\Gamma \\
\mathsf{coe}^{\mathsf{id}} &: \mathsf{coe}\,\mathsf{id}\,\Gamma \equiv \Gamma \\
\mathsf{coe}^{\circ} &: \mathsf{coe}\,(\underline{\sigma} \circ \underline{\delta})\,\Gamma \equiv \mathsf{coe}\,\underline{\sigma}\,(\mathsf{coe}\,\underline{\delta}\,\Gamma) \\
\mathsf{coh}^{\mathsf{id}} &: \mathsf{coh}\,\mathsf{id}\,\Gamma \equiv \mathsf{id} \\
\mathsf{coh}^{\circ} &: \mathsf{coh}\,(\underline{\sigma} \circ \underline{\delta})\,\Gamma \equiv \mathsf{coh}\,\underline{\sigma}\,(\mathsf{coe}\,\underline{\delta}\,\Gamma) \circ \mathsf{coh}\,\underline{\delta}\,\Gamma
\end{aligned}
$$

Here, coe and coh abbreviate "coercion" and "coherence" respectively.

Definition 63 (Type iso-cleaving)**.** This consists of

$$
\begin{aligned}
\mathsf{coe} &: \underline{A} \simeq \underline{B} \to \mathsf{Ty}_{\boldsymbol{A}}\,\Gamma\,\underline{A} \to \mathsf{Ty}_{\boldsymbol{A}}\,\Gamma\,\underline{B} \\
\mathsf{coh} &: (\underline{t} : \underline{A} \simeq \underline{B})(A : \mathsf{Ty}_{\boldsymbol{A}}\,\Gamma\,\underline{A}) \to A \simeq_{\underline{t}} \mathsf{coe}\,\underline{t}\,A \\
\mathsf{coe}^{\mathsf{id}} &: \mathsf{coe}\,\mathsf{id}\,A \equiv A \\
\mathsf{coe}^{\circ} &: \mathsf{coe}\,(\underline{t} \circ \underline{u})\,A \equiv \mathsf{coe}\,\underline{t}\,(\mathsf{coe}\,\underline{u}\,A) \\
\mathsf{coh}^{\mathsf{id}} &: \mathsf{coh}\,\mathsf{id}\,A \equiv \mathsf{id} \\
\mathsf{coh}^{\circ} &: \mathsf{coh}\,(\underline{t} \circ \underline{u})\,A \equiv \mathsf{coh}\,\underline{t}\,(\mathsf{coe}\,\underline{u}\,A) \circ \mathsf{coh}\,\underline{u}\,A
\end{aligned}
$$

Additionally, for $\sigma : \mathsf{Sub}_{\boldsymbol{A}}\,\Gamma\,\Delta\,\underline{\sigma}$, we have

$$
\begin{aligned}
\mathsf{coe}[] &: \mathsf{coe}\,(\underline{t}[\underline{\sigma} \circ \mathsf{p}, \mathsf{q}])\,(A[\sigma]) \equiv (\mathsf{coe}\,\underline{t}\,A)[\sigma] \\
\mathsf{coh}[] &: \mathsf{coh}\,(\underline{t}[\underline{\sigma} \circ \mathsf{p}, \mathsf{q}])\,(A[\sigma]) \equiv (\mathsf{coh}\,\underline{t}\,A)[\sigma]
\end{aligned}
$$

Definition 64. A *split flcwf isofibration* is a displayed flCwF equipped with isocleaving for contexts and types.

Remark. It is not possible to model types as fibrations or opfibrations, because we have no restriction on the variance of ToS types. For example, the type which extends a pointed set signature to a natural number signature, is neither a fibration nor an opfibration.

Type substitution

We aim to define $-[-] : \mathbf{Ty}\,\Delta \to \mathbf{Sub}\,\Gamma\,\Delta \to \mathbf{Ty}\,\Gamma$, such that $\boldsymbol{A}[\mathbf{id}] \equiv \boldsymbol{A}$ and $\boldsymbol{A}[\boldsymbol{\sigma} \circ \boldsymbol{\delta}] \equiv \boldsymbol{A}[\boldsymbol{\sigma}][\boldsymbol{\delta}]$. As before, the underlying sets are given by simple composition:

$$\begin{aligned}
\mathsf{Con}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\,\underline{\Gamma} &:\equiv \mathsf{Con}_{\boldsymbol{A}}\,(\boldsymbol{\sigma}\,\underline{\Gamma}) \\
\mathsf{Sub}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv \mathsf{Sub}_{\boldsymbol{A}}\,\Gamma\,\Delta\,(\boldsymbol{\sigma}\,\underline{\sigma}) \\
\mathsf{Ty}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\,\Gamma\,\underline{A} &:\equiv \mathsf{Ty}_{\boldsymbol{A}}\,\Gamma\,(\boldsymbol{\sigma}\,\underline{A}) \\
\mathsf{Tm}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\,\Gamma\,A\,\underline{t} &:\equiv \mathsf{Tm}_{\boldsymbol{A}}\,\Gamma\,A\,(\boldsymbol{\sigma}\,\underline{t})
\end{aligned}$$

The difference from the finitary case is that instead of preservation equations, we have isomorphisms, coercions and coherence. However, we can recover essentially the same reasoning as before, because all the previous transports still work. Context and type formers are given by coercing $\boldsymbol{A}$ structures along preservation isomorphisms by $\boldsymbol{\sigma}$. For example:

$$\begin{aligned}
\bullet_{\boldsymbol{A}[\boldsymbol{\sigma}]} &:\equiv \mathsf{coe}\,\boldsymbol{\sigma}_{\bullet}^{-1}\,\bullet_{\boldsymbol{A}} \\
\Gamma \triangleright_{\boldsymbol{A}[\boldsymbol{\sigma}]} A &:\equiv \mathsf{coe}\,\boldsymbol{\sigma}_{\triangleright}^{-1}\,(\Gamma \triangleright_{\boldsymbol{A}} A) \\
\mathsf{Id}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\,t\,u &:\equiv \mathsf{coe}\,\boldsymbol{\sigma}_{\mathsf{Id}}^{-1}\,(\mathsf{Id}_{\boldsymbol{A}}\,t\,u) \\
\mathsf{K}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\,\Delta &:\equiv \mathsf{coe}\,\boldsymbol{\sigma}_{\mathsf{K}}^{-1}\,(\mathsf{K}_{\boldsymbol{A}}\,\Delta)
\end{aligned}$$

Term and substitution formers are given by composing coh-lifted isomorphisms with term and substitution formers from $\boldsymbol{A}$. For example:

$$\begin{aligned}
\epsilon_{\boldsymbol{A}[\boldsymbol{\sigma}]} &:\equiv \mathsf{coh}\,\boldsymbol{\sigma}_{\bullet}^{-1}\,\bullet_{\boldsymbol{A}} \circ \epsilon_{\boldsymbol{A}} \\
\mathsf{p}_{\boldsymbol{A}[\boldsymbol{\sigma}]} &:\equiv \mathsf{p}_{\boldsymbol{A}} \circ (\mathsf{coh}\,\boldsymbol{\sigma}_{\triangleright}^{-1}\,(\Gamma \triangleright A))^{-1} \\
(\sigma, {}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\ t) &:\equiv \mathsf{coh}\,\boldsymbol{\sigma}_{\triangleright}^{-1}\,(\Delta \triangleright A) \circ (\sigma, {}_{\boldsymbol{A}}\ t)
\end{aligned}$$

As we mentioned, only weak K is supported in $\boldsymbol{A}[\boldsymbol{\sigma}]$. For strict K we would have to show:

$$\mathsf{Sub}_{\boldsymbol{A}}\,\Gamma\,\Delta\,(\boldsymbol{\sigma}\,\underline{\sigma}) \equiv \mathsf{Tm}_{\boldsymbol{A}}\,\Gamma\,(\mathsf{coe}\,\boldsymbol{\sigma}^{-1}_{\mathsf{K}}\,(\mathsf{K}_{\boldsymbol{A}}\,\Delta))\,(\boldsymbol{\sigma}\,\underline{\sigma})$$

By strict K in $\boldsymbol{A}$, it would be enough to show

$$\mathsf{Tm}_{\boldsymbol{A}}\,\Gamma\,(\mathsf{K}_{\boldsymbol{A}}\,\Delta)\,(\boldsymbol{\sigma}\,\underline{\sigma}) \equiv \mathsf{Tm}_{\boldsymbol{A}}\,\Gamma\,(\mathsf{coe}\,\boldsymbol{\sigma}^{-1}_{\mathsf{K}}\,(\mathsf{K}_{\boldsymbol{A}}\,\Delta))\,(\boldsymbol{\sigma}\,\underline{\sigma})$$

But there is no reason why these sets should be equal, so we instead produce an isomorphism.

Equations for term and type substitution follow from naturality of preservation isomorphisms in $\boldsymbol{\sigma}$, $\mathsf{coe}[]$, $\mathsf{coh}[]$ and substitution equations in $\boldsymbol{A}$.

Iso-cleaving is given by iso-cleaving in $\boldsymbol{A}$ and the action of $\boldsymbol{\sigma}$ on isomorphisms, so that we have $\mathsf{coe}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\,\underline{\sigma}\,\Gamma :\equiv \mathsf{coe}_{\boldsymbol{A}}\,(\boldsymbol{\sigma}\,\underline{\sigma})\,\Gamma$ and $\mathsf{coh}_{\boldsymbol{A}[\boldsymbol{\sigma}]}\,\underline{\sigma}\,\Gamma :\equiv \mathsf{coh}_{\boldsymbol{A}}\,(\boldsymbol{\sigma}\,\underline{\sigma})\,\Gamma$.

Functoriality of type substitution, i.e. $\boldsymbol{A}[\mathbf{id}] \equiv \boldsymbol{A}$ and $\boldsymbol{A}[\boldsymbol{\sigma} \circ \boldsymbol{\delta}] \equiv \boldsymbol{A}[\boldsymbol{\sigma}][\boldsymbol{\delta}]$, follows from Lemma 6 and split cleaving given by $\mathsf{coe}^{\mathsf{id}}$, coe°, $\mathsf{coh}^{\mathsf{id}}$ and coh° laws in $\boldsymbol{A}$.

Terms

$\mathbf{Tm}\,\boldsymbol{\Gamma}\,\boldsymbol{A} : \mathsf{Set}$ is defined as the type of *weak flCwF sections* of $\boldsymbol{A}$. The underlying functions of $\boldsymbol{t} : \mathbf{Tm}\,\boldsymbol{\Gamma}\,\boldsymbol{A}$ are as follows:

$$\begin{aligned}
&\boldsymbol{t} : (\underline{\Gamma} : \mathsf{Con}_{\boldsymbol{\Gamma}}) \to \mathsf{Con}_{\boldsymbol{A}}\,\underline{\Gamma}\\
&\boldsymbol{t} : (\underline{\sigma} : \mathsf{Sub}_{\boldsymbol{\Gamma}}\,\underline{\Gamma}\,\underline{\Delta}) \to \mathsf{Sub}_{\boldsymbol{A}}\,(\boldsymbol{t}\,\underline{\Gamma})\,(\boldsymbol{t}\,\underline{\Delta})\,\underline{\sigma}\\
&\boldsymbol{t} : (\underline{A} : \mathsf{Ty}_{\boldsymbol{\Gamma}}) \to \mathsf{Ty}_{\boldsymbol{A}}\,(\boldsymbol{t}\,\underline{\Gamma})\,\underline{A}\\
&\boldsymbol{t} : (\underline{t} : \mathsf{Tm}_{\boldsymbol{\Gamma}}\,\underline{\Gamma}\,\underline{A}) \to \mathsf{Tm}_{\boldsymbol{A}}\,(\boldsymbol{t}\,\underline{\Gamma})\,(\boldsymbol{t}\,\underline{A})\,\underline{t}
\end{aligned}$$

Such that

1. $\boldsymbol{t}\,(\underline{A}[\underline{\sigma}]) \equiv (\boldsymbol{t}\,\underline{A})\,[\boldsymbol{t}\,\underline{\sigma}]$
2. $\boldsymbol{t}\,(\underline{t}[\underline{\sigma}]) \equiv (\boldsymbol{t}\,\underline{t})\,[\boldsymbol{t}\,\underline{\sigma}]$
3. The unique $\epsilon_{\boldsymbol{A}} : \mathsf{Sub}\,(\boldsymbol{t}\,\bullet)\,\bullet\,\mathsf{id}$ has a vertical retraction.
4. Each $(\boldsymbol{t}\,\mathsf{p}, \boldsymbol{t}\,\mathsf{q}) : \mathsf{Sub}\,(\boldsymbol{t}\,(\underline{\Gamma} \triangleright \underline{A}))\,(\boldsymbol{t}\,\underline{\Gamma} \triangleright \boldsymbol{t}\,\underline{A})\,\mathsf{id}$ has a vertical inverse.

Similarly to what we had in **Sub**, we denote the evident preservation isomorphisms as $\boldsymbol{t}_{\bullet} : \boldsymbol{t}\,\bullet \simeq_{\mathsf{id}} \bullet$ and $\boldsymbol{t}_{\triangleright} : \boldsymbol{t}\,(\underline{\Gamma} \triangleright \underline{A}) \simeq_{\mathsf{id}} \boldsymbol{t}\,\underline{\Gamma} \triangleright \boldsymbol{t}\,\underline{A}$. In short, weak sections

are dependently typed analogues of weak morphisms, with dependent underlying functions and displayed preservation isomorphisms. We also have the derived fl-preservation isomorphisms.

Theorem 8. *A weak section* $\boldsymbol{t} : \mathbf{Tm}\,\boldsymbol{\Gamma}\,\boldsymbol{A}$ *preserves fl-structure up to vertical type isomorphisms, that is, the following are derivable:*

$$\boldsymbol{t}_\Sigma : \boldsymbol{t}\,(\Sigma\,\underline{A}\,\underline{B}) \simeq_{\mathsf{id}} \Sigma\,(\boldsymbol{t}\,\underline{A})\,((\boldsymbol{t}\,\underline{B})[\boldsymbol{t}_\triangleright^{-1}])$$
$$\boldsymbol{t}_\mathsf{K} : \boldsymbol{t}\,(\mathsf{K}\,\underline{\Delta}) \simeq_{\mathsf{id}} \mathsf{K}\,(\boldsymbol{t}\,\underline{\Delta})$$
$$\boldsymbol{t}_\mathsf{Id} : \boldsymbol{t}\,(\mathsf{Id}\,\underline{t}\,\underline{u}) \simeq_{\mathsf{id}} \mathsf{Id}\,(\boldsymbol{t}\,\underline{t})\,(\boldsymbol{t}\,\underline{u})$$

Also, the above isomorphisms are natural in the sense of Theorem 7, and $\boldsymbol{t}$ *preserves term and substitution formers in the fl-structure.*

Proof. The construction of isomorphisms is the same as in Theorem 7. Indeed, every construction there has a displayed counterpart which we can use here. □

We note though that the move from Theorem 7 to here is not simply a logical predicate translation, because we are only lifting the codomain of a weak morphism to a displayed version, and we leave the domain non-displayed. We leave to future work the investigation of such asymmetrical logical predicate translations.

Term substitution

$-[-] : \mathbf{Tm}\,\boldsymbol{\Delta}\,\boldsymbol{A} \to (\boldsymbol{\sigma} : \mathbf{Sub}\,\boldsymbol{\Gamma}\,\boldsymbol{\Delta}) \to \mathbf{Tm}\,\boldsymbol{\Gamma}\,(\boldsymbol{A}[\boldsymbol{\sigma}])$ is given similarly to $-\circ-$ in Section 5.2.2. Underlying functions are given by function composition, and preservation morphisms are also similar:

$$(\boldsymbol{t}[\boldsymbol{\sigma}])_\bullet^{-1} :\equiv \boldsymbol{t}\,\boldsymbol{\sigma}_\bullet^{-1} \circ \boldsymbol{t}_\bullet^{-1}$$
$$(\boldsymbol{t}[\boldsymbol{\sigma}])_\triangleright^{-1} :\equiv \boldsymbol{t}\,\boldsymbol{\sigma}_\triangleright^{-1} \circ \boldsymbol{t}_\triangleright^{-1}$$

We also have the same decomposition of derived isomorphisms as in Lemma 6. We do not have to show functoriality of term substitution here, since that is derivable in any cwf, see e.g. [KKA19].

Comprehension

$\boldsymbol{\Gamma} \triangleright \boldsymbol{A} : \mathbf{Con}$ is defined as the total flcwf of $\boldsymbol{A}$, in exactly the same way as in the finitary case, since the additional iso-cleaving structure plays no role in the result.

$\mathbf{p} : \mathbf{Sub}\,(\Gamma \triangleright A)\,\Gamma$ and $\mathbf{q} : \mathbf{Tm}\,(\Gamma \triangleright A)\,(A[\mathbf{p}])$ are likewise unchanged; they are strict morphisms, so also automatically weak morphisms. Substitution extension (σ, t) is given by pointwise combining σ and t, e.g. $\mathsf{Con}_{(\sigma,t)}\,\underline{\Gamma} :\equiv (\sigma\,\underline{\Gamma},\, t\,\underline{\Gamma})$.

Strict constant families

We have the same definition for $\mathbf{K}\,\Delta : \mathbf{Ty}\,\Gamma$ as in the finitary case, although we need to define iso-cleaving in addition. Fortunately, coercions and coherences are all trivial, because $\mathbf{K}\,\Delta$ does not actually depend on Γ.

$$\begin{aligned}\mathsf{coe}_{\mathbf{K}\,\Delta}\,\underline{\sigma}\,\Gamma &:\equiv \Gamma\\ \mathsf{coe}_{\mathbf{K}\,\Delta}\,\underline{t}\,A &:\equiv A\end{aligned}$$

Universe

$\mathbf{U} : \mathbf{Ty}\,\Gamma$ is exactly the same as before. We define it as the type which is constantly the flcwf of inner types, so it inherits the trivial iso-cleaving from $\mathbf{K}$.

$\mathbf{El}\,a : \mathbf{Ty}\,\Gamma$ is again the displayed flcwf of the elements of $a : \mathbf{Tm}\,\Gamma\,\mathbf{U}$. The underlying sets are unchanged:

$$\begin{aligned}\mathsf{Con}_{\mathbf{El}\,a}\,\underline{\Gamma} &:\equiv \mathsf{Tm}_0\,(a\,\underline{\Gamma})\\ \mathsf{Sub}_{\mathbf{El}\,a}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv a\,\underline{\sigma}\,\Gamma \equiv \Delta\\ \mathsf{Ty}_{\mathbf{El}\,a}\,\Gamma\,\underline{A} &:\equiv \mathsf{Tm}_0\,(a\,\underline{A}\,\Gamma)\\ \mathsf{Tm}_{\mathbf{El}\,a}\,\Gamma\,A\,\underline{t} &:\equiv a\,\underline{t}\,\Gamma \equiv A\end{aligned}$$

We need to adjust definitions to show that $\mathbf{El}\,a$ supports all required structure. Previously, all context and type formers were inherited from $\mathbf{U}$, since a strictly preserved them. Now, a preserves structure up to (definitional) isomorphism of inner types. Hence, the adjustments are quite mechanical; they are like wrapping all definitions in "unary record constructors" given by preservation isomorphisms. For example:

$$\begin{aligned}\bullet_{\mathbf{El}\,a} &:\equiv a_\bullet^{-1}\,\mathsf{tt}\\ (\Gamma \triangleright_{\mathbf{El}\,a} A) &:\equiv a_\triangleright^{-1}\,(\Gamma,\, A)\end{aligned}$$

We likewise use preservation isomorphisms to define K, Id and Σ. Context coercion is $\mathsf{coe}\,\underline{\sigma}\,\Gamma :\equiv a\,\underline{\sigma}\,\Gamma$. Type coercion, for $A : a\,\underline{A}\,\Gamma$ is given as $\mathsf{coe}\,\underline{t}\,A :\equiv a\,\underline{t}\,(a_\triangleright^{-1}\,(\Gamma,\, A))$.

Unit type

$\top : \mathbf{Tm}\,\Gamma\,\mathbf{U}$ is the constantly $\top_0$ morphism, i.e. it maps objects to $\top_0$ and types to $\lambda\_.\,\top_0$, and maps morphisms and terms to the identity function. It clearly preserves $\bullet$ and $-\triangleright-$ up to isomorphism.

Sigma type

For $\boldsymbol{a} : \mathbf{Tm}\,\Gamma\,\mathbf{U}$ and $\boldsymbol{b} : \mathbf{Tm}\,(\Gamma \triangleright \mathbf{El}\,\boldsymbol{a})\,\mathbf{U}$, we define $\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b} : \mathbf{Tm}\,\Gamma\,\mathbf{U}$ as the component-wise Σ of $\boldsymbol{a}$ and $\boldsymbol{b}$. For the action on $\underline{\Gamma} : \mathsf{Con}_{\Gamma}$, we have:

$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{\Gamma} : \mathsf{Ty}_0$$
$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{\Gamma} :\equiv (\alpha : \boldsymbol{a}\,\underline{\Gamma}) \times \boldsymbol{b}\,(\underline{\Gamma},\, \alpha)$$

For the action on $\underline{\sigma} : \mathsf{Sub}\,\underline{\Gamma}\,\underline{\Delta}$, we have:

$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{\sigma} : (\alpha : \boldsymbol{a}\,\underline{\Gamma}) \times \boldsymbol{b}\,(\underline{\Gamma},\, \alpha) \to (\alpha : \boldsymbol{a}\,\underline{\Delta}) \times \boldsymbol{b}\,(\underline{\Delta},\, \alpha)$$
$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{\sigma} :\equiv \lambda\,(\alpha,\, \beta).\,(\boldsymbol{a}\,\underline{\sigma}\,\alpha,\, \boldsymbol{b}\,(\underline{\sigma},\, \mathsf{refl})\,\beta)$$

Above, the second field should have type $\boldsymbol{b}\,(\underline{\Delta},\, \boldsymbol{a}\,\underline{\sigma}\,\alpha)$, while $\beta : \boldsymbol{b}\,(\underline{\Gamma},\, \alpha)$. Therefore we need a morphism in $\mathbf{\Gamma} \triangleright \mathbf{El}\,\boldsymbol{a}$ from $(\underline{\Gamma},\, \alpha)$ to $(\underline{\Delta},\, \boldsymbol{a}\,\underline{\sigma}\,\alpha)$, which is defined as $(\underline{\sigma},\, \mathsf{refl})$, where $\mathsf{refl} : \boldsymbol{a}\,\underline{\sigma}\,\alpha \equiv \boldsymbol{a}\,\underline{\sigma}\,\alpha$. The action on $\underline{A} : \mathsf{Ty}\,\underline{\Gamma}$ is

$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{A} : (\alpha : \boldsymbol{a}\,\underline{\Gamma}) \times \boldsymbol{b}\,(\underline{\Gamma},\, \alpha) \to \mathsf{Ty}_0$$
$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{A} :\equiv \lambda\,(\alpha,\, \beta).\,(\alpha' : \boldsymbol{a}\,\underline{A}\,\alpha) \times \boldsymbol{b}\,(\underline{A},\, \alpha')\,\beta$$

Here we are somewhat running out of notation: we use α' to refer to a *type* over $\alpha : \boldsymbol{a}\,\underline{\Gamma}$ in the displayed cwf of elements $\mathbf{El}\,\boldsymbol{a}$. The action on terms is analogous:

$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{t} : ((\alpha,\, \beta) : (\alpha : \boldsymbol{a}\,\underline{\Gamma}) \times \boldsymbol{b}\,(\underline{\Gamma},\, \alpha)) \to (\alpha' : \boldsymbol{a}\,\underline{A}\,\alpha) \times \boldsymbol{b}\,(\underline{A},\, \alpha')\,\beta$$
$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{t} :\equiv \lambda\,(\alpha,\, \beta).\,(\boldsymbol{a}\,\underline{t}\,\alpha,\, \boldsymbol{b}\,(\underline{t},\, \mathsf{refl})\,\beta)$$

For the preservation of $\bullet$, we need to show $(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{\bullet} \simeq \top_0$. Unfolding the definition, we get $((\alpha : \boldsymbol{a}\,\underline{\bullet}) \times \boldsymbol{b}\,(\underline{\bullet},\, \alpha)) \simeq \top_0$. This holds since $\boldsymbol{a}\,\underline{\bullet} \simeq \top_0$, so $\boldsymbol{a}\,\underline{\bullet}$ is contractible, thus $(\underline{\bullet},\, \alpha) \equiv \underline{\bullet}_{\Gamma \triangleright \mathbf{El}\,\boldsymbol{a}}$, and we also know $\boldsymbol{b}\,\underline{\bullet} \simeq \top_0$. For the preservation $-\triangleright-$, we need

$$(\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,(\underline{\Gamma} \triangleright \underline{A}) \simeq (\gamma : (\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{\Gamma}) \times (\mathbf{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b})\,\underline{A}\,\gamma$$

Unfolding definitions and reassociating Σ on the right side:

$$\begin{aligned}&(\alpha : \boldsymbol{a}\,(\underline{\Gamma} \triangleright \underline{A}))\times \boldsymbol{b}\,((\underline{\Gamma} \triangleright \underline{A}),\,\alpha)\\ &\simeq\\ &(\alpha : \boldsymbol{a}\,\underline{\Gamma})\times(\beta : \boldsymbol{b}\,(\underline{\Gamma},\,\alpha))\times(\alpha' : \boldsymbol{a}\,\underline{A}\,\alpha)\times \boldsymbol{b}\,(\underline{A},\,\alpha')\,\beta\end{aligned}$$

Since $\boldsymbol{a}_{\triangleright} : \boldsymbol{a}\,(\underline{\Gamma} \triangleright \underline{A})) \simeq (\alpha : \boldsymbol{a}\,\underline{\Gamma}) \times (\beta : \boldsymbol{b}\,(\underline{\Gamma},\,\alpha))$, we can rewrite the left side using pattern matching notation as

$$(\boldsymbol{a}_{\triangleright}^{-1}\,(\gamma,\,\alpha) : \boldsymbol{a}\,(\underline{\Gamma} \triangleright \underline{A}))\times \boldsymbol{b}\,((\underline{\Gamma} \triangleright \underline{A}),\,(\gamma,\,\alpha))$$

Now, since $((\underline{\Gamma} \triangleright \underline{A}),\,(\gamma,\,\alpha)) \equiv (\underline{\Gamma},\,\gamma) \triangleright_{\boldsymbol{\Gamma} \triangleright \mathsf{El}\,\boldsymbol{a}} (\underline{A},\,\alpha)$, we know that $\boldsymbol{b}\,((\underline{\Gamma} \triangleright \underline{A}),\,(\gamma,\,\alpha))$ is also isomorphic to the evident Σ type, and the preservation isomorphism follows.

Projections and pairing for $\boldsymbol{\Sigma}\,\boldsymbol{a}\,\boldsymbol{b}$ are defined in the obvious way by componentwise projection and pairing.

Identity

For $\boldsymbol{t}$ and $\boldsymbol{u}$ in $\mathsf{Tm}\,\boldsymbol{\Gamma}\,(\mathsf{El}\,\boldsymbol{a})$, we define $\mathsf{Id}\,\boldsymbol{t}\,\boldsymbol{u} : \mathsf{Tm}\,\boldsymbol{\Gamma}\,\mathsf{U}$ as expressing pointwise equality of weak sections. We rely on the assumption that Ty_0 has identity type.

$$\begin{aligned}(\mathsf{Id}\,\boldsymbol{t}\,\boldsymbol{u})\,\underline{\Gamma} &:\equiv (\boldsymbol{t}\,\underline{\Gamma} = \boldsymbol{u}\,\underline{\Gamma})\\ (\mathsf{Id}\,\boldsymbol{t}\,\boldsymbol{u})\,\underline{A} &:\equiv \lambda\, e.\,(\boldsymbol{t}\,\underline{A} = \boldsymbol{u}\,\underline{A})\end{aligned}$$

Above, $\boldsymbol{t}\,\underline{A} = \boldsymbol{u}\,\underline{A}$ is well-typed because of $e : \boldsymbol{t}\,\underline{\Gamma} = \boldsymbol{u}\,\underline{\Gamma}$. For substitutions, we have to complete a square of equalities:

$$(\mathsf{Id}\,\boldsymbol{t}\,\boldsymbol{u})\,(\underline{\sigma} : \mathsf{Sub}\,\underline{\Gamma}\,\underline{\Delta}) : (\boldsymbol{t}\,\underline{\Gamma} = \boldsymbol{u}\,\underline{\Gamma}) \to (\boldsymbol{t}\,\underline{\Delta} = \boldsymbol{u}\,\underline{\Delta})$$

This can be given by $\boldsymbol{t}\,\underline{\sigma} : \boldsymbol{a}\,\underline{\sigma}\,(\boldsymbol{t}\,\underline{\Gamma}) = \boldsymbol{t}\,\underline{\Delta}$ and $\boldsymbol{u}\,\underline{\sigma} : \boldsymbol{a}\,\underline{\sigma}\,(\boldsymbol{u}\,\underline{\Gamma}) = \boldsymbol{u}\,\underline{\Delta}$. The action on terms is analogous.

The $\bullet$-preservation $(\boldsymbol{t}\,\underline{\bullet} = \boldsymbol{u}\,\underline{\bullet}) \simeq \top_0$ follows from $\boldsymbol{a}\,\underline{\bullet} \simeq \top_0$. For $\triangleright$-preservation, we need to show

$$(\boldsymbol{t}\,(\underline{\Gamma} \triangleright \underline{A}) = \boldsymbol{u}\,(\underline{\Gamma} \triangleright \underline{A})) \simeq ((e : \boldsymbol{t}\,\underline{\Gamma} = \boldsymbol{u}\,\underline{\Gamma}) \times (\boldsymbol{t}\,\underline{A} = \boldsymbol{u}\,\underline{A}))$$

This follows from $\triangleright$-preservation by $\boldsymbol{a}$. Equality reflection and $\mathsf{refl} : \mathsf{Id}\,\boldsymbol{t}\,\boldsymbol{t}$ are also evident.

Infinitary function type

For $Ix : \mathsf{Ty}_0$ and $\boldsymbol{b} : Ix \to \mathbf{Tm}\,\boldsymbol{\Gamma}\,\mathbf{U}$, we aim to define $\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b} : \mathbf{Tm}\,\boldsymbol{\Gamma}\,\mathbf{U}$. The underlying functions are:

$$\begin{aligned}
(\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})\,\underline{\Gamma} &:= (i : Ix) \to \boldsymbol{b}\,i\,\underline{\Gamma}\\
(\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})\,\underline{\sigma} &:= \lambda\,f\,i.\,\boldsymbol{b}\,i\,\underline{\sigma}\,(f\,i)\\
(\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})\,\underline{A} &:= \lambda\,\Gamma.\,(i : Ix) \to \boldsymbol{b}\,i\,\underline{A}\,(\Gamma\,i)\\
(\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})\,\underline{t} &:= \lambda\,f\,i.\,\boldsymbol{b}\,i\,\underline{t}\,(f\,i)
\end{aligned}$$

We rely on Π in the inner theory. The preservation isomorphisms are pointwise inherited from $\boldsymbol{b}$. One direction of the isomorphisms is defined as follows. Note that $\bullet_{\mathbf{U}} \equiv \top$ and $\triangleright_{\mathbf{U}}$ is Σ.

$$\begin{aligned}
(\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})^{-1}_{\bullet} &: \top \to (\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})\,\bullet\\
(\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})^{-1}_{\bullet} &:\equiv \lambda\,_\,i.\,(\boldsymbol{b}\,i)^{-1}_{\bullet}\,\mathsf{tt}
\end{aligned}$$

$$\begin{aligned}
(\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})^{-1}_{\triangleright} &: (\Gamma : (\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})\,\underline{\Gamma}) \times ((\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})\,\underline{A}\,\Gamma)\\
&\qquad \to (\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})\,(\underline{\Gamma} \triangleright \underline{A})\\
(\mathbf{\Pi^{inf}}\,Ix\,\boldsymbol{b})^{-1}_{\triangleright} &:\equiv \lambda\,(\Gamma, A)\,i.\,(\boldsymbol{b}\,i)^{-1}_{\triangleright}(\Gamma\,i,\ A\,i)
\end{aligned}$$

Inductive function type

For $\boldsymbol{a} : \mathbf{Tm}\,\boldsymbol{\Gamma}\,\mathbf{U}$ and $\boldsymbol{B} : \mathbf{Ty}\,(\boldsymbol{\Gamma} \triangleright \mathbf{El}\,\boldsymbol{a})$, we aim to define $\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B} : \mathbf{Ty}\,\boldsymbol{\Gamma}$. The underlying sets are unchanged.

$$\begin{aligned}
\mathsf{Con}_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\underline{\Gamma} &:\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Con}_{\boldsymbol{B}}\,(\underline{\Gamma},\,\gamma)\\
\mathsf{Sub}_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Sub}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\Delta\,(\boldsymbol{a}\,\underline{\sigma}\,\gamma))\,(\underline{\sigma},\,\mathsf{refl})\\
\mathsf{Ty}_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,\underline{A} &:\equiv \{\gamma : \boldsymbol{a}\,\underline{\Gamma}\}(\alpha : \boldsymbol{a}\,\underline{A}\,\gamma) \to \mathsf{Ty}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\underline{A},\,\alpha)\\
\mathsf{Tm}_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}}\,\Gamma\,A\,\underline{t} &:\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to \mathsf{Tm}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\mathsf{A}\,(\boldsymbol{a}\,\underline{t}\,\gamma))\,(\underline{t},\,\mathsf{refl})
\end{aligned}$$

Likewise, all structure is defined pointwise using $\boldsymbol{B}$ structure. Similarly to the **El** case, we have to sometimes fall through the defining isomorphisms for $\boldsymbol{a}$ structure. For comparison, in the finitary case we had the following definition:

$$(\Gamma \triangleright_{\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}} A)\,(\gamma,\,\alpha) :\equiv (\Gamma\,\gamma \triangleright_{\boldsymbol{B}} A\,\alpha)$$

Here, $(\gamma,\,\alpha) : \boldsymbol{a}\,(\underline{\Gamma} \triangleright \underline{A})$, so also $(\gamma,\,\alpha) : (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \times \boldsymbol{a}\,\underline{A}\,\gamma$, so the Σ pattern-matching notation is justified in the definition. In the current infinitary case, we

have $(\boldsymbol{a}_{\triangleright}, \boldsymbol{a}_{\triangleright}^{-1}) : \boldsymbol{a}\,(\underline{\Gamma} \triangleright \underline{A}) \simeq ((\gamma : \boldsymbol{a}\,\underline{\Gamma}) \times \boldsymbol{a}\,\underline{A}\,\gamma)$ instead. But we can use the intuition that set isomorphisms are like unary record types, so we can still give a pattern-matching definition:

$$(\Gamma \triangleright_{\boldsymbol{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}} A)\,(\boldsymbol{a}_{\triangleright}^{-1}(\gamma, \alpha)) :\equiv (\Gamma\,\gamma \triangleright_{\boldsymbol{B}}\, A\,\alpha)$$

For the definitions of other type and term formers, we likewise insert the isomorphisms appropriately. It remains to define iso-cleaving $\boldsymbol{\Pi}$. Coercion is given by mapping indices backwards in $\mathsf{El}\,\boldsymbol{a}$ and coercing outputs forwards in $\boldsymbol{B}$.

$$\mathsf{coe}\,\underline{\sigma}\,\Gamma :\equiv \lambda\,\gamma.\,\mathsf{coe}_{\boldsymbol{B}}\,(\underline{\sigma}, \mathsf{refl})\,(\Gamma\,(\boldsymbol{a}\,(\underline{\sigma}^{-1})\,\gamma))$$
$$\mathsf{coe}\,\underline{t}\,A :\equiv \lambda\,\gamma\,a.\,\mathsf{coe}_{\boldsymbol{B}}\,(\underline{t}, \mathsf{refl})\,(A\,(\boldsymbol{a}\,(\underline{t}^{-1})\,(\boldsymbol{a}_{\triangleright}^{-1}(\gamma, a))))$$

Likewise, coh-s are given by backwards-forwards coh-s. As before, **app** and **lam** are defined as currying and uncurrying the underlying functions.

External function type

For $Ix : \mathsf{Set}_j$ and $\boldsymbol{B} : Ix \to \mathbf{Ty}\,\boldsymbol{\Gamma}$, we define $\boldsymbol{\Pi}^{\mathbf{ext}}\,Ix\,\boldsymbol{B} : \mathbf{Ty}\,\boldsymbol{\Gamma}$ as the Ix-indexed direct product of $\boldsymbol{B}$. Since the indexing is given by a metatheoretic function, every component is given in the evident pointwise way, including iso-cleaving. This concludes the definition of the $\mathbf{M}$ model.

5.3 Left Adjoints of Substitutions

In the following we adapt Section 4.3 to infinitary signatures.

- We again write $[\![-]\!]$ for the interpretation into the flcwf model $\mathbf{M}$.
- We also add $\top : \mathsf{Ty}\,\Gamma$ and $\Sigma : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$ to the ToS. Again we do not elaborate much on their semantics; $\top$ is given as $\mathbf{K}\,\bullet$ in the model and Σ is given by component-wise Σ.

We again fix $\Omega : \mathsf{Con}$ and define heterogeneous morphisms. The types of eliminators stay exactly the same:

$$\_^{HM} : (\Gamma : \mathsf{Con}) \to \Gamma^A \to \mathsf{Sub}\,\Omega\,\Gamma \to \mathsf{Ty}\,\Omega$$
$$\_^{HM} : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Omega\,(\Gamma^{HM}\,\gamma_0\,\gamma_1) \to \mathsf{Tm}\,\Omega\,(\Delta^{HM}\,(\sigma^A\,\gamma_0)\,(\sigma \circ \gamma_1))$$
$$\_^{HM} : (A : \mathsf{Ty}\,\Gamma) \to A^A\,\gamma_0 \to \mathsf{Tm}\,\Omega\,(A[\gamma_1]) \to \mathsf{Tm}\,\Omega\,(\Gamma^{HM}\,\gamma_0\,\gamma_1) \to \mathsf{Ty}\,\Omega$$
$$\_^{HM} : (t : \mathsf{Tm}\,\Gamma\,A)(\gamma^{HM} : \mathsf{Tm}\,\Omega\,(\Gamma^{HM}\,\gamma_0\,\gamma_1)) \to \mathsf{Tm}\,\Omega\,(A^{HM}\,(t^A\,\gamma_0)\,(t[\gamma_1])\,\gamma^{HM})$$

We only need to show that the new type formers in U, namely $\top$, Σ, Id and Π^{inf}, can be also covered in the definition of $-^{HM}$. The new type formers turn out to work exactly as mechanically as the previous ones. We have the following:

$$\top^{HM}\,\gamma^{HM} : \mathsf{Tm}\,\Omega\,(\top \to^{\mathsf{ext}} \mathsf{El}\,\top)$$
$$\top^{HM}\,\gamma^{HM} :\equiv \lambda^{\mathsf{ext}}\,\_.\,\mathsf{tt}$$

$$(\Sigma\,a\,b)^{HM}\,\gamma^{HM} :$$
$$\quad \mathsf{Tm}\,\Omega\,(((\alpha : a^A\,\gamma_0) \times (b^A\,(\gamma_0,\,\alpha))) \to^{\mathsf{ext}} \mathsf{El}\,(\Sigma\,(\alpha : a[\gamma_1])\,(b[\gamma_1,\,\alpha])))$$
$$(\Sigma\,a\,b)^{HM}\,\gamma^{HM} :\equiv \lambda^{\mathsf{ext}}\,(\alpha,\,\beta).\,(a^{HM}\,\gamma^{HM}\,\alpha,\,b^{HM}\,(\gamma^{HM},\,\mathsf{refl})\,\beta)$$

$$(\Pi^{\mathsf{inf}}\,\mathsf{Ix}\,b)^{HM}\,\gamma^{HM} : \mathsf{Tm}\,\Omega\,(((i : \mathsf{Ix}) \to (b\,i)^A\,\gamma_0) \to^{\mathsf{ext}} \mathsf{El}\,((i : \mathsf{Ix}) \to^{\mathsf{inf}} (b\,i)[\gamma_1]))$$
$$(\Pi^{\mathsf{inf}}\,\mathsf{Ix}\,b)^{HM}\,\gamma^{HM} :\equiv \lambda^{\mathsf{ext}}\,f.\,\lambda^{\mathsf{inf}}\,i.\,(f\,i)^{HM}\,\gamma^{HM}$$

For Id, we again have to complete a square.

$$(\mathsf{Id}\,t\,u)\,\gamma^{HM} : \mathsf{Tm}\,\Omega\,((t^A\,\gamma_0 = u^A\,\gamma_0) \to^{\mathsf{ext}} \mathsf{El}\,(\mathsf{Id}\,(t[\gamma_1])\,(u[\gamma_1])))$$

This follows from $t^{HM}\,\gamma^{HM}$ and $u^{HM}\,\gamma^{HM}$, the same way as in the flcwf semantics before.

Theorem 9. *If every infinitary QII signature has an initial algebra, then for every $\nu : \mathsf{Sub}\,\Omega\,\Delta$, there exists a left adjoint of $[\![\nu]\!] : [\![\Omega]\!] \to [\![\Delta]\!]$.*

Proof. For each $\delta : \Delta^A$, the comma category $\delta/[\![\nu]\!]$ can be specified (up to isomorphism) by the signature $\Omega \triangleright \Delta^{HM}\,\delta\,\nu$, thus it has an initial object. Hence $[\![\nu]\!]$ has a left adjoint. □

5.4 Signature-Based Semantics of Signatures

We have seen that the $-^{HM}$ interpretation yields a notion of algebra morphism that is specified inside ToS. What else can we represent in ToS? For example, can we internalize $-^D$, $-^M$ and $-^S$? In this section we show that the full flcwf semantics can be expressed internally to the ToS syntax.

This means that for each $\Gamma : \mathsf{Con}$, we get $\Gamma^A : \mathsf{Ty}\,\bullet$ as the notion of algebras, $\Gamma^M : \mathsf{Ty}\,(\bullet \triangleright (\gamma_0 : \Gamma^A) \triangleright (\gamma_1 : \Gamma^A))$ as the notion of morphisms, $\mathsf{id} : \mathsf{Tm}\,(\bullet \triangleright (\gamma :$

$\Gamma^A))\,(\Gamma^M[\gamma_0 \mapsto \gamma,\, \gamma_1 \mapsto \gamma])$ for the identity morphisms, and likewise we get the whole flcwf of algebras in such an internal manner.

As we will shortly see, capturing the full flcwf semantics is possible with the infinitary ToS, but not with the finitary ToS, because it lacks the necessary type formers in U.

It would be needlessly tedious and repetitive to redo the flcwf semantics while explicitly working with ToS components. Instead, we repurpose 2LTT for this use case. Recall that 2LTT allows to get semantics internally to any cwf with Π, Σ, $\top$ and Id. In the current section we aim to get semantics internally to the ToS syntax. In short, this means that we work in a 2LTT where the inner theory is the theory of signatures. The picture is a bit more nuanced though.

First, since ToS lives inside 2LTT, and we want to get presheaves over ToS in the presheaf model, the metatheoretic setting of the presheaf model must be also a 2LTT. This might get a bit confusing, so let us expand:

- The syntax of 2LTT internalizes the ToS syntax as an assumed type former.
- The presheaf model of 2LTT lives inside yet another 2LTT, let us call it 2LTT*, which embeds *both* the 2LTT syntax and the ToS syntax separately.
- In the presheaf model, the base cwf is the cwf of the ToS syntax in 2LTT*.
- The Ty_0 type former in 2LTT is interpreted in the presheaf model using the Ty_0 type former in 2LTT*.
- We add $\mathsf{Ty}_{\mathsf{sig}} : \mathsf{Set}$ and $\mathsf{Tm}_{\mathsf{sig}} : \mathsf{Ty}_{\mathsf{sig}} \to \mathsf{Set}$ to 2LTT. $\mathsf{Ty}_{\mathsf{sig}}$ is interpreted as the presheaf of ToS types, and $\mathsf{Tm}_{\mathsf{sig}}$ is interpreted using the displayed presheaf of ToS terms, following Definition 36.
- We close $\mathsf{Ty}_{\mathsf{sig}}$ under type formers which represent all type formers in ToS. Like in the previous section, we assume that ToS types are closed under $\top$ and Σ, so we have $\top$, Σ, inductive Π, Π^{ext} and U in $\mathsf{Ty}_{\mathsf{sig}}$. The U in $\mathsf{Ty}_{\mathsf{sig}}$ has $\mathsf{El} : \mathsf{Tm}_{\mathsf{sig}}\,\mathsf{U} \to \mathsf{Ty}_{\mathsf{sig}}$, and it is closed under $\top$, Σ, Π^{inf} and Id. In the presheaf model, all structure in $\mathsf{Ty}_{\mathsf{sig}}$ is interpreted using ToS type formers in the evident way.

Notation 20. We shall omit $\mathsf{Tm}_{\mathsf{sig}}$ in the following, similarly to how we previously omitted Tm_0. We keep omitting Tm_0 in the new setup as well. However, we will still mark $\mathsf{El} : \mathsf{U} \to \mathsf{Ty}_{\mathsf{sig}}$ explicitly.

For reference, we list type formers in $\mathsf{Ty}_{\mathsf{sig}}$ below.

$$
\begin{aligned}
&\mathsf{U} && : \mathsf{Ty}_{\mathsf{sig}}\\
&\mathsf{El} && : \mathsf{U} \to \mathsf{Ty}_{\mathsf{sig}}\\
&\top && : \mathsf{U}\\
&\Sigma && : (a : \mathsf{U}) \to (\mathsf{El}\, a \to \mathsf{U}) \to \mathsf{U}\\
&\mathsf{Id} && : \mathsf{El}\, a \to \mathsf{El}\, a \to \mathsf{U}\\
&\Pi^{\mathsf{inf}} && : (\mathsf{Ix} : \mathsf{Ty}_0) \to (\mathsf{Ix} \to \mathsf{U}) \to \mathsf{U}\\
&\Pi && : (a : \mathsf{U}) \to (\mathsf{El}\, a \to \mathsf{Ty}_{\mathsf{sig}}) \to \mathsf{Ty}_{\mathsf{sig}}\\
&\Pi^{\mathsf{ext}} && : (\mathsf{Ix} : \mathsf{Ty}_0) \to (\mathsf{Ix} \to \mathsf{Ty}_{\mathsf{sig}}) \to \mathsf{Ty}_{\mathsf{sig}}\\
&\Sigma && : (A : \mathsf{Ty}_{\mathsf{sig}}) \to (A \to \mathsf{Ty}_{\mathsf{sig}}) \to \mathsf{Ty}_{\mathsf{sig}}\\
&\top && : \mathsf{Ty}_{\mathsf{sig}}
\end{aligned}
$$

Notation 21. We will use the $- \to^{\mathsf{ext}} -$ and $- \to^{\mathsf{inf}} -$ notations in the following for Π^{inf} and Π^{ext}, but additionally we use $- \to^{\mathsf{ind}} -$ for inductive functions, to disambiguate them from outer functions in 2LTT.

We revisit now the flcwf semantics in the new setting. The goal is to produce output by the signature-based semantics, such that if we use the original $-^A$ interpretation on that, we get results that are equivalent to what we get from the direct semantics. For the simplest example, for $\Gamma : \mathsf{Con}$, we get $\Gamma^A_{\mathsf{sig}} : \mathsf{Ty}\, \bullet$ from the signature-based semantics, then we get $(\Gamma^A_{\mathsf{sig}})^A\, \mathsf{tt} : \mathsf{Set}$, which should be equivalent to $\Gamma^A : \mathsf{Set}$.

In this section, we only describe the signature-based semantics, and we do not formally check the round-trip property. The round-tripping seems very plausible though, since as we will see, the signature-based semantics is exactly the same as the direct semantics, modulo the change of universes and type formers.

We look at key parts of the model. In each case, we generally only check that we have sufficient type formers. We again write components of the model in **bold** font.

Base cwf

Contexts in the model are still flcwfs, but now Con, Sub, Ty and Tm in flcwfs all return in $\mathsf{Ty}_{\mathsf{sig}}$. Hence, assuming $\boldsymbol{\Gamma} : \mathbf{Con}$, we have

$$\begin{aligned}
&\mathsf{Con}_{\boldsymbol{\Gamma}} : \mathsf{Ty}_{\mathsf{sig}}\\
&\mathsf{Sub}_{\boldsymbol{\Gamma}} : \mathsf{Con}_{\boldsymbol{\Gamma}} \to \mathsf{Con}_{\boldsymbol{\Gamma}} \to \mathsf{Ty}_{\mathsf{sig}}\\
&\mathsf{Ty}_{\boldsymbol{\Gamma}} \ : \mathsf{Con}_{\boldsymbol{\Gamma}} \to \mathsf{Ty}_{\mathsf{sig}}\\
&\mathsf{Tm}_{\boldsymbol{\Gamma}} \ : (\Gamma : \mathsf{Con}_{\boldsymbol{\Gamma}}) \to \mathsf{Ty}_{\boldsymbol{\Gamma}}\,\Gamma \to \mathsf{Ty}_{\mathsf{sig}}
\end{aligned}$$

We specify all equations using outer equality (since the Id types in $\mathsf{Ty}_{\mathsf{sig}}$ are extensional, this makes no difference). Similarly, components of $\boldsymbol{A} : \mathbf{Ty}\,\boldsymbol{\Gamma}$ return in $\mathsf{Ty}_{\mathsf{sig}}$. Substitutions and terms in the model are unchanged, they are weak morphisms and sections respectively. For $\bullet : \mathbf{Con}$, we use $\top : \mathsf{Ty}_{\mathsf{sig}}$ to define the components. Likewise, we use the Σ type in $\mathsf{Ty}_{\mathsf{sig}}$ to define $- \triangleright -$.

If we write $\mathsf{Tm}_{\mathsf{sig}}$ explicitly, we have e.g. $\mathsf{Sub}_{\boldsymbol{\Gamma}} : \mathsf{Tm}_{\mathsf{sig}}\,\mathsf{Con}_{\boldsymbol{\Gamma}} \to \mathsf{Tm}_{\mathsf{sig}}\,\mathsf{Con}_{\boldsymbol{\Gamma}} \to \mathsf{Ty}_{\mathsf{sig}}$. Thus, we may use the simplified interpretation of functions with inner domains, from Section 3.4.3, and if we interpret the type of $\mathsf{Sub}_{\boldsymbol{\Gamma}}$ at the empty context in the presheaf model, we get $\mathsf{Ty}\,(\bullet \triangleright |\mathsf{Con}_{\boldsymbol{\Gamma}}|\,\mathsf{tt} \triangleright |\mathsf{Con}_{\boldsymbol{\Gamma}}|\,\mathsf{tt})$.

Universe

$\mathbf{U} : \mathbf{Ty}\,\boldsymbol{\Gamma}$ is defined as $\mathbf{U} : \mathbf{Con}$, and we take the constant displayed flcwf of the definition. Now, we have $\mathbf{U} : \mathbf{Con}$ as the flcwf of types in $\mathsf{U} : \mathsf{Ty}_{\mathsf{sig}}$.

$$\begin{aligned}
\mathsf{Con}_{\mathbf{U}} \quad &:\equiv \mathsf{U}\\
\mathsf{Sub}_{\mathbf{U}}\,\Gamma\,\Delta &:\equiv \Gamma \to^{\mathsf{ind}} \mathsf{El}\,\Delta\\
\mathsf{Ty}_{\mathbf{U}}\,\Gamma \quad &:\equiv \Gamma \to^{\mathsf{ind}} \mathsf{U}\\
\mathsf{Tm}_{\mathbf{U}}\,\Gamma\,A &:\equiv (\gamma : \Gamma) \to^{\mathsf{ind}} \mathsf{El}\,(A\,\gamma)
\end{aligned}$$

$\bullet_{\mathbf{U}}$, $- \triangleright_{\mathbf{U}} -$ and $\mathsf{Id}_{\mathbf{U}}$ are defined using the type formers in U. As before, $\mathsf{K}_{\mathbf{U}}$ is defined simply as a constant function. In $\mathbf{El}\,\boldsymbol{a} : \mathbf{Ty}\,\boldsymbol{\Gamma}$, we use the Id type in Tys in morphisms and sections:

$$\begin{aligned}
\mathsf{Con}_{\mathbf{El}\,\boldsymbol{a}}\,\underline{\Gamma} \quad &:\equiv \mathsf{El}\,(\boldsymbol{a}\,\underline{\Gamma})\\
\mathsf{Sub}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,\Delta\,\underline{\sigma} &:\equiv \mathsf{Id}\,(\boldsymbol{a}\,\underline{\sigma}\,\Gamma)\,\Delta\\
\mathsf{Ty}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,\underline{A} \quad &:\equiv \mathsf{El}\,(\boldsymbol{a}\,\underline{A}\,\Gamma)\\
\mathsf{Tm}_{\mathbf{El}\,\boldsymbol{a}}\,\Gamma\,A\,\underline{t} \ &:\equiv \mathsf{Id}\,(\boldsymbol{a}\,\underline{t},\Gamma)\,A
\end{aligned}$$

Type formers in U

For $\top$, $\mathbf{\Sigma}$ and $\mathbf{Id}$ in $\mathbf{U}$, we use $\top$, Σ and Id in $\mathsf{U} : \mathsf{Ty}_{\mathsf{sig}}$ in a straightforward way. For $\mathbf{\Pi}^{\mathbf{inf}}\,\mathsf{Ix}\,\boldsymbol{b}$, we have the following:

$$\mathsf{Con}_{(\mathbf{\Pi}^{\mathbf{inf}}\,\mathsf{Ix}\,\boldsymbol{b})}\,\underline{\Gamma} :\equiv (i : \mathsf{Ix}) \to^{\mathsf{inf}} (\boldsymbol{b}\,i)\,\underline{\Gamma}$$

Let us look at morphisms:

$$\mathsf{Sub}_{(\mathbf{\Pi}^{\mathbf{inf}}\,\mathsf{Ix}\,\boldsymbol{b})}\,\underline{\sigma} : ((i : \mathsf{Ix}) \to^{\mathsf{inf}} (\boldsymbol{b}\,i)\,\underline{\Gamma}) \to^{\mathsf{ind}} \mathsf{El}\,((i : \mathsf{Ix}) \to^{\mathsf{inf}} (\boldsymbol{b}\,i)\,\underline{\Delta})$$
$$\mathsf{Sub}_{(\mathbf{\Pi}^{\mathbf{inf}}\,\mathsf{Ix}\,\boldsymbol{b})}\,\underline{\sigma} :\equiv \lambda^{\mathsf{ind}}\,t.\,\lambda^{\mathsf{inf}}\,i.\,(\boldsymbol{b}\,i)\,\underline{\sigma}\,(t\,i)$$

Here, we map an infinitary function to another one, which checks out just fine, since $\to^{\mathsf{ind}}$ allows such mapping. We have just enough higher-order functions to complete this definition. The rest of $\mathbf{\Pi}^{\mathbf{inf}}\,\mathsf{Ix}\,\boldsymbol{b}$ follows evidently.

$\mathbf{\Pi}$, $\mathbf{\Pi}^{\mathbf{ext}}$, $\top$, $\mathbf{\Sigma}$

In $\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B}$, we use inductive functions in components:

$$\mathsf{Con}_{(\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B})}\,\underline{\Gamma} \quad :\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to^{\mathsf{ind}} \mathsf{Con}_{\boldsymbol{B}}\,(\underline{\Gamma},\,\gamma)$$
$$\mathsf{Sub}_{(\mathbf{\Pi}\,\boldsymbol{a}\,\boldsymbol{B})}\,\Gamma\,\Delta\,\underline{\sigma} :\equiv (\gamma : \boldsymbol{a}\,\underline{\Gamma}) \to^{\mathsf{ind}} \mathsf{Sub}_{\boldsymbol{B}}\,(\Gamma\,\gamma)\,(\Delta\,(\boldsymbol{a}\,\underline{\sigma}\,\gamma))\,(\underline{\sigma},\,\mathsf{refl})$$
$$\ldots$$

In $\mathbf{\Pi}^{\mathbf{ext}}$, we use $\to^{\mathsf{ext}}$. In $\top$ and $\mathbf{\Sigma}$, we use $\top$ and Σ in $\mathsf{Ty}_{\mathsf{sig}}$. This concludes the definition of the signature-based semantics.

Definition 65 (Signature-based AMDS interpretation). For some $\Gamma : \mathsf{Con}$, we define the following by interpreting Γ in the signature-based flcwf model, then interpreting the result in the presheaf model of 2LTT.

$$\Gamma^{A}_{\mathsf{sig}} : \mathsf{Ty}\,\bullet$$
$$\Gamma^{M}_{\mathsf{sig}} : \mathsf{Ty}\,(\bullet \triangleright (\gamma_0 : \Gamma^{A}_{\mathsf{sig}}) \triangleright (\gamma_1 : \Gamma^{A}_{\mathsf{sig}}))$$
$$\Gamma^{D}_{\mathsf{sig}} : \mathsf{Ty}\,(\bullet \triangleright (\gamma : \Gamma^{A}_{\mathsf{sig}}))$$
$$\Gamma^{S}_{\mathsf{sig}} : \mathsf{Ty}\,(\bullet \triangleright (\gamma : \Gamma^{A}_{\mathsf{sig}}) \triangleright (\gamma^{D} : \Gamma^{D}_{\mathsf{sig}}))$$

Backporting to finitary signatures

It is apparent from the previous section that the signature-based full flcwf model requires at least $\top$, Σ and Id in U: in the definition of $\mathbf{U}$ in the model these are needed to define the family structure and the finite limit structure.

Hence, if we want to only support structure in $\mathsf{Ty}_{\mathsf{sig}}$ corresponding to a theory of finitary signatures, we need to drop all semantic components which rely on the missing type formers. We have seen this kind of trimmed semantics in Section 4.4.2. In particular, we still get a category of algebras for each signature, since that can be modeled without $\top$, Σ and Id.

Application: colimits

The signature-based semantics is often helpful when we want to construct new signatures from old ones. We give an example application, in the construction of colimits.

We would like to use left adjoints of substitutions to build colimits in categories of algebras. For this, it enough to build indexed coproducts and binary coequalizers.

For some $\Gamma : \mathsf{Con}$, we get $\Gamma^A_{\mathsf{tos}} : \mathsf{Ty}\,\bullet$. For convenience we shall work with Γ^A_{tos} in the following, instead of Γ. First, we construct Ix-indexed coproducts in the category of Γ-algebras, by taking the left adjoint of the following diagonal substitution:

$$\begin{aligned}&\mathsf{diag} : \mathsf{Sub}\,(\bullet \triangleright (\gamma : \Gamma^A_{\mathsf{tos}}))\,(\bullet \triangleright (f : (\mathsf{Ix} \to^{\mathsf{ext}} \Gamma^A_{\mathsf{tos}})))\\ &\mathsf{diag} :\equiv (f \mapsto \lambda^{\mathsf{ext}}\, i.\, \gamma)\end{aligned}$$

For coequalizers,we again take the left adjoint of a diagonal substitution, but here we need to rely on internal morphisms in the signature:

$$\begin{aligned}&\mathsf{diag} : \mathsf{Sub}\,(\bullet \triangleright (\gamma : \Gamma^A_{\mathsf{tos}}))\\ &\qquad(\bullet \triangleright (\gamma_0 : \Gamma^A_{\mathsf{tos}}) \triangleright (\gamma_1 : \Gamma^A_{\mathsf{tos}})\\ &\qquad\quad \triangleright (f : \Gamma^M_{\mathsf{tos}}[\gamma_0 \mapsto \gamma_0,\, \gamma_1 \mapsto \gamma_1])\\ &\qquad\quad \triangleright (g : \Gamma^M_{\mathsf{tos}}[\gamma_0 \mapsto \gamma_1,\, \gamma_1 \mapsto \gamma_0]))\\ &\mathsf{diag} :\equiv (\gamma_0 \mapsto \gamma,\, \gamma_1 \mapsto \gamma,\, f \mapsto \mathsf{id}_{\mathsf{tos}}[\gamma \mapsto \gamma],\, g \mapsto \mathsf{id}_{\mathsf{tos}}[\gamma \mapsto \gamma])\end{aligned}$$

Above, we use $\mathsf{id}_{\mathsf{tos}} : \mathsf{Tm}\,(\bullet \triangleright (\gamma : \Gamma^A_{\mathsf{tos}}))\,(\Gamma^M_{\mathsf{tos}}\,[\gamma_0 \mapsto \gamma,\, \gamma_1 \mapsto \gamma])$, which also comes from the signature-based semantics.

Of course, if we want to be fully precise, we need to show that what we get is equivalent to coproducts and coequalizers in the external sense. For this, we would need the round-trip property of the signature-based semantics.

5.5 Discussion of Semantics

Iso-fibrancy as a structure identity principle

The flcwfs of algebras that we get from the infinitary semantics are exactly the same as in the finitary case. However, semantic types are a bit more interesting. The iso-fibrancy of types can be understood as a weaker version of the *structure identity principle* in homotopy type theory.

The structure identity principle says that isomorphism of algebras is equivalent to equality of algebras. This is the same as saying that categories of algebras are univalent [AKS15]. Assuming a signature Γ and algebras $\gamma \simeq \gamma'$, we have $\gamma = \gamma'$. This equality is respected by every construction in HoTT, which implies that for any HoTT type family $F : \Gamma^A \to \mathsf{Type}$, we have a function $F\,\gamma \to F\,\gamma'$.

We get a similar but weaker statement from the infinitary semantics: for $\sigma : \gamma \simeq \gamma'$ and some ToS type $A : \mathsf{Ty}\,\Gamma$, we have a function $\mathsf{coe}\,\sigma : A^A\,\gamma \to A^A\,\gamma'$. We also have $\mathsf{coh}\,\sigma\,\alpha : \alpha \simeq_\sigma \mathsf{coe}\,\sigma\,\alpha$ for some $\alpha : A^A\,\gamma$. So we can transport over isomorphisms, but not all metatheoretic families can be transported, only those which arise as ToS types.

Of course, we can transport over multiple types, or telescopes of types too, by iterated transport. For instance, given $A : \mathsf{Ty}\,\Gamma$, $B : \mathsf{Ty}\,(\Gamma \triangleright A)$, $\alpha : A^A\,\gamma$ and $\beta : B^A\,(\gamma, \alpha)$, we can transport α first, then transport β by $(\sigma, \mathsf{coh}\,\sigma\,\alpha)$. Alternatively, if have large Σ types in ToS, as $\Sigma : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\Gamma$, that makes iterated transport superfluous.

Variations of semantics

First, unlike in the finitary case, we have no opportunity to minimize assumptions on the inner theory. Already when we compute algebras, we need inner Π for infinitary functions, inner $\top$ for $\top$, inner Σ for Σ and inner $- = -$ for Id. Note though that we still get semantics in any LCCC (locally cartesian closed category), since we can build a cwf with the required type formers from any LCCC [CD14].

Second: can we add the "large" equality type, which includes sort equations, back to infinitary signatures? We dropped sort equations in this chapter because they are clearly not isofibrant. We can add them back into the mix though, at the price of dropping components from the semantics of signatures. The reason for having isofibrant types is that type formers in U preserve $\bullet$ and $-\triangleright-$ only up

to isomorphism. If we drop all semantic components which depend on $\bullet$ and $-\triangleright-$, we can drop isofibrancy too from the model, and everything works. In this case, we still get a category of algebras, plus a notion of induction, but we cannot show that initiality is equivalent to induction, as the proof of Theorem 1 depends on $-\triangleright-$.

Model Constructions

In this chapter we gain some expressive power in defining model constructions using substitutions or terms. For starters, the construction of categories from monoids works now:

Example 30. Let us have $\mathsf{MonoidSig}$ as the signature for monoids, with $\mathsf{M} : \mathsf{U}$ as the carrier set, $-\cdot- : \mathsf{M} \to \mathsf{M} \to \mathsf{El}\,\mathsf{M}$ as multiplication and $\epsilon : \mathsf{El}\,\mathsf{M}$ as identity the element. We define $\sigma : \mathsf{Sub}\,\mathsf{MonoidSig}\,\mathsf{CatSig}$ to contain $\mathsf{Obj} :\equiv \top$, $\mathsf{Hom} :\equiv \lambda \_\,\_.\,\mathsf{M}$, $\mathsf{id} :\equiv \epsilon$ and $-\circ- :\equiv -\cdot-$.

Many constructions in the literature which have been dubbed *syntactic models* [BPT17] or *syntactic translations* can be defined now in the ToS, for the following reasons.

- Syntactic translations usually do not rely on models being actually syntactic: they do not use induction on *target* theory syntax. A rare counterexample is our construction of recursors and eliminators for term models. These are perhaps syntactic in the sense that they prominently involve the syntax of some type theory, and they construct recursor/eliminator functions by induction on terms.
- Syntactic translations rarely if ever involve higher-order constructions. Such would be interpreting Con with $(\mathsf{Con} \to \mathsf{Con}) \to \mathsf{Con}$, for a contrived example.

The gluing construction in Example 21 is already a fairly general example that only requires the finitary ToS to define. That construction is more in an "indexed" style, but now we can also do constructions in a more "fibered" style.

Example 31. We may consider a unary parametricity translation in the style of Bernardy, Jansson, and Paterson [BJP10], which makes use of the small Σ-type in the theory of signature. We assume $\mathsf{TT} : \mathsf{Ty}\,\bullet$ as the signature for the theory, and

$\mathsf{TT}^D : \mathsf{Ty}\,(\bullet \triangleright (M : \mathsf{TT}))$ as the signature for displayed models. The translation can be typed as $\mathsf{Tm}\,(\bullet \triangleright (M : \mathsf{TT}))\,\mathsf{TT}^D$: we assume a model of the theory, and build a displayed model over the same theory. Informally, when M is initial, we get a translation which doubles each context:

$$[\![\Gamma \triangleright (a : A)]\!] \equiv [\![\Gamma]\!] \triangleright (a : A) \triangleright (a^D : [\![A]\!]\,a)$$

Formally, however, this is not well-typed because A lives in Γ, not in $[\![\Gamma]\!]$. Hence, in the definition of contexts in the displayed model, we also include a substitution which projects out the "base" parts of contexts. This can be used to weaken types in base contexts to total contexts.

$$\begin{aligned}&\mathsf{Con} : \mathsf{Con}_M \to \mathsf{U}\\ &\mathsf{Con}\,\underline{\Gamma} :\equiv \Sigma\,(\underline{\Gamma}' : \mathsf{Con}_M)\,(\mathsf{proj} : \mathsf{Sub}_M\,\underline{\Gamma}'\,\underline{\Gamma})\end{aligned}$$

This requires the small Σ-type in ToS. It is possible to rephrase the construction without type formers in U; again, Example 21 has unary parametricity as a special case. However, the fibered version has the advantage that contexts are translated to contexts, types to types, and terms to terms, which makes it more convenient if we actually want to implement it as program translation. In contrast, the gluing definition of unary parametricity maps contexts to types.

5.6 Term Algebras

We adapt now the previous term algebra construction to the infinitary case. We again switch to the ETT setup with cumulative universes. We assume Section 4.5.1 without any change. Also, we adapt 4.5.2 to infinitary signatures and semantics. All definitions are the same, the only change is that the $\mathbf{M}_{i,j}$ model is now the isofibrant flcwf model, and we have the infinitary ToS.

5.6.1 Term Algebra Construction

The term algebra construction changes significantly. The reason is the following. In the finitary case, the key property was that "small types evaluated in the term model are sets of terms". Formally, we had for $a : \mathsf{Tm}\,\Omega\,\mathsf{U}$ that $a^A\,(\Omega^T\,\mathsf{id}) \equiv \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a)$. This is now weakened to an isomorphism, i.e. $a^A\,(\Omega^T\,\mathsf{id}) \simeq \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a)$.

This is again necessary because of the closure of U under type formers. For example, $\top^A\,(\Omega^T\,\mathsf{id}) \equiv \top$, and $\mathsf{Tm}\,\Omega\,(\mathsf{El}\,\top)$ is merely isomorphic to $\top$. We assume $\Omega : \mathsf{Sig}_j$ for some j level, and define $-^T$ by induction on syn_j.

$$
\begin{aligned}
&-^T : (\Gamma : \mathsf{Con}) \qquad (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^A \\
&-^T : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta)(\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^T\,(\sigma \circ \nu) \simeq \sigma^A\,(\Gamma^T\,\nu) \\
&-^T : (A : \mathsf{Ty}\,\Gamma) \qquad (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,(A[\nu]) \to A^A\,(\Gamma^T\,\nu) \\
&-^T : (t : \mathsf{Tm}\,\Gamma\,A)\ (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^T\,\nu\,(t[\nu]) \simeq_{\mathsf{id}} t^A\,(\Gamma^T\,\nu)
\end{aligned}
$$

In short, interpretations of substitutions and terms are weakened to isomorphisms. By $\simeq_{\mathsf{id}}$ we mean a displayed isomorphism of objects in the semantic A type (which is an flcwf isofibration); recall Definition 60. The isomorphism is "vertical" since it lies over id.

The interpretation of the cwf is the same as before, but like in the isofibrant semantics, we have to use explicit coe instead of silently transporting over equalities. In the interpretations of substitutions and terms, we have to explicitly compose isomorphisms and sometimes lift them using coh. We give some examples. The interpretation of context formers is the same as before:

$$
\begin{aligned}
&\bullet^T\,\nu &&:\equiv \mathsf{tt} \\
&(\Gamma \triangleright A)^T(\nu,\, t) &&:\equiv (\Gamma^T\,\nu,\, A^T\,\nu\,t)
\end{aligned}
$$

Type substitution with $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$ is interpreted as coercion:

$$
\begin{aligned}
&(A[\sigma])^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(A[\sigma][\nu]) \to A^A\,(\sigma^A\,(\Gamma^T\,\nu)) \\
&(A[\sigma])^T\,\nu\,t :\equiv \mathsf{coe}\,(\sigma^T\,\nu)\,(A^T\,(\sigma \circ \nu)\,t)
\end{aligned}
$$

Composition of $\sigma : \mathsf{Sub}\,\Delta\,\Xi$ and $\delta : \mathsf{Sub}\,\Gamma\,\Delta$ is the following:

$$
\begin{aligned}
&(\sigma \circ \delta)^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^T\,(\sigma \circ \delta \circ \nu) \simeq \sigma^A\,(\delta^A\,(\Gamma^T\,\nu)) \\
&(\sigma \circ \delta)^T\,\nu :\equiv \sigma^M\,(\delta^T\,\nu) \circ \sigma^T\,(\delta \circ \nu)
\end{aligned}
$$

Above, we have

$$
\begin{aligned}
&\delta^T\,\nu &&: \Xi^T\,(\delta \circ \nu) \simeq \delta^A\,(\Gamma^T\,\nu) \\
&\sigma^M\,(\delta^T\,\nu) &&: \sigma^A\,(\Xi^T\,(\delta \circ \nu)) \simeq \sigma^A\,(\delta^A\,(\Gamma^T\,\nu)) \\
&\sigma^T\,(\delta \circ \nu) &&: \Delta^T\,(\sigma \circ \delta \circ \nu) \simeq \sigma^A\,(\Xi^T\,(\delta \circ \nu))
\end{aligned}
$$

Hence, the type of the composition in the definition checks out. We make use of the fact that σ^M sends an isomorphism in Γ to an isomorphism in Δ.

Substitution extension is a somewhat more complicated case. We want to interpret the extension of $\sigma : \mathsf{Sub}\,\Gamma\,\Delta$ with $t : \mathsf{Tm}\,\Gamma\,(A[\sigma])$:

$$(\sigma,\, t)^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to (\Delta \triangleright A)^T\,((\sigma,\, t) \circ \nu) \simeq (\sigma,\, t)^A\,(\Gamma^T\,\nu)$$

The goal is an isomorphism in the semantic $\Gamma \triangleright A$ category, i.e. the total category of A. Every isomorphism in $\Gamma \triangleright A$ arises as packing together a Γ isomorphism and a displayed A isomorphism over it. We can compute the type further:

$$(\sigma,\, t)^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to (\Delta^T\,(\sigma \circ \nu),\, A^T\,(\sigma \circ \nu)\,(t[\nu])) \simeq (\sigma^A\,(\Gamma^T\,\nu),\, t^A\,(\Gamma^T\,\nu))$$

We can exhibit $\sigma^T\,\nu : \Delta^T\,(\sigma \circ \nu) \simeq \sigma^A\,(\Gamma^T\,\nu)$ as the base component of the goal isomorphism. Now we need a displayed isomorphism over it. Following the pattern, we may try $t^T\,\nu$:

$$t^T\,\nu : (A[\sigma])^T\,\nu\,(t[\nu]) \simeq_{\mathsf{id}} t^A\,(\Gamma^T\,\nu)$$

Computing the type:

$$t^T\,\nu : \mathsf{coe}\,(\sigma^T\,\nu)\,(A^T\,(\sigma \circ \nu)\,(t[\nu])) \simeq_{\mathsf{id}} t^A\,(\Gamma^T\,\nu)$$

So this is not quite what is needed; we want a displayed iso over $\sigma^T\,\nu$, but we have something over id. We can fix this using coh:

$$\mathsf{coh}\,(\sigma^T\,\nu)\,(A^T\,(\sigma \circ \nu)\,(t[\nu])) : A^T\,(\sigma \circ \nu)\,(t[\nu]) \simeq_{\sigma^T\,\nu} \mathsf{coe}\,(\sigma^T\,\nu)\,(A^T\,(\sigma \circ \nu)\,(t[\nu]))$$

The composition of $t^T\,\nu$ and the above now checks out:

$$(\sigma,\, t)^T\,\nu :\equiv (\sigma^T\,\nu,\, t^T\,\nu \circ \mathsf{coh}\,(\sigma^T\,\nu)\,(A^T\,(\sigma \circ \nu)\,(t[\nu])))$$

We omit the rest of the cwf interpretation. It should be apparent that explicit coe and coh-handling is fairly technical. We note though that in a proof assistant, the finitary and infinitary term model constructions would be of similar difficulty, because there we cannot rely on equality reflection and implicit transports to magically tidy up the formalization. In fact, even in the finitary case it would be a good idea to structure the formalization around coercions and coherences.

The high-level explanation for why the weakened constructions continue to work, is the same as what we gave in the section on iso-fibrant semantics: we do nothing which would violate stability under isomorphisms; additionally, because

our isofibrations are *split*, coercion and coherence compute strictly on identities and compositions, which ensures that conversion equations in the syntax are respected. For example, functoriality of type substitution relies on coe computation on identity and composition.

Universe

The universe is interpreted as follows.

$$\begin{aligned}
&\mathsf{U}^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,\mathsf{U} \to \mathsf{Set}_{j+1} \\
&\mathsf{U}^T\,\nu\,a :\equiv \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a)
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{El}\,a)^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu]))) \to a^A\,(\Gamma^T\,\nu) \\
&(\mathsf{El}\,a)^T\,\nu\,t :\equiv (a^T\,\nu)\,t
\end{aligned}$$

In the interpretation of El, note that

$$a^T\,\nu : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu])) \simeq_{\mathsf{id}} a^A\,(\Gamma\,\nu)$$

But this is an isomorphism in the semantic U, which is the category of sets in Set_{j+1}. So coercion along $a^T\,\nu$ is simply function application, and we are justified in writing $(a^T\,\nu)\,t$.

For each type former in U, we have to exhibit an isomorphism of sets in the interpretation.

$\top$, Σ

We need

$$\top^T : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{U}^T\,\nu\,(\top[\nu]) \simeq_{\mathsf{id}} \top^A\,(\Gamma^T\,\nu)$$

The result type computes to $\mathsf{Tm}\,\Omega\,(\mathsf{El}\,\top) \simeq \top$, which is evident. For Σ, we have to show

$$\mathsf{Tm}\,\Omega\,(\mathsf{El}\,(\Sigma\,(a[\nu])\,(b[\nu \circ \mathsf{p}, \mathsf{q}]))) \simeq ((\alpha : a^A\,(\Gamma^T\,\nu)) \times b^A\,(\Gamma^T\,\nu, \alpha))$$

This follows from the induction hypotheses a^T and b^T, which establish the first and second components of the desired isomorphism.

Identity

For the identity type, we need

$$\mathsf{Tm}\,\Omega\,(\mathsf{El}\,(\mathsf{Id}\,(t[\nu])\,(u[\nu]))) \simeq (t^A\,(\Gamma^T\,\nu) \equiv u^A\,(\Gamma^T\,\nu))$$

This follows from $t^T\,\nu$, $u^T\,\nu$ and the specifying isomorphism of Id.

Infinitary functions

This function type follows the same pattern. We define the isomorphism below using induction hypotheses and the specifying isomorphism of Π^{inf}.

$$\mathsf{Tm}\,\Omega\,(\mathsf{El}\,(\Pi^{\mathsf{inf}}\,Ix\,(\lambda\,i.\,(b\,i)[\nu]))) \simeq ((i : Ix) \to (b\,i)^A\,(\Gamma\,\nu))$$

Inductive functions

Inductive functions are interpreted using transport along $a^T\,\nu : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu])) \simeq a^A\,(\Gamma^T\,\nu)$:

$$\begin{aligned}(\Pi\,a\,B)^T &: (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(\Pi\,(a[\nu])\,(B[\nu \circ \mathsf{p},\, \mathsf{q}]))) \\ &\quad \to (\alpha : a^A\,(\Gamma^T\,\nu)) \to B^A\,(\Gamma^T\,\nu,\, \alpha) \\ (\Pi\,a\,B)^T\,\nu\,t &:\equiv \lambda\,\alpha.\,B\,(\nu,\, (a^T\,\nu)^{-1}\,\alpha)\,(t\,((a^T\,\nu)^{-1}\,\alpha))\end{aligned}$$

External functions are interpreted the same way as in the finitary case.

5.6.2 Eliminator Construction

We only present the eliminator construction in the following, since (unique) recursors are derivable from this.

Compared to the finitary case, the eliminator construction does not change as much as the term algebra construction. The reason is that although we have weakened strict algebra equality to isomorphism, in the current construction we only have to show equalities of substitutions and terms, which we do not need to weaken (and they cannot be sensibly weakened anyway).

We assume j and k such that $j + 1 \leq k$, and also $\Omega : \mathsf{Sig}_j$ and $\omega^D : \Omega_k^D\,(\Omega^T\,\mathsf{id})$. Hence, ω^D is a displayed Ω-algebra over the term algebra, and we aim to construct its section. Note that we lift $\Omega^T\,\mathsf{id} : \Omega_{j+1}^A$ to level k by cumulativity. We define $-^E$

by induction on syn_j.

$$
\begin{aligned}
&-^E : (\Gamma : \mathsf{Con}) \qquad (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^S\,(\nu^A\,(\Omega^T\,\mathsf{id}))\,(\nu^D\,\omega^D)\\
&-^E : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta)(\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^E\,(\sigma \circ \nu) \equiv \sigma^S\,(\Gamma^E\,\nu)\\
&-^E : (A : \mathsf{Ty}\,\Gamma) \qquad (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(A[\nu])) \to A^S\,(t^A\,(\Omega^T\,\mathsf{id}))\,(t^D\,\omega^D)\,(\Gamma^E\,\nu)\\
&-^E : (t : \mathsf{Tm}\,\Gamma\,A)\ (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^E\,\nu\,(t[\nu]) \equiv t^S\,(\Gamma^E\,\nu)
\end{aligned}
$$

This is so far exactly the same as in Section 4.5.5. The subsequent changes arise from the need to transport along $-^T$ in definitions.

Universe

For the universe, we need

$$\mathsf{U}^E : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(a : \mathsf{Tm}\,\Omega\,\mathsf{U}) \to (\alpha : a^A\,(\Omega^T\,\mathsf{id})) \to a^D\,\omega^D\,\alpha$$

Since we only have $a^T\,\mathsf{id} : a^A\,(\Omega^T\,\mathsf{id}) \simeq \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a)$, the definition becomes

$$\mathsf{U}^E\,\nu\,a\,t :\equiv (a^T\,\mathsf{id}\,t)^D\,\omega^D$$

That this is well-typed, follows from

$$
\begin{aligned}
&((a^T\,\mathsf{id})\,t)^T\,\mathsf{id} \quad : t \equiv ((a^T\,\mathsf{id})\,t)^A\,(\Omega^T\,\mathsf{id})\\
&((a^T\,\mathsf{id})\,t)^D\,\omega^D : a^D\,\omega^D\,(((a^T\,\mathsf{id})\,t)^A\,(\Omega^T\,\mathsf{id}))
\end{aligned}
$$

For El, we need to show

$$(\mathsf{El}\,a)^E : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu]))) \to a^S\,(\Gamma^E\,\nu)\,(t^A\,(\Omega^T\,\mathsf{id})) \equiv t^D\,\omega^D$$

We have

$$t^T\,\mathsf{id} : (a[\nu])^T\,\mathsf{id}\,t \equiv t^A\,(\Omega^T\,\mathsf{id})$$

Moreover

$$a^E\,\nu : \mathsf{U}^E\,\nu\,(a[\nu]) \equiv a^S\,(\Gamma^E\,\nu)$$

Hence

$$a^E\,\nu : (\lambda\,t.\,((a[\nu])^T\,\mathsf{id}\,t)^D\,\omega^D) \equiv a^S\,(\Gamma^E\,\nu)$$

Applying both sides to $((a[\nu])^T\,\mathsf{id})^{-1}\,t$, we have

$$((a[\nu])^T\,\mathsf{id}\,(((a[\nu])^T\,\mathsf{id})^{-1}\,t))^D\,\omega^D \equiv a^S\,(\Gamma^E\,\nu)\,(((a[\nu])^T\,\mathsf{id})^{-1}\,t)$$

This simplifies to

$$t^D\,\omega^D \equiv a^S\,(\Gamma^E\,\nu)\,(((a[\nu])^T\,\mathsf{id})^{-1}\,t)$$

By $(a^T\,\mathsf{id}\,t)^T\,\mathsf{id} : t \equiv (a^T\,\mathsf{id}\,t)^A\,(\Omega^T\,\mathsf{id})$ this becomes:

$$t^D\,\omega^D \equiv a^S\,(\Gamma^E\,\nu)\,(((a[\nu])^T\,\mathsf{id})^{-1}\,((a^T\,\mathsf{id}\,t)^A\,(\Omega^T\,\mathsf{id})))$$

Thus we have the required

$$t^D\,\omega^D \equiv a^S\,(\Gamma^E\,\nu)\,(\Omega^T\,\mathsf{id})$$

$\top$, Σ

For $\top$, we need

$$\top^E : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{U}^E\,\nu\,\top \equiv \top^S\,(\Gamma^E\,\nu)$$

But this is clearly trivial, since $\top^S\,(\Gamma^E\,\nu) : \top \to \top$. Considering Σ:

$$(\Sigma\,a\,b)^E : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{U}^E\,\nu\,(\Sigma\,(a[\nu])\,(b[\nu \circ \mathsf{p},\,\mathsf{q}])) \equiv (\Sigma\,a\,b)^S\,(\Gamma^E\,\nu)$$

This case is a bit tedious. The sides above are functions, so appealing to function extensionality we apply both sides to $(\alpha,\,\beta)$, where $\alpha : a^A\,(\nu^A\,(\Omega^T\,\mathsf{id}))$ and $\beta : b^A\,(\nu^A\,(\Omega^T\,\mathsf{id}),\,\alpha)$. We also unfold some definitions:

$$(((\Sigma\,(a[\nu])\,(b[\nu \circ \mathsf{p},\,\mathsf{q}]))^T\,\mathsf{id})^{-1}\,(\alpha,\,\beta))^D\,\omega^D \equiv (a^S\,(\Gamma^E\,\nu)\,\alpha,\,b^S\,(\Gamma^E\,\nu,\,\mathsf{refl})\,\beta)$$

Unfolding the left side of this equation, we have

$$((((a[\nu])^T\,\mathsf{id})^{-1}\,\alpha)^D\,\omega^D,\,(((b[\nu \circ \mathsf{p},\,\mathsf{q}])^T\,(\mathsf{id},\,((a[\nu])^T\,\mathsf{id})^{-1}\,\alpha))^{-1}\,\beta)^D\,\omega^D)$$

Let us abbreviate $((a[\nu])^T\,\mathsf{id})^{-1}\,\alpha : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu]))$ as α':

$$(\alpha'^D\,\omega^D,\,(((b[\nu \circ \mathsf{p},\,\mathsf{q}])^T\,(\mathsf{id},\,\alpha'))^{-1}\,\beta)^D\,\omega^D)$$

Hence, we need to show component-wise equality of pairs. The equality of first components follow from the following:

$$a^E\,\nu : \mathsf{U}^E\,\nu\,(a[\nu]) \equiv a^S\,(\Gamma^E\,\nu)$$

Unfolding definitions and applying both sides to α, we the equality of first components:

$$\alpha'^D\,\omega^D \equiv a^S\,(\Gamma^E\,\nu)\,\alpha$$

Analogously, the equality of second components follows from

$$b^E\,(\nu,\,\alpha') : (((b[\nu,\,\alpha'])^T\,\mathsf{id})^{-1}\,\beta)^D\,\omega^D \equiv b^S\,(\Gamma^E\,\nu,\,\mathsf{refl})\,\beta$$

The right hand side is what we need, the left hand side though does not immediately match up. Hence, it remains to show that

$$(((b[\nu,\,\alpha'])^T\,\mathsf{id})^{-1}\,\beta)^D\,\omega^D \equiv (((b[\nu \circ \mathsf{p},\,\mathsf{q}])^T\,(\mathsf{id},\,\alpha'))^{-1}\,\beta)^D\,\omega^D$$

Thus, it suffices to show

$$(b[\nu,\,\alpha'])^T\,\mathsf{id} \equiv (b[\nu \circ \mathsf{p},\,\mathsf{q}])^T\,(\mathsf{id},\,\alpha')$$

This equation follows from a somewhat laborious unfolding of all involved definitions. In particular, we use that for some $a : \mathsf{Tm}\,\Gamma\,\mathsf{U}$, we have

$$(a[\sigma])^T\,\nu \equiv a^M\,(\sigma^T\,\nu) \circ a^T\,(\sigma \circ \nu)$$

which follows from the definition of $-^T$.

Inductive functions

In Π we likewise transport along the domain isomorphism.

$$\begin{aligned}
&(\Pi\,a\,B)^E : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(\Pi\,(a[\nu])\,(B[\nu \circ \mathsf{p},\,\mathsf{q}])))\\
&\qquad \to (\alpha : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,(a[\nu]))) \to B^S\,(t^A\,(\Omega^T\,\mathsf{id})\,\alpha)\,(t^D\,\omega^D\,(\alpha^D\,\omega^D))\,(\Gamma^E\,\nu,\,\mathsf{refl})\\
&(\Pi\,a\,B)^E\,\nu\,t :\equiv \lambda\,\alpha.\,B^E\,(\nu,\,(a[\nu])^T\,\mathsf{id}\,\alpha)\,(t\,((a[\nu])^T\,\mathsf{id}\,\alpha))
\end{aligned}$$

This is well-typed by the following:

$$\begin{aligned}
&a^E\,\nu && : ((a[\nu])^T\,\mathsf{id}\,\alpha)^D\,\omega^D \equiv a^S\,(\Gamma^E\,\nu)\,\alpha\\
&((a[\nu])^T\,\mathsf{id}\,\alpha)^T\,\mathsf{id} && : \alpha \equiv ((a[\nu])^T\,\mathsf{id}\,\alpha)^A\,(\Omega^T\,\mathsf{id})
\end{aligned}$$

Id, Π^{inf}, Π^{ext}

Id is trivial by UIP, and for Π^{inf} and Π^{ext} we again do a straightforward recursion under the indexing function.

This concludes the definition of $-^E$. We again show the initiality of term algebras.

Definition 66 (**Eliminators**)**.** Assuming $\Omega : \mathsf{Sig}_j$, a k level such that $k \geq j+1$ and $\omega^D : \Omega_k^D\,(\Omega^T\,\mathsf{id})$, we have $\Omega^E\,\mathsf{id} : \Omega^S\,(\Omega^T\,\mathsf{id})\,\omega^D$ as the eliminator.

Theorem 10. *$\Omega^T\,\mathsf{id} : \Omega^A_{j+1}$ is initial when lifted to any $k \geq j+1$ level.*

Proof. $\Omega^T\,\mathsf{id}$ supports elimination by Definition 66, and elimination is equivalent to initiality by Theorem 1. □

5.7 Levitation and Bootstrapping

In this section we adapt the bootstrapping procedure from Section 4.6 to infinitary signatures.

Bootstrapping for 2LTT semantics

If we only want to write down signatures and get their 2LTT-based semantics, a simplified bootstrapping suffices, which is essentially the same as in Section 4.6. We write $\mathsf{ToS}_i : \mathsf{Set}_{i+1}$ for the type of models where underlying sets are in Set_i and external indexing is over Ty_0. We also have $\mathbf{M}_i : \mathsf{ToS}_{i+2}$ for the flcwf models where underlying sets in algebras are in Set_i and external indexing is over types in Ty_0.

Definition 67. The type of **bootstrap signatures** is defined as follows:

$$\mathsf{BootSig} :\equiv (i : \mathsf{Level}) \to (M : \mathsf{ToS}_i) \to \mathsf{Con}_M$$

These bootstrap signatures only allow external indexing by types in Ty_0. We can write bootstrap signatures and interpret them in $\mathbf{M}_i$, by applying them to $\mathbf{M}_i$.

Bootstrapping for term algebras

Now we reuse the ETT setting from Section 4.5.2. We have $\mathsf{ToS}_{i,j} : \mathsf{Set}_{i+1 \sqcup j+1}$ for the type of models where underlying sets are in Set_i and Π^{inf} and Π^{ext} abstract over Set_j. We also have $\mathbf{M}_{i,j} : \mathsf{ToS}_{(i+1 \sqcup j)+1,j}$ as the flcwf models, again with underlying sets of algebras in Set_i and external indexing types in Set_j.

Definition 68. The type of **bootstrap signatures** at level j is defined as follows. These may contain external indexing by types in Set_j.

$$\mathsf{BootSig}_j :\equiv (i\,j : \mathsf{Level}) \to (M : \mathsf{ToS}_{i,j}) \to \mathsf{Con}_M$$

Definition 69 (Signature for ToS). We define $\mathsf{ToSSig}_j : \mathsf{BootSig}_{j+1}$ as the bootstrap signature for ToS, where the described signatures may be indexed by types in Set_j. Like in Section 4.6, we use an internal notation. We present an excerpt.

$$
\begin{aligned}
&\mathsf{Con} &&: \mathsf{U}\\
&\mathsf{Sub} &&: \mathsf{Con} \to \mathsf{Con} \to \mathsf{U}\\
&\mathsf{Ty} &&: \mathsf{Con} \to \mathsf{U}\\
&\mathsf{Tm} &&: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{U}\\
&\ldots\\
&\mathsf{SigU} &&: \{\Gamma : \mathsf{Con}\} \to \mathsf{El}\,(\mathsf{Ty}\,\Gamma)\\
&\mathsf{SigEl} &&: \{\Gamma : \mathsf{Con}\} \to \mathsf{Tm}\,\Gamma\,\mathsf{SigU} \to \mathsf{El}\,(\mathsf{Ty}\,\Gamma)\\
&\Pi^{\mathsf{inf}} &&: \{\Gamma : \mathsf{Con}\}(A : \mathsf{Set}_j) \to^{\mathsf{ext}} (A \to^{\mathsf{inf}} \mathsf{Tm}\,\Gamma\,\mathsf{SigU}) \to \mathsf{El}\,(\mathsf{Tm}\,\Gamma\,\mathsf{SigU})\\
&\ldots
\end{aligned}
$$

Now the interpretation of ToSSig_j in $\mathbf{M}_{i,j+1}$ yields the flcwf where objects are elements of $\mathsf{ToS}_{i,j}$. Note the level bump: ToSSig_j is in $\mathsf{BootSig}_{j+1}$, so we expend one level at each round of self-description. We get the notion of ToS-induction from $\mathbf{M}_{i,j+1}$, and we have $\mathsf{ToS}_{i,j} \leq \mathsf{ToS}_{i+1,j}$ (by definition of ToS and the rules of subtyping), which allows us to specify what it means for a model to support elimination into any universe. Thus we recover all concepts that are used in the term algebra and eliminator constructions.

5.8 Related Work

This chapter is based on the publication "Large and Infinitary Quotient Inductive-Inductive Types" [KK20b]. We make the following changes:

- We use 2LTT for the flcwf semantics, while the paper only used the cumulative ETT setting.
- We add the construction of left adjoints and the signature-based semantics in Sections 5.3-5.4.
- We add small $\top$ and Σ to the ToS, and also their large counterparts in Sections 5.3-5.4.

Fiore, Pitts and Steenkamp developed QW types [FPS20], a notion of quotient inductive types that generalizes W-types. They construct QW-types in Agda, using sized types and "nested" inductive usage of simple quotients. This means using $-/-$ as an external type operator, applied to internal inductive sorts in a signature, similarly to how we use lists in the rose tree example in Section 4.1.

Essentially algebraic theories generalize to the infinitary cases in a straightforward way [AAR$^+$94].

Specific examples of infinitary QIITs where introduced in [Uni13], as QIITs for Cauchy real numbers, surreal numbers, and cumulative set hierarchies. In [ADK17], a partiality monad is specified as an infinitary QIITs.

CHAPTER 6

Higher Inductive-Inductive Signatures

So far we only considered semantics of signatures where equality constructors are interpreted as proof-irrelevant equalities, i.e. those satisfying UIP. This inspires the naming of *quotient* inductive-inductive signatures. In contrast, *higher inductive-inductive* signatures are characterized by having possibly proof-relevant and iterated equalities in algebras. The natural setting of HIITs is homotopy type theory (HoTT) [Uni13], where higher equalities can be manipulated and constructed in non-trivial ways. We might think of HIITs as generalizations of QIITs, or alternatively, view QIITs as set-truncated HIITs.

The theory of HII signatures is fairly similar to the theory of infinitary QII signatures. The main difference is that the internal Id type does not support equality reflection, nor UIP. In fact, infinitary QII signatures already allow iterated Id, and most HIITs that occur in the literature can be already expressed using QII signatures. In contrast, the semantics of signatures changes markedly: the semantic inner theory is now intensional, and Id is interpreted as intensional inner equality. This may not seem that dramatic, but note that so far we have made very heavy use of UIP and inner equality reflection in the semantics, and now these are not available.

The more general semantics introduces significant complications. As a result, in the following we shall restrict ourselves to the AMDS fragment of the semantics. This is sufficient to compute what we mean by induction and initiality (which has been called "homotopy initiality" in the context of HoTT [Soj15a]).

Why do not we go further? The main reason is that the natural semantics is actually in $(\omega, 1)$-categories: we want $(\omega, 1)$-categories of algebras. This requires a different approach and toolset. In particular, in [KK20a, Section 9] we gave an

example that a naive attempt to extend the AMDS semantics of signatures with the notion of identity morphisms already fails. The author of this thesis is not versed enough in higher category theory, so we leave the exposition of the full semantics to future work.

We do note that a higher semantics has been developed by Capriotti and Sattler. See [CS] for an abstract; the bulk of the work remains unpublished as of now. In short, Capriotti and Sattler define the ToS in 2LTT, and also use 2LTT to give a model where signatures are higher categories, specified as complete Segal types. They show that categories of algebras have finite limits and that initiality is equivalent to induction. Additionally, the setup yields a structure identity principle for each signature. However, reductions to simpler type formers are not discussed, nor possible term algebra constructions. Both of these appear to be far more difficult than in the quotient setting, and to the author's knowledge there are no concrete proposals how to approach them.

The necessity of 2LTT

2LTT is firmly necessary in the specification of HIITs, and the ToS must live in the outer layer. The reason is that there is no known way to sensibly internalize the metatheory of type theories purely inside HoTT. This is the problem of "HoTT eating itself" [Shu14]. It is also closely related to the problem of representing semisimplicial types in HoTT. If we can construct semisimplicial types in an embedded type theory, and interpret that into non-truncated HoTT types, that would indeed solve the problem. But so far it has not been solved, or proven impossible to solve. A key original motivation for 2LTT was precisely to allow construction of semisimplicial types [ACKS19].

We give a short summary of the problem; see [KK20a, Section 4] for more discussion. The goal is to have a notion of model of a dependent type theory in HoTT, such that we have a standard model where contexts are HoTT types.

We may define the notion of model naively using types and equalities, by having $\mathsf{Con} : \mathsf{U}$, $\mathsf{Ty} : \mathsf{Con} \to \mathsf{U}$, etc. and $\mathsf{idl} : \sigma \circ \mathsf{id} = \sigma$. However, this does not yield a well-behaved notion of *syntax*. If we define the syntax as HIIT for the above notion (i.e. the initial model), nothing forces the underlying types to be sets; the HIIT definition freely adds a large number of non-trivial higher paths. Since the underlying types are not sets, this syntax does not have decidable equality, by Hedberg's theorem [Hed98]. This is regardless of what type formers we include.

Alternatively, we may define the notion of model as having homotopy sets for underlying types. The corresponding HIIT will be in fact a QIIT, where every inductive sort is set-truncated. While this is better-behaved as syntax, we do not get a standard model. Contexts in a model cannot be arbitrary types, because in HoTT, types (of a universe) do not form a h-set. In fact, not even h-sets form a h-set; they form a groupoid. So we do not get any reasonable notion of standard interpretation.

2LTT solves this issue in the following way: the embedded syntax is an outer QIIT, and equations in the syntax are given as strict (outer) equalities. The standard inner type model is now possible, because in that model all equations hold strictly, up to inner definitional equality. However, this implies that we can *only* define strict models; this leads to the following consideration.

Strict vs. weak signatures

We have an important choice in the semantics: homomorphisms (and sections) can preserve structure strictly, i.e. up to outer equality, or weakly, up to inner equality. This choice has an impact on the supported ToS features.

- With strict preservation, the semantics does not support an elimination rule for Id. The problem is that Id is necessarily modeled as inner equality, but we cannot eliminate from that to outer types, and strict equality is an outer type.

- With weak preservation, we do have elimination for Id. However, the semantics does not support strict $\beta\eta$ rules in Id, Σ, Π^{inf} and Π. In short, the problem is that $(\mathsf{El}\, a)^M$ and $(\mathsf{El}\, s)^S$ are defined as inner equality types, so we need to use inner path induction in the semantics of eliminators. This implies that $\beta\eta$-rules also hold only up to inner paths, but not definitionally. Thus, in the "weak" case, we may have $\beta\eta$ only up to internal Id.

It makes sense to develop both semantics. Weak morphisms and sections are useful, because they can be defined purely in the inner theory (or in HoTT). Strict morphisms and sections are useful if we want to specify type formers, since type theories usually assume strict β-rules for recursors and eliminators. In this chapter, we specify theories of signatures and semantics for both cases.

Metatheory

We work in 2LTT. We assume that Ty_0 is closed under Π, Σ, $\top$ and intensional identity $- = -$. We assume the "based" path induction principle [Uni13, Section 1.12.1]. Assuming $A : \mathsf{Ty}_0$, $x : A$ and $B : (y : A) \to x = y \to \mathsf{Ty}_0$, we have

$$\begin{aligned}\mathsf{J}_P &: P\,x\,\mathsf{refl} \to \{y : A\}(p : x = y) \to P\,y\,p\\ \mathsf{J}_P\beta &: \mathsf{J}_P\,pr\,\mathsf{refl} \equiv pr\end{aligned}$$

The following operations are defined in the standard way [Uni13, Section 2].

- Path inversion $-^{-1} : x = y \to y = x$.
- Path composition $- \centerdot - : x = y \to y = z \to x = z$.
- Assuming $P : A \to \mathsf{Ty}_0$, we have transport $\mathsf{tr}_P : x = y \to P\,x \to P\,y$.
- Path lifting $\mathsf{ap} : (f : A \to B) \to x = y \to f\,x = f\,y$.
- Dependent path lifting $\mathsf{apd} : (f : (x : A) \to B\,x) \to (p : x = y) \to \mathsf{tr}_B\,p\,(f\,x) = f\,y$.

6.1 Strict Signatures

Definition 70. A **model of strict ToS** is the same as a model of the theory of infinitary QII signatures, with the following change: the Id type former in U only supports refl, but no elimination rule or reflection rule.

We assume that the syntax of ToS exists, and a signature is a context in the syntax. We could use bootstrap signatures as well, without loss of generality, as we will not use actual induction on signatures in the following, and we will also not discuss fine-grained sizing or cumulativity of algebras.

Example 32. The circle is one of the simplest higher inductive types [Uni13, Section 6.4]. The signature is the following.

$$\begin{aligned}\mathsf{S}^1 &: \mathsf{U}\\ \mathsf{base} &: \mathsf{El}\,\mathsf{S}^1\\ \mathsf{loop} &: \mathsf{El}\,(\mathsf{Id}\,\mathsf{base}\,\mathsf{base})\end{aligned}$$

Note that the circle signature is expressible as a QII signature, but in the QII semantics the loop entry is made trivial by UIP.

Non-examples

From the HoTT book, all higher-inductive types are supported, except

- The torus [Uni13, Section 6.6], since the specification contains Id composition, which requires Id elimination.
- The "hubs-and-spokes" HITs [Uni13, Section 6.7]. This involves abstracting over some external $x : \mathsf{S}^1$ (a point of the circle), then referring to a ToS term which is computed by elimination on x. This is also not permitted in our setup, because signature terms live in the outer theory of 2LTT, and external parameters are in Ty_0.

  If instead signatures and external parameters lived in the same theory (like in our ETT setup for term algebra constructions of QIITs), this elimination would be possible. For HIITs, we cannot do that, since the inner theory cannot reasonably internalize the ToS.

6.1.1 Semantics

For each signature Γ, we wish to compute

$$\begin{aligned}\Gamma^A &: \mathsf{Set}\\ \Gamma^M &: \Gamma^A \to \Gamma^A \to \mathsf{Set}\\ \Gamma^D &: \Gamma^A \to \mathsf{Set}\\ \Gamma^S &: (\gamma : \Gamma^A) \to \Gamma^D\,\gamma \to \mathsf{Set}\end{aligned}$$

corresponding respectively to algebras, morphisms, displayed algebras and sections. Note that all of these return in Set. Morphisms and sections in particular are forced to return in Set, because they may contain strict equalities.

The AMDS interpretations can be found in Appendix B in a tabular manner, together with a listing of ToS components. We discuss these in the following.

In **algebras** and **displayed algebras** there is no complication; all equations holds in these (displayed) models strictly, and we do not use equations from induction hypotheses anywhere.

In **morphisms**, note that all term formers returning in El specify a strict equation. We write refl in their definition for brevity, which is technically correct (by equality reflection), but the definitions may involve using the strict equalities

from induction hypotheses. $\top^M\,\gamma^M : \mathsf{tt}_0 \equiv \mathsf{tt}_0$ is trivial, but

$$(\mathsf{proj}_1\,t)^M\,\gamma^M : a^M\,\gamma^M\,((\mathsf{proj}_1\,t)^A\,\gamma_0) \equiv (\mathsf{proj}_1\,t)^A\,\gamma_1$$

requires us to use

$$t^M\,\gamma^M : (a^M\,\gamma^M\,((\mathsf{proj}_1\,t)^A\,\gamma_0),\, b^M\,(\gamma^M,\,\mathsf{refl})\,((\mathsf{proj}_2\,t)^A\,\gamma_1)) \equiv t^A\,\gamma_1$$

Likewise we use $t^M\,\gamma^M$ in the equation for $(\mathsf{proj}_2\,t)^M$.

Also note that the definition for $(\mathsf{Id}\,t\,u)^M\,\gamma^M$ relies on t^M and u^M for well-typing. The goal is

$$\begin{aligned}&(\mathsf{Id}\,t\,u)^M\,\gamma^M : (\mathsf{Id}\,t\,u)^A\,\gamma_0 \to (\mathsf{Id}\,t\,u)^A\,\gamma_1\\&(\mathsf{Id}\,t\,u)^M\,\gamma^M : t^A\,\gamma_0 = u^A\,\gamma_0 \to t^A\,\gamma_0 = u^A\,\gamma_1\end{aligned}$$

Assuming $p : t^A\,\gamma_0 = u^A\,\gamma_0$, we have $\mathsf{ap}\,(a^M\,\gamma^M)\,p : a^M\,\gamma^M\,(t^A\,\gamma_0) = a^M\,\gamma^M\,(u^A\,\gamma_0)$, so we rewrite the sides along $t^M\,\gamma^M : a^M\,\gamma^M\,(t^A\,\gamma_0) \equiv t^A\,\gamma_1$ and $u^M\,\gamma^M$. The ap application must stay explicit in the definition, since inner equalities can be proof-relevant.

We also demonstrate the failure of Id elimination. It is enough to show that Id inversion fails. This would entail the following in the ToS:

$$-^{-1} : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\mathsf{Id}\,t\,u)) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\mathsf{Id}\,u\,t))$$

In the $-^M$ interpretation, we would need to show

$$\begin{aligned}&(p^{-1})^M\,\gamma^M : \mathsf{ap}\,(a^M\,\gamma^M)\,((p^{-1})^A\,\gamma_0) \equiv ((p^{-1})^A\,\gamma_1)\\&(p^{-1})^M\,\gamma^M : \mathsf{ap}\,(a^M\,\gamma^M)\,(p^A\,\gamma_0)^{-1} \equiv (p^A\,\gamma_1)^{-1}\end{aligned}$$

We have $p^M : \mathsf{ap}\,(a^M\,\gamma^M)\,(p^A\,\gamma_0) \equiv p^A\,\gamma_1$, so we would need to show

$$\mathsf{ap}\,(a^M\,\gamma^M)\,(p^A\,\gamma_0)^{-1} \equiv (\mathsf{ap}\,(a^M\,\gamma^M)\,(p^A\,\gamma_0))^{-1}$$

This is not provable in 2LTT; it is false as a universal statement in the initial model (syntax) of the inner theory. It holds in the empty context, where both sides are necessarily equal to refl by canonicity, but not in arbitrary contexts. It does hold as an inner equality, by induction on $p^A\,\gamma_0$.

Sections are a mostly mechanical generalization of morphisms, where the codomain depends on the domain. Note that the $(\mathsf{Id}\,t\,u)^D$ definition is a path-over-path, and accordingly we have apd instead of ap in $(\mathsf{Id}\,t\,u)^S$.

Definition 71. For some Γ signature, notions of **initiality** and **induction** are as follows.

$$\begin{aligned}&\mathsf{Initial} \quad (\gamma : \Gamma^A) :\equiv (\gamma' : \Gamma^A) \to \mathsf{isContr}\,(\Gamma^M\,\gamma\,\gamma')\\&\mathsf{Inductive}\,(\gamma : \Gamma^A) :\equiv (\gamma^D : \Gamma^D\,\gamma) \to \Gamma^S\,\gamma\,\gamma^D\end{aligned}$$

This is the same as Definition 43, except we do not have an flcwf of algebras, so do not have properties that are evident in an flcwf, such as Theorems 1 and 2.

Example 33. For the circle signature $\mathsf{S^1Sig}$, we have the following (disregarding the leading $\top$ components):

$$\mathsf{S^1Sig^A} \equiv (S^1 : \mathsf{Ty}_0) \times (\mathit{base} : S^1) \times (\mathit{loop} : \mathit{base} = \mathit{base})$$

$$\begin{aligned}&\mathsf{S^1Sig^D}\,(S^1,\,\mathit{loop},\,\mathit{base}) \equiv\\&\quad (S^{1D} \quad : S^1 \to \mathsf{Ty}_0)\\&\times (\mathit{base}^D : S^{1D}\,\mathit{base})\\&\times (\mathit{loop}^D : \mathsf{tr}_{S^{1D}}\,\mathit{loop}\,\mathit{base}^D = \mathit{base}^D)\end{aligned}$$

$$\begin{aligned}&\mathsf{S^1Sig^S}\,(S^1,\,\mathit{loop},\,\mathit{base})\,(S^{1D},\,\mathit{loop}^D,\,\mathit{base}^D) \equiv\\&\quad (S^{1S} \quad : (s : S^1) \to S^{1D}\,s)\\&\times (\mathit{base}^S : S^{1S}\,\mathit{base} \equiv \mathit{base}^D)\\&\times (\mathit{loop}^S : \mathsf{apd}\,S^{1S}\,\mathit{loop} \equiv \mathit{loop}^D)\end{aligned}$$

The computed induction principles are close to what we find in [Uni13]. The difference is that β-rules for path constructors are strict, while in ibid. they are up to propositional equality. One reason for choosing weak β-rules for paths is that we have ap and apd applications on the left sides of such rules, and it is unconventional to definitionally specify the behavior of operations which are derived from J. In cubical type theories, path β-rules are specified in a more primitive way, so strict computation is more organic.

Currently, we have semantics in intensional inner theories, but it would be possible to do the same in cubical inner theories. Intensional TT is clearly much simpler, and has a wider variety of known models. On the other hand, cubical type theories support strictly computing transports, so it is possible that they would support stricter ToS β-rules in the case of the "weak" semantics. We leave this to possible future work.

6.2 Weak Signatures

Metatheory

On top of what we had so far in this chapter, we assume *strong function extensionality* in the inner theory: this means that for each $f, g : (a : A) \to B\,a$, the following function is an equivalence.

$$\begin{aligned}&\mathsf{happly} : (f = g) \to ((a : A) \to f\,a = g\,a)\\&\mathsf{happly}\,p\,a :\equiv \mathsf{ap}\,(\lambda\,f.\,f\,a)\,p\end{aligned}$$

funext is obtained as the inverse of happly. This definition, unlike the simple assumption of funext, is well-behaved in intensional settings [Uni13, Section 2.9].

Moreover, we assume two universes U_0 and U_1, such that $\mathsf{U}_0 \leq \mathsf{U}_1 \leq \mathsf{Ty}_0$. We use this to develop semantics which is entirely in the inner theory: if algebra sorts are in U_0, we need an U_1 on top of that to accommodate types of algebras.

Definition 72. A **model of weak ToS** consists of a base cwf (with Con, Sub, Ty and Tm returning in Set) extended with certain type formers. We omit all substitution rules in the following. As before, substitution rules are given with strict equality. We list type formers below.

- A "large" identity type $\mathsf{ID} : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma$, with the following rules:

$$\begin{aligned}&\mathsf{refl} : \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,t\,t)\\&\mathsf{J} \quad : \{t : \mathsf{Tm}\,\Gamma\,A\}(P : \mathsf{Ty}\,(\Gamma \triangleright (u : A) \triangleright (p : \mathsf{ID}\,t\,u)))\\&\quad \to \mathsf{Tm}\,\Gamma\,(P[u \mapsto t,\, p \mapsto \mathsf{refl}])\\&\quad \to \{u : \mathsf{Tm}\,\Gamma\,A\}(p : \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,t\,u)) \to \mathsf{Tm}\,\Gamma\,(P[u \mapsto u,\, p \mapsto p])\\&\mathsf{J}\beta \ : \mathsf{J}\,b\,pr\,\mathsf{refl} \equiv pr\end{aligned}$$

  Notation 22. We may use a name binding notation in the induction motive for J. For example, assuming $A : \mathsf{Ty}\,\Gamma$, $B : \mathsf{Ty}\,(\Gamma \triangleright A)$, $p : \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,t\,u)$ and $pt : \mathsf{Tm}\,\Gamma\,(B[\mathsf{id},\, t])$, we may define transport along p as

$$\mathsf{J}\,(x\,p.\,B[\mathsf{id},\, x])\,pt\,p : \mathsf{Tm}\,\Gamma\,(B[\mathsf{id},\, u])$$

  where $x\,p.$ binds the term and path dependencies of the induction motive.

- A universe U with decoding El.
- U is closed under a "small" identity type $\mathsf{Id} : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a) \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$, with elimination principle J targeting any type (not just types in U!). The β-rule is specified with ID.

$$
\begin{aligned}
&\mathsf{refl} : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\mathsf{Id}\,t\,t))\\
&\mathsf{J} \quad : \{t : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)\}(P : \mathsf{Ty}\,(\Gamma \triangleright (u : \mathsf{El}\,a) \triangleright (p : \mathsf{El}\,(\mathsf{Id}\,t\,u))))\\
&\quad \to \mathsf{Tm}\,\Gamma\,(P[u \mapsto t,\, p \mapsto \mathsf{refl}])\\
&\quad \to \{u : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)\}(p : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\mathsf{Id}\,t\,u))) \to \mathsf{Tm}\,\Gamma\,(P[u \mapsto u,\, p \mapsto p])\\
&\mathsf{J}\beta \ : \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,(\mathsf{J}\,b\,pr\,\mathsf{refl})\,pr)
\end{aligned}
$$

- U is also closed under $\top$, Σ, and Π^{inf}. Of these, $\top$ is specified with a strict Set isomorphism $\mathsf{Tm}\,\Gamma\,(\mathsf{El}\,\top) \simeq \top$, while Σ and Π^{inf} are specified with equivalences up to ID. These are equivalences in the sense of HoTT [Uni13, Chapter 4]. There are several equivalent formulations of equivalence; we pick the bi-invertible definitions here.

  Thus, Σ is specified as follows.

$$
\begin{aligned}
&\neg,\neg \ : (t : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)) \times \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b[\mathsf{id},\, t])) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Sigma\,a\,b))\\
&\mathsf{proj} \ : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Sigma\,a\,b)) \to (t : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)) \times \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b[\mathsf{id},\, t]))\\
&\mathsf{proj}' : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Sigma\,a\,b)) \to (t : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)) \times \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b[\mathsf{id},\, t]))\\
&\beta_1 \quad : \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,(\mathsf{proj}_1\,(t,\, u))\,t)\\
&\beta_2 \quad : \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,((\mathsf{J}\,(x\ \_.\ (\mathsf{El}\,b)[\mathsf{id},\, x])\,(\mathsf{proj}_2\,(t,\, u))\,\beta_1)\,u))\\
&\eta \quad : \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,(\mathsf{proj}'_1\,t,\, \mathsf{proj}'_2\,t)\,t)
\end{aligned}
$$

  We write proj_i and proj'_i for composing metatheoretic projections with ToS projections. The additional proj' component is required to get a bi-invertible equivalence. Also note that β_2 is only well-typed up to β_1, so we need to use a transport in the specification.

  $\Pi^{\mathsf{inf}} : (Ix : \mathsf{U}_0) \to (Ix \to \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$ is specified below.

$$
\begin{aligned}
&\mathsf{app}^{\mathsf{inf}} \ : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Pi^{\mathsf{inf}}\,Ix\,b)) \to ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b\,i)))\\
&\mathsf{lam}^{\mathsf{inf}} \ : ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b\,i))) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Pi^{\mathsf{inf}}\,Ix\,b))\\
&\mathsf{lam}^{\mathsf{inf}'} : ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b\,i))) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Pi^{\mathsf{inf}}\,Ix\,b))
\end{aligned}
$$

$$\begin{aligned}\beta &: \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,(\mathsf{app}^{\mathsf{inf}}\,(\mathsf{lam}^{\mathsf{inf}}\,t)\,i)\,(t\,i))\\ \eta &: \mathsf{Tm}\,\Gamma\,(\mathsf{ID}\,(\mathsf{lam}^{\mathsf{inf}'}\,(\mathsf{app}^{\mathsf{inf}}\,t))\,t)\end{aligned}$$

Why have equivalences in the specification of models, would it be enough to have isomorphisms? We choose equivalences because they yield better-behaved models, and they do not make it any harder to construct models, since we can always construct the required equivalences from isomorphisms [Uni13, Chapter 4].

- Inductive function type $\Pi : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Ty}\,(\Gamma \triangleright \mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma$, with the specifying equivalence given up to ID, analogously as for Σ and Π^{inf}:

$$(\mathsf{app}, \mathsf{lam}, \mathsf{lam}') : \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) \simeq \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B$$

- External function type $\Pi^{\mathsf{ext}} : (Ix : \mathsf{U}_0) \to (Ix \to \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,\Gamma$, specified as a strict Set isomorphism:

$$(\mathsf{app}^{\mathsf{ext}}, \mathsf{lam}^{\mathsf{ext}}) : \mathsf{Tm}\,\Gamma\,(\Pi^{\mathsf{ext}}\,Ix\,B) \simeq ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(B\,i))$$

To give a short summary of changes compared to strict signatures:

1. Types are closed under an extra ID type former which has a strict β-rule.
2. We can eliminate from Id to proper types, but with a weak β-rule.
3. Σ and Π^{inf} support eliminators, but with weak β-rules.

Example 34. The torus is now expressible thanks to path elimination in signatures. We define $-\blacksquare-$ as path composition for Id in the evident way.

$$\begin{aligned}\mathsf{T}^2 &: \mathsf{U}\\ \mathsf{b} &: \mathsf{El}\,\mathsf{T}^2\\ \mathsf{p} &: \mathsf{El}\,(\mathsf{Id}\,\mathsf{b}\,\mathsf{b})\\ \mathsf{q} &: \mathsf{El}\,(\mathsf{Id}\,\mathsf{b}\,\mathsf{b})\\ \mathsf{t} &: \mathsf{El}\,(\mathsf{Id}\,(\mathsf{p} \blacksquare \mathsf{q})\,(\mathsf{q} \blacksquare \mathsf{p}))\end{aligned}$$

We could also use ID instead of Id and get equivalent semantics.

Example 35. The ID type lets us express "sort equivalences". For example, a signature for integers can be compactly written as follows [AS20]:

$$\begin{aligned}\mathsf{Int} &: \mathsf{U}\\ \mathsf{zero} &: \mathsf{El}\,\mathsf{Int}\\ \mathsf{p} &: \mathsf{ID}\,\mathsf{Int}\,\mathsf{Int}\end{aligned}$$

We get the suc constructor by coercing along p, and predecessors by coercing backwards.

Recall that in Chapter 5 we dropped sort equations because of their non-fibrancy in the semantics. In contrast, there is no issue with sort equations here. Sort equations simply become inner paths between types in the semantics; if we assume univalence in the inner theory, such paths are equivalent to type equivalences. Hence, sort equations in HITTs can be viewed as shorthands for sort equivalences. Without sort equations, it is still possible to write equivalences in signatures, using any of the standard definitions [Uni13, Chapter 4].

6.2.1 Semantics

We do not repeat the tables for the strict ToS semantics in Appendix B, as much of it remains essentially the same in the weak case. We consider the components of the model in order, highlighting relevant changes and points of interest.

Notation 23. We may omit induction motives in tr and J in the following, as they will often get excessively verbose. So we may write $\mathsf{tr}\,p\,px : P\,y$ for $p : x = y$ and $px : P\,x$, and use $\mathsf{J}\,pr\,p$ similarly.

Cwf

A notable change here is that the entirety of the semantics is now in the inner theory. This means that the interpretation functions of contexts and types all return in U_1, e.g. $\Gamma^A : \mathsf{U}_1$ and $\Gamma^M : \Gamma^A \to \Gamma^A \to \mathsf{U}_1$. Accordingly, we use type formers in U_1 to interpret structure in the base cwf, e.g. $\top^A :\equiv \top$, where the $\top$ on the right is in U_1. The only change though is the move from Set to U_1, all definitions are essentially the same.

ID

The new ID type former is interpreted as pointwise equality of semantic terms. We assume t, $u : \mathsf{Tm}\,\Gamma\,A$.

$$\begin{aligned}
(\mathsf{ID}\,t\,u)^A\,\gamma &:\equiv t^A\,\gamma = u^A\,\gamma\\
(\mathsf{ID}\,t\,u)^M\,p_0\,p_1\,\gamma^M &:\equiv \mathsf{tr}\,p_1\,(\mathsf{tr}\,p_0\,(t^M\,\gamma^M)) = u^M\,\gamma^M\\
(\mathsf{ID}\,t\,u)^D\,p\,\gamma^D &:\equiv \mathsf{tr}_{(\lambda\,x.\,A^D\,x\,\gamma^D)}\,p\,(t^D\,\gamma^D) = u^D\,\gamma^D\\
(\mathsf{ID}\,t\,u)^S\,p\,p^D\,\gamma^S &:\equiv \mathsf{tr}\,p^D\,(\mathsf{J}\,(t^S\,\gamma^S)\,p) = u^S\,\gamma^S
\end{aligned}$$

Above, we dropped induction motives in tr and J in $-^M$ and $-^S$. For illustration, the more explicit definitions are:

$$\begin{aligned}
&(\mathsf{ID}\,t\,u)^M\,p_0\,p_1\,\gamma^M :\equiv\\
&\quad \mathsf{tr}_{(\lambda\,x.\,A^M\,x\,(t^A\,\gamma_1)\,\gamma^M)}\,p_1\,(\mathsf{tr}_{(\lambda\,x.\,A^M\,(t^A\,\gamma_1)\,x\,\gamma^M)}\,p_0\,(t^M\,\gamma^M)) = u^M\,\gamma^M\\
&(\mathsf{ID}\,t\,u)^S\,p\,p^D\,\gamma^S :\equiv\\
&\quad \mathsf{tr}_{(\lambda\,x.\,A^S\,x\,(u^A\,\gamma)\,\gamma^S)}\,p^D\\
&\qquad (\mathsf{J}_{(\lambda\,y\,p.\,A^S\,y\,(\mathsf{tr}_{(\lambda\,x.\,A^D\,x\,\gamma^D)}\,p\,(t^D\,\gamma^D)))}\,(t^S\,\gamma^S)\,p) = u^S\,\gamma^S
\end{aligned}$$

From now on, we shall generally avoid this amount of detail in motives.

refl is interpreted as pointwise refl-s:

$$\begin{aligned}
\mathsf{refl}^A\ _ &:\equiv \mathsf{refl}\\
\mathsf{refl}^M\ _ &:\equiv \mathsf{refl}\\
\mathsf{refl}^D\ _ &:\equiv \mathsf{refl}\\
\mathsf{refl}^S\ _ &:\equiv \mathsf{refl}
\end{aligned}$$

Let us look at J for ID now. It is helpful to temporarily consider a bundled AMDS model instead of the four interpretation maps. Then, we have the following equivalence up to $-=-$:

$$\mathsf{Tm}_{\mathsf{AMDS}}\,\Gamma\,(\mathsf{ID}_{\mathsf{AMDS}}\,t\,u) \simeq (t = u)$$

This follows from function extensionality and the characterization of equivalence for inner Σ [Uni13, Section 2.7]. Thus, semantic ID is the same as equality of semantic terms. It follows that everything in the inner theory respects ID, so we can certainly define the semantic J for ID.

The actual definition of J involves doing induction on all paths that are available as induction hypotheses.

$$\begin{aligned}
(\mathsf{J}\,P\,pr\,p)^A\,\gamma &:\equiv \mathsf{J}\,(pr^A\,\gamma)\,(p^A\,\gamma)\\
(\mathsf{J}\,P\,pr\,p)^M\,\gamma^M &:\equiv \mathsf{J}\,(\mathsf{J}\,(\mathsf{J}\,(pr^M\,\gamma^M)\,(p^A\,\gamma_1))\,(p^A\,\gamma_0))\,(p^M\,\gamma^M)\\
(\mathsf{J}\,P\,pr\,p)^D\,\gamma^D &:\equiv \mathsf{J}\,(\mathsf{J}\,(pr^D\,\gamma^D)\,(p^A\,\gamma))\,(p^D\,\gamma^D)\\
(\mathsf{J}\,P\,pr\,p)^S\,\gamma^S &:\equiv \mathsf{J}\,(\mathsf{J}\,(\mathsf{J}\,(pr^S\,\gamma^S)\,(p^A\,\gamma))\,(p^D\,\gamma^D))\,(p^S\,\gamma^S)
\end{aligned}$$

The strict β-rule for J is supported, as the above definition computes everywhere when p is refl.

Universe

We have the following changes. First, the interpretations of U now return in U_0:

$$\begin{aligned}
\mathsf{U}^A\,\gamma &:\equiv \mathsf{U}_0\\
\mathsf{U}^D\,a\,\gamma^D &:\equiv a \to \mathsf{U}_0
\end{aligned}$$

Second, in El, morphisms and sections are given by inner equality:

$$\begin{aligned}
(\mathsf{El}\,a)^M\,\alpha_0\,\alpha_1\,\gamma^M &:\equiv a^M\,\gamma^M\,\alpha_0 = \alpha_1\\
(\mathsf{El}\,a)^M\,\alpha\,\alpha^D\,\gamma^D &:\equiv a^S\,\gamma^S\,\alpha = \alpha^D
\end{aligned}$$

Id

In this identity type, $-^A$ and $-^D$ are pointwise equality as usual, and $-^M$ and $-^S$ complete squares of equalities. We assume t, $u : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)$.

$$\begin{aligned}
(\mathsf{Id}\,t\,u)^A\,\gamma &:\equiv t^A\,\gamma = u^A\,\gamma\\
(\mathsf{Id}\,t\,u)^M\,\gamma^M &:\equiv \lambda\,(p : t^A\,\gamma_0 = u^A\,\gamma_0).\,(t^M\,\gamma^M)^{-1} \cdot \mathsf{ap}\,(a^M\,\gamma^M)\,p \cdot u^M\,\gamma^M\\
(\mathsf{Id}\,t\,u)^D\,\gamma^D &:\equiv \lambda\,(p : t^A\,\gamma = u^A\,\gamma).\,\mathsf{tr}_{(a^D\,\gamma^D)}\,(t^D\,\gamma^D) = u^D\,\gamma^D\\
(\mathsf{Id}\,t\,u)^S\,\gamma^S &:\equiv \lambda\,(p : t^A\,\gamma = u^A\,\gamma).\,\mathsf{ap}\,(\mathsf{tr}_{(a^D\,\gamma^D)}\,p)\,(t^S\,\gamma^S)^{-1} \cdot \mathsf{apd}\,(a^S\,\gamma^S)\,p \cdot u^S\,\gamma^S
\end{aligned}$$

We have $\mathsf{refl}^A\,\_ :\equiv \mathsf{refl}$ and $\mathsf{refl}^D\,\_ :\equiv \mathsf{refl}$. For $\mathsf{refl}^M\,\gamma^M$, the goal type is

$$(t^M\,\gamma^M)^{-1} \cdot t^M\,\gamma^M = \mathsf{refl}$$

which is one of the groupoid laws for paths [Uni13, Section 2.1]. We have a more dependent variant as goal type for $\mathsf{refl}^S\,\gamma^S$:

$$\mathsf{ap}\,(\lambda\,x.\,x)\,(t^S\,\gamma^S)^{-1} \cdot t^S\,\gamma^S = \mathsf{refl}$$

This again follows from groupoid laws and the functoriality of ap.

It is still the case that $\mathsf{Tm}_{\mathsf{AMDS}}\,\Gamma\,(\mathsf{El}_{\mathsf{AMDS}}\,(\mathsf{Id}_{\mathsf{AMDS}}\,t\,u)) \simeq (t = u)$ up to $- = -$. Although $(\mathsf{Id}\,t\,u)^M$ and $(\mathsf{Id}\,t\,u)^S$ do not express equality of t and u, we do get the component-wise equalities if we apply El. We have that

$$(\mathsf{El}\,(\mathsf{Id}\,t\,u))^M\,p_0\,p_1\,\gamma^M \equiv ((t^M\,\gamma^M)^{-1} \centerdot \mathsf{ap}\,(a^M\,\gamma^M)\,p_0 \centerdot u^M\,\gamma^M = p_1)$$

We can rearrange the definition to make it more apparent that this is an equality of $t^M\,\gamma^M$ and $u^M\,\gamma^M$, which is well-typed up to p_0 and p_1.

$$\mathsf{ap}\,(a^M\,\gamma^M)\,p_0 \centerdot u^M\,\gamma^M = t^M\,\gamma^M \centerdot p_1$$

Thus, we can again expect that J is definable for Id. The definition gets a bit more complicated than in ID, as in $-^M$ and $-^S$ we need to generalize over values in the inner most J, by returning a function from the J and applying it to the appropriate values.

$$\begin{aligned}
&(\mathsf{J}\,P\,pr\,p)^A\,\gamma :\equiv \mathsf{J}\,(pr^A\,\gamma)\,(p^A\,\gamma)\\
&(\mathsf{J}\,P\,pr\,p)^M\,\gamma^M :\equiv\\
&\quad \mathsf{J}\,(\mathsf{J}\,(\mathsf{J}\,(\mathsf{J}\,(\lambda\,x\,y.\,y)\,(t^M\,\gamma^M)\,(pr^A\,\gamma_1)\,(pr^M\,\gamma^M))\,(p^A\,\gamma_0))\,(u^M\,\gamma^M))\,(p^M\,\gamma^M)\\
&(\mathsf{J}\,P\,pr\,p)^D\,\gamma^D :\equiv \mathsf{J}\,(\mathsf{J}\,(pr^D\,\gamma^D)\,(p^A\,\gamma))\,(p^D\,\gamma^D)\\
&(\mathsf{J}\,P\,pr\,p)^S\,\gamma^S :\equiv\\
&\quad \mathsf{J}\,(\mathsf{J}\,(\mathsf{J}\,(\mathsf{J}\,(\lambda\,x\,y.\,y)\,(t^S\,\gamma^S)\,(pr^D\,\gamma^D)\,(pr^S\,\gamma^S))\,(p^A\,\gamma))\,(u^S\,\gamma^S))\,(p^S\,\gamma^S)
\end{aligned}$$

Remark. Definitions like this probably look far from enlightening, but they should be actually helpful for formalization in a proof assistant. The essential information, which is somewhat tedious to work out independently, is which things to generalize, and in what order to use path induction, and these are conveyed above.

Regarding the β-rule, note that refl^M and refl^S are not defined as refl, but rather by induction on $t^M\,\gamma^M$ and $t^S\,\gamma^S$. Therefore, if we apply J to refl, the $-^M$ and $-^S$ components do not strictly compute.

$\top$

$\top$ is unchanged. tt^M and tt^S could possibly change (since El has changed, and $\mathsf{tt} : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,\top)$), but they are still definable with refl-s.

Σ

Pairing and the projections change in Σ; now their $-^M$ and $-^S$ cases return proof-relevant inner equalities. In pairing, we do path induction on hypotheses:

$$\begin{aligned}(t,\, u)^M\, \gamma^M &:\equiv \mathsf{J}\,(\mathsf{J}\,\mathsf{refl}\,(t^M\,\gamma^M))\,(u^M\,\gamma^M)\\ (t,\, u)^S\, \gamma^S &:\equiv \mathsf{J}\,(\mathsf{J}\,\mathsf{refl}\,(t^S\,\gamma^S))\,(u^S\,\gamma^S)\end{aligned}$$

In proj_1, we use $\mathsf{ap}\,\mathsf{proj}_1$ on path hypotheses:

$$\begin{aligned}(\mathsf{proj}_1\,t)^M\,\gamma^M &:\equiv \mathsf{ap}\,\mathsf{proj}_1\,(t^M\,\gamma^M)\\ (\mathsf{proj}_1\,t)^S\,\gamma^S &:\equiv \mathsf{ap}\,\mathsf{proj}_1\,(t^S\,\gamma^S)\end{aligned}$$

In proj_2, the definitions could be given using $\mathsf{apd}\,\mathsf{proj}_2$, but the result type does not immediately line up, so we can just do direct path induction.

$$\begin{aligned}(\mathsf{proj}_2\,t)^M\,\gamma^M &:\equiv \mathsf{J}\,\mathsf{refl}\,(t^M\,\gamma^M)\\ (\mathsf{proj}_2\,t)^S\,\gamma^S &:\equiv \mathsf{J}\,\mathsf{refl}\,(t^S\,\gamma^S)\end{aligned}$$

proj_1' and proj_2' (required by the bi-invertible specification) are defined the same way. We do not have strict $\beta\eta$-rules. For example:

$$(\mathsf{proj}_1\,(t,\, u))^M\,\gamma^M \equiv \mathsf{ap}\,\mathsf{proj}_1\,(\mathsf{J}\,(\mathsf{J}\,\mathsf{refl}\,(t^M\,\gamma^M))\,(u^M\,\gamma^M)) \not\equiv t^M\,\gamma^M$$

We still get $(\mathsf{proj}_1\,(t,\, u))^M\,\gamma^M = t^M\,\gamma^M$ by path induction on $t^M\,\gamma^M$ and $u^M\,\gamma^M$, and similarly in other cases, so Σ in the ToS does support the specifying equivalence.

Π^{inf}

Again, the $-^M$ and $-^S$ cases change in term formers. Application is given by happly:

$$\begin{aligned}(\mathsf{app}^{\mathsf{inf}}\,t\,i)^M\,\gamma^M &:\equiv \mathsf{happly}\,(t^M\,\gamma^M)\\ (\mathsf{app}^{\mathsf{inf}}\,t\,i)^S\,\gamma^S &:\equiv \mathsf{happly}\,(t^S\,\gamma^S)\end{aligned}$$

Abstraction is by funext:

$$\begin{aligned}(\mathsf{lam}^{\mathsf{inf}}\,t)^M\,\gamma^M &:\equiv \mathsf{funext}\,(\lambda\,i.\,(t\,i)^M\,\gamma^M)\\ (\mathsf{lam}^{\mathsf{inf}}\,t)^S\,\gamma^S &:\equiv \mathsf{funext}\,(\lambda\,i.\,(t\,i)^S\,\gamma^S)\end{aligned}$$

Thus, weak $\beta\eta$-rules for Π^{inf} follow from strong function extensionality.

Π

We need to use explicit path induction in app^M and app^S:

$$(\mathsf{app}\, t)^M\, (\gamma^M,\, \alpha^M) :\equiv \mathsf{J}\, (t^M\, \gamma^M\, \alpha_0)\, \alpha^M \quad \text{where} \quad \alpha^M : a^M\, \gamma^M\, \alpha_0 = \alpha_1$$
$$(\mathsf{app}\, t)^S\, (\gamma^S,\, \alpha^S) \quad :\equiv \mathsf{J}\, (t^S\, \gamma^S\, \alpha)\, \alpha^S \qquad \text{where} \quad \alpha^S \ : a^S\, \gamma^S\, \alpha = \alpha^D$$

In contrast, lam does not change. $\beta\eta$-rules are given by replaying the path inductions on app^M and app^S.

Π^{ext}

The interpretation of Π^{ext} is unchanged. This concludes the AMDS semantics of weak signatures.

6.3 Discussion & Related Work

6.3.1 Evaluation

The main advantage of the signatures in the current chapter is their generality. We cover almost every higher inductive definition in the literature, and do so in a direct manner, with minimal encoding overhead.

It is also possible to mechanically check validity of signatures and compute AMDS interpretations. The current author has written a Haskell program which takes as input a weak HII signature, and outputs ADS interpretations as well-formed Agda source code [Kov20]. The syntax is a bit more restricted than what we have in this chapter, and the program does not compute morphisms; but it is clear that the deficiencies would be straightforward to patch up.

On the other hand, we note that our semantics is in a minimal intensional theory, a fragment of the "book" version of homotopy type theory. This setting supports neither computational univalence nor computational higher inductive types. If our goal is to add computationally adequate HIITs to a theory (and eventually to its implementation), the current chapter is not immediately applicable. As we mentioned in Section 4.4.4, in a cubical setting we would need to reformulate both signatures and semantics. However, the current work should be still helpful as a guideline, and a provide a point of comparison and validation.

6.3.2 Related Work

This chapter is based on "A Syntax for Higher Inductive-Inductive Types" [KK18] and "Signatures and Induction Principles for Higher Inductive-Inductive Types" [KK20a], both by Ambrus Kaposi and the current author. The latter is an extended journal version of the former. In this chapter, we extend and refine these sources in the following ways.

- We use 2LTT. In the papers, we instead used a custom syntactic translation: the theory of signatures was an ad hoc mixture of the inner and outer theory, and the AMDS interpretations were syntactic translations targeting the inner theory. The setup turns out to be mostly the same as here; but 2LTT brings a lot of clarity and convenience.

- We add the strict/weak signature distinction. The papers only considered weak signatures and semantics.

- We improve on the specification of signatures. The papers had a small Id type with elimination only to U, not to arbitrary types. The journal version also had a second identity type, but only for sort equations, i.e. it expressed only equality of inhabitants of U.

  The small and large identity types in this chapter are more expressive; the weaker definitions in the paper were just oversights.

  The papers also omitted eliminators of type formers in weak signatures, and thus their $\beta\eta$ rules, and they did not have $\top$ or Σ. However, this was done mostly for the sake of brevity, as these extra features are not really used in any HIIT signature in the literature. It makes more sense to include the extras here, to match infinitary QII signatures as much as possible.

The homotopy type theory book [Uni13] introduced numerous higher inductive types and developed their use cases, but it did not give a theory of signatures, nor discussed semantics.

Sojakova [Soj15b] specified a class of HITs called W-suspensions (building on W-types), and proved the equivalence of induction and homotopy initiality, working internally to an intensional type theory.

Lumsdaine and Shulman gave a general specification of models of type theories supporting higher inductive types [LS]. They gave a more semantic specification

of algebras, as algebras of a cell monad, and characterized the class of models which support initial algebras. They did not cover indexed families or induction-induction.

Dybjer and Moeneclaey [DM18] gave signatures for class of finitary HITs with up to 2-dimensional path constructors, and built semantics in groupoids.

Coquand, Huber and Mörtberg [CHM18] specified syntax for a cubical type theory which supports several HITs (sphere, torus, suspensions, truncations, pushouts) and built semantics in cubical sets.

Cavallo and Harper [CH19] specify HITs which support indexed families and arbitrary higher paths, although not induction-induction. They provide semantics in a PER (partial equivalence relation) realizability setting.

Cubical Agda [VMA21] is the principal proof assistant which natively supports computational univalence and HITs. Its implementation of pattern matching, mutual inductive definitions, termination checking and strict positivity checking yields of a large class of higher *inductive-inductive* types. However, there is no compact theory of signatures (valid specifications fall out from positivity/termination checking) nor a categorical semantics.

APPENDIX A

AMDS interpretation of FQII signatures

This appendix supplements Chapter 4. It contains the AMDS interpretation for finitary QII signatures. We omit substitution and $\beta\eta$-rules. We also omit the Tm_0 decoding operation of two-level type theory.

Components of ToS (without substitution and $\beta\eta$-rules)

$$
\begin{array}{ll}
\mathsf{Con} & : \mathsf{Set} \\
\mathsf{Sub} & : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\mathsf{Ty} & : \mathsf{Con} \to \mathsf{Set} \\
\mathsf{Tm} & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \\
\bullet & : \mathsf{Con} \\
\epsilon & : \mathsf{Sub}\,\Gamma\,\bullet \\
\mathsf{id} & : \mathsf{Sub}\,\Gamma\,\Gamma \\
- \circ - & : \mathsf{Sub}\,\Delta\,\Xi \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Sub}\,\Gamma\,\Xi \\
-[-] & : \mathsf{Ty}\,\Delta \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma \\
-[-] & : \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma]) \\
\mathsf{p} & : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma \\
\mathsf{q} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}]) \\
(-,-) & : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma]) \to \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A) \\
\mathsf{U} & : \mathsf{Ty}\,\Gamma \\
\mathsf{El} & : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma \\
\mathsf{Id} & : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma \\
\mathsf{refl} & : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,t\,t) \\
\mathsf{reflect} & : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,t\,u) \to t \equiv u \\
\Pi & : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Ty}\,(\Gamma \triangleright \mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma \\
\mathsf{app} & : \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) \to \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) \\
\Pi^{\mathsf{ext}} & : (Ix : \mathsf{Ty}_0) \to (Ix \to \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,\Gamma \\
\mathsf{app}^{\mathsf{ext}} & : \mathsf{Tm}\,\Gamma\,(\Pi^{\mathsf{ext}}\,Ix\,B) \to (i : Ix) \to \mathsf{Tm}\,\Gamma\,(B\,i) \\
\mathsf{lam}^{\mathsf{ext}} & : ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(B\,i)) \to \mathsf{Tm}\,\Gamma\,(\Pi^{\mathsf{ext}}\,Ix\,B)
\end{array}
$$

Algebras

$$
\begin{array}{ll}
{\_}^A & : \mathsf{Con} \to \mathsf{Set} \\
{\_}^A & : \mathsf{Sub}\,\Gamma\,\Delta \to \Gamma^A \to \Delta^A \\
{\_}^A & : \mathsf{Ty}\,\Gamma \to \Gamma^A \to \mathsf{Set} \\
{\_}^A & : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \Gamma^A) \to A^A\,\gamma \\
\bullet^A & :\equiv \top \\
\epsilon^A\,\gamma & :\equiv \mathsf{tt} \\
\mathsf{id}^A\,\gamma & :\equiv \gamma \\
(\sigma \circ \delta)^A\,\gamma & :\equiv \sigma^A\,(\delta^A\,\gamma) \\
(\Gamma \triangleright A)^A & :\equiv (\gamma : \Gamma^A) \times A^A\,\gamma \\
(A[\sigma])^A\,\gamma & :\equiv A^A\,(\sigma^A\,\gamma) \\
(t[\sigma])^A\,\gamma & :\equiv t^A\,(\sigma^A\,\gamma) \\
\mathsf{p}^A\,(\gamma,\,\alpha) & :\equiv \gamma \\
\mathsf{q}^A\,(\gamma,\,\alpha) & :\equiv \alpha \\
(\sigma,\,t)^A\,\gamma & :\equiv (\sigma^A\,\gamma,\,t^A\,\gamma) \\
\mathsf{U}^A\,\gamma & :\equiv \mathsf{Ty}_0 \\
(\mathsf{El}\,a)^A\,\gamma & :\equiv a^A\,\gamma \\
(\mathsf{Id}\,t\,u)^A\,\gamma & :\equiv t^A\,\gamma \equiv u^A\,\gamma \\
\mathsf{refl}^A\,\gamma & :\equiv \mathsf{refl} : t^A\,\gamma \equiv t^A\,\gamma \\
(\mathsf{reflect}\,p)^A & :\equiv \mathsf{funext}\,(\lambda\,\gamma.\,p^A\,\gamma) \\
(\Pi\,a\,B)^A\,\gamma & :\equiv (\alpha : a^A\,\gamma) \to B^A\,(\gamma,\,\alpha) \\
(\mathsf{app}\,t)^A\,(\gamma,\,\alpha) & :\equiv t^A\,\gamma\,\alpha \\
(\mathsf{lam}\,t)^A\,\gamma & :\equiv \lambda\,\alpha.\,t^A\,(\gamma,\,\alpha) \\
(\Pi^{\mathsf{ext}}\,Ix\,B)^A\,\gamma & :\equiv (i : Ix) \to (B\,i)^A\,\gamma \\
(\mathsf{app}^{\mathsf{ext}}\,t\,i)^A\,\gamma & :\equiv t^A\,\gamma\,i \\
(\mathsf{lam}^{\mathsf{ext}}\,t)^A\,\gamma & :\equiv \lambda\,i.\,(t\,i)^A\,\gamma
\end{array}
$$

Morphisms

$$
\begin{array}{ll}
\_^M & : (\Gamma : \mathsf{Con}) \to \Gamma^A \to \Gamma^A \to \mathsf{Set} \\
\_^M & : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \Gamma^M\,\gamma_0\,\gamma_1 \to \Delta^M\,(\sigma^A\,\gamma_0)\,(\sigma^A\,\gamma_1) \\
\_^M & : (A : \mathsf{Ty}\,\Gamma) \to A^A\,\gamma_0 \to A^A\,\gamma_1 \to \Gamma^M\,\gamma_0\,\gamma_1 \to \mathsf{Set} \\
\_^M & : (t : \mathsf{Tm}\,\Gamma\,A) \to (\gamma^M : \Gamma^M\,\gamma_0\,\gamma_1) \to A^M\,(t^A\,\gamma_0)\,(t^A\,\gamma_1)\,\gamma^M \\
\bullet^M\,\gamma_0\,\gamma_1 & :\equiv \top \\
\epsilon^M\,\gamma^M & :\equiv \mathsf{tt} \\
\mathsf{id}^M\,\gamma^M & :\equiv \gamma^M \\
(\sigma \circ \delta)^M\,\gamma^M & :\equiv \sigma^M\,(\delta^M\,\gamma^M) \\
(\Gamma \triangleright A)^M\,(\gamma_0,\,\alpha_0)\,(\gamma_1,\,\alpha_1) & :\equiv (\gamma^M : \Gamma^M\,\gamma_0\,\gamma_1) \times A^M\,\alpha_0\,\alpha_1\,\gamma^M \\
(A[\sigma])^M\,\alpha_0\,\alpha_1\,\gamma^M & :\equiv A^M\,\alpha_0\,\alpha_1\,(\sigma^M\,\gamma^M) \\
(t[\sigma])^M\,\gamma^M & :\equiv t^M\,(\sigma^M\,\gamma^M) \\
\mathsf{p}^M\,(\gamma^M,\,\alpha^M) & :\equiv \gamma^M \\
\mathsf{q}^M\,(\gamma^M,\,\alpha^M) & :\equiv \alpha^M \\
(\sigma,\,t)^M\,\gamma^M & :\equiv (\sigma^M\,\gamma^M,\,t^M\,\gamma^M) \\
\mathsf{U}^M\,a_0\,a_1\,\gamma^M & :\equiv a_0 \to a_1 \\
(\mathsf{El}\,a)^M\,\alpha_0\,\alpha_1\,\gamma^M & :\equiv a^M\,\gamma^M\,\alpha_0 \equiv \alpha_1 \\
(\mathsf{Id}\,t\,u)^M\,p_0\,p_1\,\gamma^M & :\equiv t^M\,\gamma^M \equiv u^M\,\gamma^M \\
\mathsf{refl}^M\,\gamma^M & :\equiv \mathsf{refl} : t^M\,\gamma^M \equiv t^M\,\gamma^M \\
(\mathsf{reflect}\,p)^M & :\equiv \mathsf{funext}\,(\lambda\,\gamma^M.\,p^M\,\gamma^M) \\
(\Pi\,a\,B)^M\,t_0\,t_1\,\gamma^M & :\equiv (\alpha : a^A\,\gamma_0) \to B^M\,(t_0\,\alpha)\,(t_1\,(a^M\,\gamma^M\,\alpha))\,(\gamma^M,\,\mathsf{refl}) \\
(\mathsf{app}\,t)^M\,(\gamma^M,\,\alpha^M) & :\equiv t^M\,\gamma^M\,\alpha_0 \quad \text{where} \quad \alpha^M : a^M\,\gamma^M\,\alpha_0 \equiv \alpha_1 \\
(\mathsf{lam}\,t)^M\,\gamma^M & :\equiv \lambda\,\alpha.\,t^M\,(\gamma^M,\,\mathsf{refl}) \quad \text{where} \quad \mathsf{refl} : a^M\,\gamma^M\,\alpha \equiv a^M\,\gamma^M\,\alpha \\
(\Pi^{\mathsf{ext}}\,Ix\,B)^M\,t_0\,t_1\,\gamma^M & :\equiv (i : Ix) \to (B\,i)^M\,(t_0\,i)\,(t_1\,i)\,\gamma^M \\
(\mathsf{app}^{\mathsf{ext}}\,t\,i)^M\,\gamma^M & :\equiv t^M\,\gamma^M\,i \\
(\mathsf{lam}^{\mathsf{ext}}\,t)^M\,\gamma^M & :\equiv (t\,i)^M\,\gamma^M
\end{array}
$$

Displayed algebras

$$
\begin{array}{ll}
{\_}^D & : (\Gamma : \mathsf{Con}) \to \Gamma^A \to \mathsf{Set} \\
{\_}^D & : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \Gamma^D\,\gamma \to \Delta^D\,(\sigma^A\,\gamma) \\
{\_}^D & : (A : \mathsf{Ty}\,\Gamma) \to A^A\,\gamma \to \Gamma^D\,\gamma \to \mathsf{Set} \\
{\_}^D & : (t : \mathsf{Tm}\,\Gamma\,A) \to (\gamma^D : \Gamma^D\,\gamma) \to A^D\,(t^A\,\gamma)\,\gamma^D \\
\bullet^D\,\gamma & :\equiv \top \\
\epsilon^D\,\gamma^D & :\equiv \mathsf{tt} \\
\mathsf{id}^D\,\gamma^D & :\equiv \gamma^D \\
(\sigma \circ \delta)^D\,\gamma^D & :\equiv \sigma^D\,(\delta^D\,\gamma^D) \\
(\Gamma \triangleright A)^D\,(\gamma,\,\alpha) & :\equiv (\gamma^D : \Gamma^D\,\gamma) \times A^D\,\alpha\,\gamma^D \\
(A[\sigma])^D\,\alpha\,\gamma^D & :\equiv A^D\,\alpha\,(\sigma^D\,\gamma^D) \\
(t[\sigma])^D\,\gamma^D & :\equiv t^D\,(\sigma^D\,\gamma^D) \\
\mathsf{p}^D\,(\gamma^D,\,\alpha^D) & :\equiv \gamma^D \\
\mathsf{q}^D\,(\gamma^D,\,\alpha^D) & :\equiv \alpha^D \\
(\sigma,\,t)^D\,\gamma^D & :\equiv (\sigma^D\,\gamma^D,\,t^D\,\gamma^D) \\
\mathsf{U}^D\,a\,\gamma^D & :\equiv a \to \mathsf{Ty}_0 \\
(\mathsf{El}\,a)^D\,t\,\gamma^D & :\equiv a^D\,\gamma^D\,t \\
(\mathsf{Id}\,t\,u)^D\,\gamma^D & :\equiv t^D\,\gamma^D \equiv u^D\,\gamma^D \\
\mathsf{refl}^D\,\gamma^D & :\equiv \mathsf{refl} : t^D\,\gamma^D \equiv t^D\,\gamma^D \\
(\mathsf{reflect}\,p)^D & :\equiv \mathsf{funext}\,(\lambda\,\gamma^D.\,p^D\,\gamma^D) \\
(\Pi\,a\,B)^D\,t\,\gamma^D & :\equiv \{\alpha : a^A\,\gamma\}(\alpha^D : a^D\,\gamma^D\,\alpha) \to B^D\,(t\,\alpha)\,(\gamma^D,\,\alpha^D) \\
(\mathsf{app}\,t)^D\,(\gamma^D,\,\alpha^D) & :\equiv t^D\,\gamma^D\,\alpha^D \\
(\mathsf{lam}\,t)^D\,\gamma^D & :\equiv \lambda\,\{\alpha\}\,\alpha^D.\,t^D\,(\gamma^D,\,\alpha^D) \\
(\Pi^{\mathsf{ext}}\,Ix\,B)^D\,t\,\gamma^D & :\equiv (i : Ix) \to (B\,i)^D\,(t\,i)\,\gamma^D \\
(\mathsf{app}^{\mathsf{ext}}\,t\,i)^D\,\gamma^D & :\equiv t^D\,\gamma^D\,i \\
(\mathsf{lam}^{\mathsf{ext}}\,t)^D\,\gamma & :\equiv \lambda\,i.\,(t\,i)^D\,\gamma^D
\end{array}
$$

Sections

$$
\begin{array}{ll}
{-}^S & : (\Gamma : \mathsf{Con}) \to (\gamma : \Gamma^A) \to \Gamma^A\,\gamma \to \mathsf{Set} \\
{-}^S & : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \Gamma^S\,\gamma\,\gamma^D \to \Delta^S\,(\sigma^A\,\gamma)\,(\sigma^D\,\gamma^D) \\
{-}^S & : (A : \mathsf{Ty}\,\Gamma) \to A^A\,\gamma \to A^D\,\gamma^D \to \Gamma^S\,\gamma\,\gamma^D \to \mathsf{Set} \\
{-}^S & : (t : \mathsf{Tm}\,\Gamma\,A) \to (\gamma^S : \Gamma^S\,\gamma\,\gamma^D) \to A^S\,(t^A\,\gamma)\,(t^D\,\gamma^D)\,\gamma^S \\
\bullet^S\,\gamma\,\gamma^D & :\equiv \top \\
\epsilon^S\,\gamma^S & :\equiv \mathsf{tt} \\
\mathsf{id}^S\,\gamma^S & :\equiv \gamma^S \\
(\sigma \circ \delta)^S\,\gamma^S & :\equiv \sigma^S\,(\delta^S\,\gamma^S) \\
(\Gamma \triangleright A)^S\,(\gamma,\,\alpha)\,(\gamma^D,\,\alpha^D) & :\equiv (\gamma^S : \Gamma^S\,\gamma\,\gamma^D) \times A^S\,\alpha\,\alpha^D\,\gamma^S \\
(A[\sigma])^S\,\alpha\,\alpha^D\,\gamma^S & :\equiv A^S\,\alpha\,\alpha^D\,(\sigma^S\,\gamma^S) \\
(t[\sigma])^S\,\gamma^S & :\equiv t^S\,(\sigma^S\,\gamma^S) \\
\mathsf{p}^S\,(\gamma^S,\,\alpha^S) & :\equiv \gamma^S \\
\mathsf{q}^S\,(\gamma^S,\,\alpha^S) & :\equiv \alpha^S \\
(\sigma,\,t)^S\,\gamma^S & :\equiv (\sigma^S\,\gamma^S,\,t^S\,\gamma^S) \\
\mathsf{U}^S\,a\,a^D\,\gamma^S & :\equiv (\alpha : a) \to a^D\,\alpha \\
(\mathsf{El}\,a)^S\,\alpha\,\alpha^D\,\gamma^S & :\equiv a^S\,\gamma^S\,\alpha \equiv \alpha^D \\
(\mathsf{Id}\,t\,u)^S\,p\,p^D\,\gamma^S & :\equiv t^S\,\gamma^S \equiv u^S\,\gamma^S \\
\mathsf{refl}^S\,\gamma^S & :\equiv \mathsf{refl} : t^S\,\gamma^S \equiv t^S\,\gamma^S \\
(\mathsf{reflect}\,p)^S & :\equiv \mathsf{funext}\,(\lambda\,\gamma^S.\,p^S\,\gamma^S) \\
(\Pi\,a\,B)^S\,t\,t^D\,\gamma^S & :\equiv (\alpha : a^A\,\gamma) \to B^S\,(t\,\alpha)\,(t^D\,(a^S\,\gamma^S\,\alpha))\,(\gamma^S,\,\mathsf{refl}) \\
(\mathsf{app}\,t)^S\,(\gamma^S,\,\alpha^S) & :\equiv t^S\,\gamma^S\,\alpha \quad \text{where} \quad \alpha^S : a^S\,\gamma^S\,\alpha \equiv \alpha^D \\
(\mathsf{lam}\,t)^S\,\gamma^S & :\equiv \lambda\,\alpha.\,t^S\,(\gamma^S,\,\mathsf{refl}) \quad \text{where} \quad \mathsf{refl} : a^S\,\gamma^S\,\alpha \equiv a^S\,\gamma^S\,\alpha \\
(\Pi^{\mathsf{ext}}\,Ix\,B)^S\,t\,t^D\,\gamma^S & :\equiv (i : Ix) \to (B\,i)^S\,(t\,i)\,(t^D\,i)\,\gamma^S \\
(\mathsf{app}^{\mathsf{ext}}\,t\,i)^S\,\gamma^S & :\equiv t^S\,\gamma^S\,i \\
(\mathsf{lam}^{\mathsf{ext}}\,t)^S\,\gamma^S & :\equiv (t\,i)^S\,\gamma^S
\end{array}
$$

APPENDIX B

AMDS interpretation of strict HII signatures

This appendix supplements Chapter 6. It contains the AMDS interpretation for strict HII signatures. We omit substitution and $\beta\eta$-rules. We also omit the Tm_0 decoding operation of two-level type theory.

Components of ToS (without equations)

$$
\begin{aligned}
&\mathsf{Con} && : \mathsf{Set}\\
&\mathsf{Sub} && : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}\\
&\mathsf{Ty} && : \mathsf{Con} \to \mathsf{Set}\\
&\mathsf{Tm} && : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}\\
&\bullet && : \mathsf{Con}\\
&\epsilon && : \mathsf{Sub}\,\Gamma\,\bullet\\
&\mathsf{id} && : \mathsf{Sub}\,\Gamma\,\Gamma\\
&-\circ- && : \mathsf{Sub}\,\Delta\,\Xi \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Sub}\,\Gamma\,\Xi\\
&-[-] && : \mathsf{Ty}\,\Delta \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma\\
&-[-] && : \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma])\\
&\mathsf{p} && : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma\\
&\mathsf{q} && : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])\\
&(-,-) && : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma]) \to \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)\\
&\mathsf{U} && : \mathsf{Ty}\,\Gamma\\
&\mathsf{El} && : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma\\
&\top && : \mathsf{Tm}\,\Gamma\,\mathsf{U}\\
&\mathsf{tt} && : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,\top)\\
&\Sigma && : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,\mathsf{U} \to \mathsf{Tm}\,\Gamma\,\mathsf{U}\\
&\mathsf{proj}_1 && : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Sigma\,a\,b)) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)\\
&\mathsf{proj}_2 && : (t : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Sigma\,a\,b))) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b[\mathsf{id},\,\mathsf{proj}_1\,t]))\\
&(-,-) && : (t : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b[\mathsf{id},\,t])) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Sigma\,a\,b))\\
&\mathsf{Id} && : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a) \to \mathsf{Tm}\,\Gamma\,\mathsf{U}\\
&\mathsf{refl} && : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\mathsf{Id}\,t\,t))\\
&\Pi^{\mathsf{inf}} && : (Ix : \mathsf{Ty}_0) \to (Ix \to \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Tm}\,\Gamma\,\mathsf{U}\\
&\mathsf{app}^{\mathsf{inf}} && : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Pi^{\mathsf{inf}}\,Ix\,b)) \to (i : Ix) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b\,i))\\
&\mathsf{lam}^{\mathsf{inf}} && : ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b\,i))) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Pi^{\mathsf{inf}}\,Ix\,b))\\
&\Pi && : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Ty}\,(\Gamma \triangleright \mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma\\
&\mathsf{app} && : \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) \to \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B\\
&\mathsf{lam} && : \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B)\\
&\Pi^{\mathsf{ext}} && : (Ix : \mathsf{Ty}_0) \to (Ix \to \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,\Gamma\\
&\mathsf{app}^{\mathsf{ext}} && : \mathsf{Tm}\,\Gamma\,(\Pi^{\mathsf{ext}}\,Ix\,B) \to (i : Ix) \to \mathsf{Tm}\,\Gamma\,(B\,i)\\
&\mathsf{lam}^{\mathsf{ext}} && : ((i : Ix) \to \mathsf{Tm}\,\Gamma\,(B\,i)) \to \mathsf{Tm}\,\Gamma\,(\Pi^{\mathsf{ext}}\,Ix\,B)
\end{aligned}
$$

Algebras

$$
\begin{array}{ll}
{-}^{A} & : \mathsf{Con} \to \mathsf{Set} \\
{-}^{A} & : \mathsf{Sub}\,\Gamma\,\Delta \to \Gamma^{A} \to \Delta^{A} \\
{-}^{A} & : \mathsf{Ty}\,\Gamma \to \Gamma^{A} \to \mathsf{Set} \\
{-}^{A} & : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \Gamma^{A}) \to A^{A}\,\gamma \\
\bullet^{A} & :\equiv \top \\
\epsilon^{A}\,\gamma & :\equiv \mathsf{tt} \\
\mathsf{id}^{A}\,\gamma & :\equiv \gamma \\
(\sigma \circ \delta)^{A}\,\gamma & :\equiv \sigma^{A}\,(\delta^{A}\,\gamma) \\
(\Gamma \triangleright A)^{A} & :\equiv (\gamma : \Gamma^{A}) \times A^{A}\,\gamma \\
(A[\sigma])^{A}\,\gamma & :\equiv A^{A}\,(\sigma^{A}\,\gamma) \\
(t[\sigma])^{A}\,\gamma & :\equiv t^{A}\,(\sigma^{A}\,\gamma) \\
\mathsf{p}^{A}\,(\gamma,\,\alpha) & :\equiv \gamma \\
\mathsf{q}^{A}\,(\gamma,\,\alpha) & :\equiv \alpha \\
(\sigma,\,t)^{A}\,\gamma & :\equiv (\sigma^{A}\,\gamma,\,t^{A}\,\gamma) \\
\mathsf{U}^{A}\,\gamma & :\equiv \mathsf{Ty}_0 \\
(\mathsf{El}\,a)^{A}\,\gamma & :\equiv a^{A}\,\gamma \\
\top^{A}\,\gamma & :\equiv \top_0 \\
\mathsf{tt}^{A}\,\gamma & :\equiv \mathsf{tt}_0 \\
(\Sigma\,a\,b)^{A}\,\gamma & :\equiv (\alpha : a^{A}\,\gamma) \times_0 b^{A}\,(\gamma,\,\alpha) \\
(\mathsf{proj}_1\,t)^{A}\,\gamma & :\equiv \mathsf{proj}_1\,(t^{A}\,\gamma) \\
(\mathsf{proj}_2\,t)^{A}\,\gamma & :\equiv \mathsf{proj}_2\,(t^{A}\,\gamma) \\
(t,\,u)^{A}\,\gamma & :\equiv (t^{A}\,\gamma,\,u^{A}\,\gamma) \\
(\mathsf{Id}\,t\,u)^{A}\,\gamma & :\equiv t^{A}\,\gamma = u^{A}\,\gamma \\
\mathsf{refl}^{A}\,\gamma & :\equiv \mathsf{refl} : t^{A}\,\gamma = t^{A}\,\gamma \\
(\Pi^{\mathsf{inf}}\,Ix\,b)^{A}\,\gamma & :\equiv (i : Ix) \to (b\,i)^{A}\,\gamma \\
(\mathsf{app}^{\mathsf{inf}}\,t\,i)^{A}\,\gamma & :\equiv t^{A}\,\gamma\,i \\
(\mathsf{lam}^{\mathsf{inf}}\,t)^{A}\,\gamma & :\equiv \lambda\,i.\,(t\,i)^{A}\,\gamma \\
(\Pi\,a\,B)^{A}\,\gamma & :\equiv (\alpha : a^{A}\,\gamma) \to B^{A}\,(\gamma,\,\alpha) \\
(\mathsf{app}\,t)^{A}\,(\gamma,\,\alpha) & :\equiv t^{A}\,\gamma\,\alpha \\
(\mathsf{lam}\,t)^{A}\,\gamma & :\equiv \lambda\,\alpha.\,t^{A}\,(\gamma,\,\alpha) \\
(\Pi^{\mathsf{ext}}\,Ix\,B)^{A}\,\gamma & :\equiv (i : Ix) \to (B\,i)^{A}\,\gamma \\
(\mathsf{app}^{\mathsf{ext}}\,t\,i)^{A}\,\gamma & :\equiv t^{A}\,\gamma\,i \\
(\mathsf{lam}^{\mathsf{ext}}\,t)^{A}\,\gamma & :\equiv \lambda\,i.\,(t\,i)^{A}\,\gamma
\end{array}
$$

Morphisms

$$
\begin{aligned}
&\_^M && : (\Gamma : \mathsf{Con}) \to \Gamma^A \to \Gamma^A \to \mathsf{Set} \\
&\_^M && : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \Gamma^M\,\gamma_0\,\gamma_1 \to \Delta^M\,(\sigma^A\,\gamma_0)\,(\sigma^A\,\gamma_1) \\
&\_^M && : (A : \mathsf{Ty}\,\Gamma) \to A^A\,\gamma_0 \to A^A\,\gamma_1 \to \Gamma^M\,\gamma_0\,\gamma_1 \to \mathsf{Set} \\
&\_^M && : (t : \mathsf{Tm}\,\Gamma\,A) \to (\gamma^M : \Gamma^M\,\gamma_0\,\gamma_1) \to A^M\,(t^A\,\gamma_0)\,(t^A\,\gamma_1)\,\gamma^M \\
&\bullet^M\,\gamma_0\,\gamma_1 && :\equiv \top \\
&\epsilon^M\,\gamma^M && :\equiv \mathsf{tt} \\
&\mathsf{id}^M\,\gamma^M && :\equiv \gamma^M \\
&(\sigma \circ \delta)^M\,\gamma^M && :\equiv \sigma^M\,(\delta^M\,\gamma^M) \\
&(\Gamma \triangleright A)^M\,(\gamma_0,\,\alpha_0)\,(\gamma_1,\,\alpha_1) && :\equiv (\gamma^M : \Gamma^M\,\gamma_0\,\gamma_1) \times A^M\,\alpha_0\,\alpha_1\,\gamma^M \\
&(A[\sigma])^M\,\alpha_0\,\alpha_1\,\gamma^M && :\equiv A^M\,\alpha_0\,\alpha_1\,(\sigma^M\,\gamma^M) \\
&(t[\sigma])^M\,\gamma^M && :\equiv t^M\,(\sigma^M\,\gamma^M) \\
&\mathsf{p}^M\,(\gamma^M,\,\alpha^M) && :\equiv \gamma^M \\
&\mathsf{q}^M\,(\gamma^M,\,\alpha^M) && :\equiv \alpha^M \\
&(\sigma,\,t)^M\,\gamma^M && :\equiv (\sigma^M\,\gamma^M,\,t^M\,\gamma^M) \\
&\mathsf{U}^M\,a_0\,a_1\,\gamma^M && :\equiv a_0 \to a_1 \\
&(\mathsf{El}\,a)^M\,\alpha_0\,\alpha_1\,\gamma^M && :\equiv a^M\,\gamma^M\,\alpha_0 \equiv \alpha_1 \\
&\top^M\,\gamma^M && :\equiv \lambda\,\_.\,\mathsf{tt}_0 \\
&\mathsf{tt}^M\,\gamma^M && :\equiv \mathsf{refl} \\
&(\Sigma\,a\,b)^M\,\gamma^M && :\equiv \lambda\,(\alpha,\,\beta).\,(a^M\,\gamma^M\,\alpha,\,b^M\,(\gamma^M,\,\mathsf{refl})\,\beta) \\
&(\mathsf{proj}_1\,t)^M\,\gamma^M && :\equiv \mathsf{refl} \\
&(\mathsf{proj}_2\,t)^M\,\gamma^M && :\equiv \mathsf{refl} \\
&(t,\,u)^M\,\gamma^M && :\equiv \mathsf{refl} \\
&(\mathsf{Id}\,t\,u)^M\,\gamma^M && :\equiv \lambda\,(p : t^A\,\gamma_0 = u^A\,\gamma_0).\,\mathsf{ap}\,(a^M\,\gamma^M)\,p \\
&\mathsf{refl}^M\,\gamma^M && :\equiv \mathsf{refl} \\
&(\Pi^{\mathsf{inf}}\,Ix\,b)^M\,\gamma^M && :\equiv \lambda\,t\,i.\,(b\,i)^M\,\gamma^M\,(t\,i) \\
&(\mathsf{app}^{\mathsf{inf}}\,t\,i)^M\,\gamma^M && :\equiv \mathsf{refl} \\
&(\mathsf{lam}^{\mathsf{inf}}\,t)^M\,\gamma^M && :\equiv \mathsf{refl} \\
&(\Pi\,a\,B)^M\,t_0\,t_1\,\gamma^M && :\equiv (\alpha : a^A\,\gamma_0) \to B^M\,(t_0\,\alpha)\,(t_1\,(a^M\,\gamma^M\,\alpha))\,(\gamma^M,\,\mathsf{refl}) \\
&(\mathsf{app}\,t)^M\,(\gamma^M,\,\alpha^M) && :\equiv t^M\,\gamma^M\,\alpha_0 \quad \text{where} \quad \alpha^M : a^M\,\gamma^M\,\alpha_0 \equiv \alpha_1 \\
&(\mathsf{lam}\,t)^M\,\gamma^M && :\equiv \lambda\,\alpha.\,t^M\,(\gamma^M,\,\mathsf{refl}) \quad \text{where} \quad \mathsf{refl} : a^M\,\gamma^M\,\alpha \equiv a^M\,\gamma^M\,\alpha \\
&(\Pi^{\mathsf{ext}}\,Ix\,B)^M\,t_0\,t_1\,\gamma^M && :\equiv (i : Ix) \to (B\,i)^M\,(t_0\,i)\,(t_1\,i)\,\gamma^M \\
&(\mathsf{app}^{\mathsf{ext}}\,t\,i)^M\,\gamma^M && :\equiv t^M\,\gamma^M\,i \\
&(\mathsf{lam}^{\mathsf{ext}}\,t)^M\,\gamma^M && :\equiv (t\,i)^M\,\gamma^M
\end{aligned}
$$

Displayed algebras

$$
\begin{array}{ll}
{\_}^D & : (\Gamma : \mathsf{Con}) \to \Gamma^A \to \mathsf{Set} \\
{\_}^D & : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \Gamma^D\,\gamma \to \Delta^D\,(\sigma^A\,\gamma) \\
{\_}^D & : (A : \mathsf{Ty}\,\Gamma) \to A^A\,\gamma \to \Gamma^D\,\gamma \to \mathsf{Set} \\
{\_}^D & : (t : \mathsf{Tm}\,\Gamma\,A) \to (\gamma^D : \Gamma^D\,\gamma) \to A^D\,(t^A\,\gamma)\,\gamma^D \\
\bullet^D\,\gamma & :\equiv \top \\
\epsilon^D\,\gamma^D & :\equiv \mathsf{tt} \\
\mathsf{id}^D\,\gamma^D & :\equiv \gamma^D \\
(\sigma \circ \delta)^D\,\gamma^D & :\equiv \sigma^D\,(\delta^D\,\gamma^D) \\
(\Gamma \triangleright A)^D\,(\gamma,\,\alpha) & :\equiv (\gamma^D : \Gamma^D\,\gamma) \times A^D\,\alpha\,\gamma^D \\
(A[\sigma])^D\,\alpha\,\gamma^D & :\equiv A^D\,\alpha\,(\sigma^D\,\gamma^D) \\
(t[\sigma])^D\,\gamma^D & :\equiv t^D\,(\sigma^D\,\gamma^D) \\
\mathsf{p}^D\,(\gamma^D,\,\alpha^D) & :\equiv \gamma^D \\
\mathsf{q}^D\,(\gamma^D,\,\alpha^D) & :\equiv \alpha^D \\
(\sigma,\,t)^D\,\gamma^D & :\equiv (\sigma^D\,\gamma^D,\,t^D\,\gamma^D) \\
\mathsf{U}^D\,a\,\gamma^D & :\equiv a \to \mathsf{Ty}_0 \\
(\mathsf{El}\,a)^D\,t\,\gamma^D & :\equiv a^D\,\gamma^D\,t \\
\top^D\,\gamma^D & :\equiv \lambda\,\_.\,\top_0 \\
\mathsf{tt}^D\,\gamma^D & :\equiv \mathsf{tt}_0 \\
(\Sigma\,a\,b)^D\,\gamma^D & :\equiv \lambda\,(\alpha,\,\beta).\,(\alpha^D : a^D\,\gamma^D\,\alpha) \times_0 b^D\,(\gamma^D,\,\alpha^D)\,\beta \\
(\mathsf{proj}_1\,t)^D\,\gamma^D & :\equiv \mathsf{proj}_1\,(t^D\,\gamma^D) \\
(\mathsf{proj}_2\,t)^D\,\gamma^D & :\equiv \mathsf{proj}_2\,(t^D\,\gamma^D) \\
(t,\,u)^D\,\gamma^D & :\equiv (t^D\,\gamma^D,\,u^D\,\gamma^D) \\
(\mathsf{Id}\,t\,u)^D\,\gamma^D & :\equiv \lambda\,(p : t^A\,\gamma = u^A\,\gamma).\,\mathsf{tr}_{(a^D\,\gamma^D)}\,p\,(t^D\,\gamma^D) = u^D\,\gamma^D \\
\mathsf{refl}^D\,\gamma^D & :\equiv \mathsf{refl} : t^D\,\gamma^D = t^D\,\gamma^D \\
(\Pi^{\mathsf{inf}}\,Ix\,b)^D\,\gamma^D & :\equiv \lambda\,t.\,(i : Ix) \to (b\,i)^D\,\gamma^D\,(t\,i) \\
(\mathsf{app}^{\mathsf{inf}}\,t\,i)^D\,\gamma^D & :\equiv t^D\,\gamma^D\,i \\
(\mathsf{lam}^{\mathsf{inf}}\,t)^D\,\gamma^D & :\equiv \lambda\,i.\,(t\,i)^D\,\gamma^D \\
(\Pi\,a\,B)^D\,t\,\gamma^D & :\equiv \{\alpha : a^A\,\gamma\}(\alpha^D : a^D\,\gamma^D\,\alpha) \to B^D\,(t\,\alpha)\,(\gamma^D,\,\alpha^D) \\
(\mathsf{app}\,t)^D\,(\gamma^D,\,\alpha^D) & :\equiv t^D\,\gamma^D\,\alpha^D \\
(\mathsf{lam}\,t)^D\,\gamma^D & :\equiv \lambda\,\{\alpha\}\,\alpha^D.\,t^D\,(\gamma^D,\,\alpha^D) \\
(\Pi^{\mathsf{ext}}\,Ix\,B)^D\,t\,\gamma^D & :\equiv (i : Ix) \to (B\,i)^D\,(t\,i)\,\gamma^D \\
(\mathsf{app}^{\mathsf{ext}}\,t\,i)^D\,\gamma^D & :\equiv t^D\,\gamma^D\,i \\
(\mathsf{lam}^{\mathsf{ext}}\,t)^D\,\gamma & :\equiv \lambda\,i.\,(t\,i)^D\,\gamma^D
\end{array}
$$

Sections

$$
\begin{array}{ll}
\_^S & : (\Gamma : \mathsf{Con}) \to (\gamma : \Gamma^A) \to \Gamma^A\,\gamma \to \mathsf{Set} \\
\_^S & : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \Gamma^S\,\gamma\,\gamma^D \to \Delta^S\,(\sigma^A\,\gamma)\,(\sigma^D\,\gamma^D) \\
\_^S & : (A : \mathsf{Ty}\,\Gamma) \to A^A\,\gamma \to A^D\,\gamma^D \to \Gamma^S\,\gamma\,\gamma^D \to \mathsf{Set} \\
\_^S & : (t : \mathsf{Tm}\,\Gamma\,A) \to (\gamma^S : \Gamma^S\,\gamma\,\gamma^D) \to A^S\,(t^A\,\gamma)\,(t^D\,\gamma^D)\,\gamma^S \\
\bullet^S\,\gamma\,\gamma^D & :\equiv \top \\
\epsilon^S\,\gamma^S & :\equiv \mathsf{tt} \\
\mathsf{id}^S\,\gamma^S & :\equiv \gamma^S \\
(\sigma \circ \delta)^S\,\gamma^S & :\equiv \sigma^S\,(\delta^S\,\gamma^S) \\
(\Gamma \triangleright A)^S\,(\gamma,\,\alpha)\,(\gamma^D,\,\alpha^D) & :\equiv (\gamma^S : \Gamma^S\,\gamma\,\gamma^D) \times A^S\,\alpha\,\alpha^D\,\gamma^S \\
(A[\sigma])^S\,\alpha\,\alpha^D\,\gamma^S & :\equiv A^S\,\alpha\,\alpha^D\,(\sigma^S\,\gamma^S) \\
(t[\sigma])^S\,\gamma^S & :\equiv t^S\,(\sigma^S\,\gamma^S) \\
\mathsf{p}^S\,(\gamma^S,\,\alpha^S) & :\equiv \gamma^S \\
\mathsf{q}^S\,(\gamma^S,\,\alpha^S) & :\equiv \alpha^S \\
(\sigma,\,t)^S\,\gamma^S & :\equiv (\sigma^S\,\gamma^S,\,t^S\,\gamma^S) \\
\mathsf{U}^S\,a\,a^D\,\gamma^S & :\equiv (\alpha : a) \to a^D\,\alpha \\
(\mathsf{El}\,a)^S\,\alpha\,\alpha^D\,\gamma^S & :\equiv a^S\,\gamma^S\,\alpha \equiv \alpha^D \\
\top^S\,\gamma^S & :\equiv \lambda\,\_.\,\mathsf{tt}_0 \\
\mathsf{tt}^S\,\gamma^S & :\equiv \mathsf{refl} \\
(\Sigma\,a\,b)^S\,\gamma^S & :\equiv \lambda\,(\alpha,\,\beta).\,(a^S\,\gamma^S\,\alpha,\,b^S\,(\gamma^S,\,\mathsf{refl})\,\beta) \\
(\mathsf{proj}_1\,t)^S\,\gamma^S & :\equiv \mathsf{refl} \\
(\mathsf{proj}_2\,t)^S\,\gamma^S & :\equiv \mathsf{refl} \\
(t,\,u)^S\,\gamma^S & :\equiv \mathsf{refl} \\
(\mathsf{Id}\,t\,u)^S\,\gamma^S & :\equiv \lambda\,(p : t^A\,\gamma = u^A\,\gamma).\,\mathsf{apd}\,(a^S\,\gamma^S)\,p \\
\mathsf{refl}^S\,\gamma^S & :\equiv \mathsf{refl} \\
(\Pi^{\mathsf{inf}}\,Ix\,b)^S\,\gamma^S & :\equiv \lambda\,t\,i.\,(b\,i)^S\,\gamma^S\,(t\,i) \\
(\mathsf{app}^{\mathsf{inf}}\,t\,i)^S\,\gamma^S & :\equiv \mathsf{refl} \\
(\mathsf{lam}^{\mathsf{inf}}\,t)^S\,\gamma^S & :\equiv \mathsf{refl} \\
(\Pi\,a\,B)^S\,t\,t^D\,\gamma^S & :\equiv (\alpha : a^A\,\gamma) \to B^S\,(t\,\alpha)\,(t^D\,(a^S\,\gamma^S\,\alpha))\,(\gamma^S,\,\mathsf{refl}) \\
(\mathsf{app}\,t)^S\,(\gamma^S,\,\alpha^S) & :\equiv t^S\,\gamma^S\,\alpha \quad \text{where} \quad \alpha^S : a^S\,\gamma^S\,\alpha \equiv \alpha^D \\
(\mathsf{lam}\,t)^S\,\gamma^S & :\equiv \lambda\,\alpha.\,t^S\,(\gamma^S,\,\mathsf{refl}) \quad \text{where} \quad \mathsf{refl} : a^S\,\gamma^S\,\alpha \equiv a^S\,\gamma^S\,\alpha \\
(\Pi^{\mathsf{ext}}\,Ix\,B)^S\,t\,t^D\,\gamma^S & :\equiv (i : Ix) \to (B\,i)^S\,(t\,i)\,(t^D\,i)\,\gamma^S \\
(\mathsf{app}^{\mathsf{ext}}\,t\,i)^S\,\gamma^S & :\equiv t^S\,\gamma^S\,i \\
(\mathsf{lam}^{\mathsf{ext}}\,t)^S\,\gamma^S & :\equiv (t\,i)^S\,\gamma^S
\end{array}
$$

Bibliography

[AAC+20] Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. A type and scope safe universe of syntaxes with binding: Their semantics and proofs. *CoRR*, abs/2001.11001, 2020. URL: `https://arxiv.org/abs/2001.11001`, `arXiv:2001.11001`.

[AAR+94] Jiří Adámek, J Adamek, J Rosicky, et al. *Locally presentable and accessible categories*, volume 189. Cambridge University Press, 1994.

[Abe13] Andreas Abel. *Normalization by Evaluation: Dependent Types and Impredicativity.* PhD thesis, Ludwig-Maximilians-Universität München, 2013. Habilitation thesis.

[ABKT19] Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. Setoid type theory - A syntactic translation. In Graham Hutton, editor, *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, volume 11825 of *Lecture Notes in Computer Science*, pages 155–196. Springer, 2019. `doi:10.1007/978-3-030-33636-3\_7`.

[ACD+18] Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. Quotient inductive-inductive types. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 293–310. Springer, 2018. `doi:10.1007/978-3-319-89366-2\_16`.

[ACKS19] Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *ArXiv e-prints*, may 2019. URL: http://arxiv.org/abs/1705.03307.

[Acz11] Peter Aczel. On voevodsky's univalence axiom. *Mathematical Logic: Proof Theory, Constructive Mathematics, Samuel R. Buss, Ulrich Kohlenbach, and Michael Rathjen (Eds.). Mathematisches Forschungsinstitut Oberwolfach, Oberwolfach*, page 2967, 2011.

[ADK17] Thorsten Altenkirch, Nils Anders Danielsson, and Nicolai Kraus. Partiality, revisited. In *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures - Volume 10203*, page 534–549, Berlin, Heidelberg, 2017. Springer-Verlag. doi:10.1007/978-3-662-54458-7_31.

[AHH18] Carlo Angiuli, Kuen-Bang Favonia Hou, and Robert Harper. Cartesian cubical computational type theory: Constructive reasoning with paths and equalities. *Computer Science Logic 2018*, 2018.

[AHW16] Carlo Angiuli, Robert Harper, and Todd Wilson. Computational higher type theory i: Abstract cubical realizability. *arXiv preprint arXiv:1604.08873*, 2016.

[AK79] Jirí Adámek and Václav Koubek. Least fixed point of a functor. *J. Comput. Syst. Sci.*, 19(2):163–178, 1979. doi:10.1016/0022-0000(79)90026-6.

[AK16] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodik and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.

[AKS15] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the rezk completion. *Math. Struct. Comput. Sci.*, 25(5):1010–1039, 2015. doi:10.1017/S0960129514000486.

[AL19] Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed Categories. *Logical Methods in Computer Science*, Volume 15, Issue 1, March 2019. URL: `https://lmcs.episciences.org/5252`, `doi:10.23638/LMCS-15(1:20)2019`.

[AMFS11] Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. A categorical semantics for inductive-inductive definitions. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *Algebra and Coalgebra in Computer Science - 4th International Conference, CALCO 2011, Winchester, UK, August 30 - September 2, 2011. Proceedings*, volume 6859 of *Lecture Notes in Computer Science*, pages 70–84. Springer, 2011. `doi:10.1007/978-3-642-22944-2\_6`.

[AMM19] Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. From signatures to monads in unimath. *J. Autom. Reason.*, 63(2):285–318, 2019. `doi:10.1007/s10817-018-9474-4`.

[AMS07] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *Proceedings of the 2007 workshop on Programming languages meets program verification*, pages 57–68. ACM, 2007.

[AÖV18] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL):23:1–23:29, 2018. `doi:10.1145/3158111`.

[AS20] Thorsten Altenkirch and Luis Scoccola. The integers as a higher inductive type. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 67–73. ACM, 2020. `doi:10.1145/3373718.3394760`.

[AVW17] Andreas Abel, Andrea Vezzosi, and Théo Winterhalter. Normalization by evaluation for sized dependent types. *Proc. ACM Program. Lang.*, 1(ICFP):33:1–33:30, 2017. `doi:10.1145/3110277`.

[Awo10] S. Awodey. *Category Theory*. Oxford Logic Guides. OUP Oxford, 2010. URL: `http://books.google.co.uk/books?id=-MCJ6x2lC7oC`.

[Awo18] Steve Awodey. Natural models of homotopy type theory. *Math. Struct. Comput. Sci.*, 28(2):241–286, 2018. doi:10.1017/S0960129516000268.

[Bar85] Michael Barr. Toposes, triples and theories. *Grundlehren der mathematischen Wissenschaften*, 278, 1985.

[BB85] Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed lambda-programs on term algebras. *Theor. Comput. Sci.*, 39:135–154, 1985. doi:10.1016/0304-3975(85)90135-5.

[BCH14] Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, 2014.

[BCM+20] Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Math. Struct. Comput. Sci.*, 30(2):118–138, 2020. doi:10.1017/S0960129519000197.

[BdB20] Guillaume Brunerie and Menno de Boer. Formalization of the initiality conjecture, 2020. URL: https://github.com/guillaumebrunerie/initiality.

[BH11] Patrick Bahr and Tom Hvitved. Compositional data types. In Jaakko Järvi and Shin-Cheng Mu, editors, *Proceedings of the seventh ACM SIGPLAN workshop on Generic programming, WGP@ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 83–94. ACM, 2011. doi:10.1145/2036918.2036930.

[BJP10] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In Paul Hudak and Stephanie Weirich, editors, *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*, pages 345–356. ACM, 2010. doi:10.1145/1863543.1863592.

[BKS21] Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. Induction principles for type theories, internally to presheaf categories. *arXiv preprint arXiv:2102.11649*, 2021.

[BPT17] Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. The next 700 syntactical models of type theory. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 182–194, New York, NY, USA, 2017. ACM. doi:10.1145/3018610.3018620.

[Bru19] Guillaume Brunerie. A formalization of the initiality conjecture in agda. Slides of a talk at the Homotopy Type Theory 2019 Conference, Carnegie Mellon University, Pittsburgh, Pennsylvania, August 2019. URL: https://guillaumebrunerie.github.io/pdf/initiality.pdf.

[Cap17] Paolo Capriotti. Models of type theory with strict equality. *arXiv preprint arXiv:1702.04912*, 2017.

[Car78] John Cartmell. *Generalised algebraic theories and contextual categories*. PhD thesis, Oxford University, 1978.

[Car86] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

[CCD17] Simon Castellan, Pierre Clairambault, and Peter Dybjer. Undecidability of equality in the free locally cartesian closed category (extended version). *Log. Methods Comput. Sci.*, 13(4), 2017. doi:10.23638/LMCS-13(4:22)2017.

[CCD19] Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019. URL: http://arxiv.org/abs/1904.00827, arXiv:1904.00827.

[CCHM17] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: A constructive interpretation of the univalence axiom. *FLAP*, 4(10):3127–3170, 2017. URL: http://collegepublications.co.uk/ifcolog/?00019.

[CD14] Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and martin-löf type theories. *Math. Struct. Comput. Sci.*, 24(6), 2014. doi:10.1017/S0960129513000881.

[CDMM10] James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. The gentle art of levitation. *ACM Sigplan Notices*, 45(9):3–14, 2010.

[CH19] Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. *Proc. ACM Program. Lang.*, 3(POPL), January 2019. doi:10.1145/3290314.

[CHM18] Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 255–264. ACM, 2018. doi:10.1145/3209108.3209197.

[CKS07] Jacques Carette, Oleg Kiselyov, and Chung-Chieh Shan. Finally tagless, partially evaluated. In *Asian Symposium on Programming Languages and Systems*, pages 222–238. Springer, 2007.

[Con19] Initiality Project Contributors. Initiality project, 2019. URL: https://ncatlab.org/nlab/show/Initiality+Project.

[Coq18] Thierry Coquand. Canonicity and normalisation for dependent type theory. *arXiv preprint arXiv:1810.09367*, 2018.

[CS] Paolo Capriotti and Christian Sattler. Higher categories of algebras for higher inductive definitions. *EUTYPES-TYPES 2020-Abstracts*.

[Dag17] Pierre-Évariste Dagand. The essence of ornaments. *J. Funct. Program.*, 27:e9, 2017. doi:10.1017/S0956796816000356.

[dB20] Menno de Boer. *A Proof and Formalization of the Initiality Conjecture of Dependent Type Theory*. PhD thesis, Department of Mathematics, Stockholm University, 2020.

[dev17] Agda developers. Agda issue 2820, 2017. URL: https://github.com/agda/agda/issues/2820.

[DM18] Peter Dybjer and Hugo Moeneclaey. Finitary higher inductive types in the groupoid model. *Electronic Notes in Theoretical Computer Science*, 336:119 – 134, 2018. The Thirty-third Conference on the Math-

ematical Foundations of Programming Semantics (MFPS XXXIII). doi:10.1016/j.entcs.2018.03.019.

[DS99] Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 1999. doi:10.1007/3-540-48959-2_11.

[dVL14] Edsko de Vries and Andres Löh. True sums of products. In José Pedro Magalhães and Tiark Rompf, editors, *Proceedings of the 10th ACM SIGPLAN workshop on Generic programming, WGP 2014, Gothenburg, Sweden, August 31, 2014*, pages 83–94. ACM, 2014. doi:10.1145/2633628.2633634.

[Dyb94] Peter Dybjer. Inductive families. *Formal Aspects Comput.*, 6(4):440–465, 1994. doi:10.1007/BF01211308.

[Dyb95] Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 1995. doi:10.1007/3-540-61780-9_66.

[FPS20] Marcelo P. Fiore, Andrew M. Pitts, and S. C. Steenkamp. Constructing infinitary quotient-inductive types. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 257–276. Springer, 2020. doi:10.1007/978-3-030-45231-5_14.

[Fre72] Peter Freyd. Aspects of topoi. *Bulletin of the Australian Mathematical Society*, 7(1):1–76, 1972.

[GKNB20] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 492–506. ACM, 2020. doi:10.1145/3373718.3394736.

[GU06] Peter Gabriel and Friedrich Ulmer. *Lokal präsentierbare kategorien*, volume 221. Springer-Verlag, 2006.

[Hed98] Michael Hedberg. A coherence theorem for martin-löf's type theory. *J. Funct. Program.*, 8(4):413–436, 1998. URL: http://journals.cambridge.org/action/displayAbstract?aid=44199.

[Hof95] Martin Hofmann. Extensional concepts in intensional type theory. 1995.

[Hof97] Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.

[HRR14] Claudio Hermida, Uday S. Reddy, and Edmund P. Robinson. Logical relations and parametricity - A reynolds programme for category theory and programming languages. *Electron. Notes Theor. Comput. Sci.*, 303:149–180, 2014. doi:10.1016/j.entcs.2014.02.008.

[HS96] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *In Venice Festschrift*, pages 83–111. Oxford University Press, 1996.

[Hub16] Simon Huber. *Cubical Interpretations of Type Theory*. PhD thesis, University of Gothenburg, 2016.

[Hug21] Jasper Hugunin. Why not W?, 2021. URL: https://jashug.github.io/papers/whynotw.pdf.

[Joh02] Peter T Johnstone. *Sketches of an elephant: A topos theory compendium*, volume 2. Oxford University Press, 2002.

[JVWW06] Simon L. Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Geoffrey Washburn. Simple unification-based type inference for gadts. In John H. Reppy and Julia L. Lawall, editors, *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming, ICFP 2006, Portland, Oregon, USA, September 16-21, 2006*, pages 50–61. ACM, 2006. doi:10.1145/1159803.1159811.

[KHS19] Ambrus Kaposi, Simon Huber, and Christian Sattler. Gluing for type theory. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 25:1–25:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.25.

[Kis14] Oleg Kiselyov. The design and implementation of BER metaocaml - system description. In Michael Codish and Eijiro Sumii, editors, *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*, volume 8475 of *Lecture Notes in Computer Science*, pages 86–102. Springer, 2014. doi:10.1007/978-3-319-07151-0_6.

[KK18] Ambrus Kaposi and András Kovács. A Syntax for Higher Inductive-Inductive Types. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, volume 108 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9190, doi:10.4230/LIPIcs.FSCD.2018.20.

[KK20a] Ambrus Kaposi and András Kovács. Signatures and induction principles for higher inductive-inductive types. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:10)2020.

[KK20b] András Kovács and Ambrus Kaposi. Large and infinitary quotient inductive-inductive types. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken,*

Germany, July 8-11, 2020, pages 648–661. ACM, 2020. doi:10.1145/3373718.3394770.

[KKA19] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, 2019. doi:10.1145/3290315.

[KKL19] Ambrus Kaposi, András Kovács, and Ambroise Lafont. For finitary induction-induction, induction is enough. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, volume 175 of *LIPIcs*, pages 6:1–6:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.TYPES.2019.6.

[KLV12] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations, 2012. http://arxiv.org/abs/1211.2851/arXiv:1211.2851.

[Kov20] András Kovács. Program for checking higher inductive-inductive signatures, 2020. URL: https://github.com/akaposi/hiit-signatures/tree/master/elims-demo.

[Kov21] András Kovács. Generalized universe hierarchies and first-class universe levels, 2021. arXiv:2103.00223.

[KvR20] Ambrus Kaposi and Jakob von Raumer. A syntax for mutual inductive families. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 23:1–23:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.23.

[LM11] Andres Löh and José Pedro Magalhães. Generic programming with indexed functors. In Jaakko Järvi and Shin-Cheng Mu, editors, *Proceedings of the seventh ACM SIGPLAN workshop on Generic programming, WGP@ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 1–12. ACM, 2011. doi:10.1145/2036918.2036920.

[LS] Peter LeFanu Lumsdaine and Michael Shulman. Semantics of higher inductive types. *Mathematical Proceedings of the Cambridge Philosophical Society*, page 1–50. doi:10.1017/S030500411900015X.

[LW15] Peter LeFanu Lumsdaine and Michael A. Warren. The local universes model: An overlooked coherence construction for dependent type theories. *ACM Trans. Comput. Log.*, 16(3):23:1–23:31, 2015. doi:10.1145/2754931.

[MAA+19] Kenji Maillard, Danel Ahman, Robert Atkey, Guido Martínez, Catalin Hritcu, Exequiel Rivas, and Éric Tanter. Dijkstra monads for all. *Proc. ACM Program. Lang.*, 3(ICFP):104:1–104:29, 2019. doi:10.1145/3341708.

[MFP91] Erik Meijer, Maarten M. Fokkinga, and Ross Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In John Hughes, editor, *Functional Programming Languages and Computer Architecture, 5th ACM Conference, Cambridge, MA, USA, August 26-30, 1991, Proceedings*, volume 523 of *Lecture Notes in Computer Science*, pages 124–144. Springer, 1991. doi:10.1007/3540543961_7.

[ML84] Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

[ML98] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer, 2nd edition, September 1998. URL: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0387984038.

[MM12] Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer Science & Business Media, 2012.

[nc21] nLab contributors. Internal logic, 2021. URL: https://ncatlab.org/nlab/show/internal+logic.

[NF13] Fredrik Nordvall Forsberg. *Inductive-inductive definitions*. PhD thesis, Swansea University, 2013.

[PV07] Erik Palmgren and Steven J. Vickers. Partial horn logic and cartesian categories. *Ann. Pure Appl. Log.*, 145(3):314–353, 2007. doi:10.1016/j.apal.2006.10.001.

[PWK19] Matthew Pickering, Nicolas Wu, and Csongor Kiss. Multi-stage programs in context. In Richard A. Eisenberg, editor, *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell, Haskell@ICFP 2019, Berlin, Germany, August 18-23, 2019*, pages 71–84. ACM, 2019. doi:10.1145/3331545.3342597.

[SA21] Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–15. IEEE, 2021. doi:10.1109/LICS52264.2021.9470719.

[SAG20] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. A cubical language for bishop sets. *CoRR*, abs/2003.01491, 2020. URL: https://arxiv.org/abs/2003.01491, arXiv:2003.01491.

[Sch17] Gabriel Scherer. Deciding equivalence with sums and the empty type. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 374–386. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3009901.

[Shu14] Michael Shulman. Homotopy type theory should eat itself (but so far, it's too big to swallow), 2014. URL: https://homotopytypetheory.org/2014/03/03/hott-should-eat-itself/.

[Soj15a] Kristina Sojakova. Higher inductive types as homotopy-initial algebras. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 31–42. ACM, 2015. doi:10.1145/2676726.2676983.

[Soj15b] Kristina Sojakova. Higher inductive types as homotopy-initial algebras. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15,

pages 31–42, New York, NY, USA, 2015. ACM. doi:10.1145/2676726.2676983.

[Ste19] Jonathan Sterling. Algebraic type theory and universe hierarchies. *CoRR*, abs/1902.08848, 2019. URL: http://arxiv.org/abs/1902.08848, arXiv:1902.08848.

[Str93] Thomas Streicher. Investigations into intensional type theory. habilitation thesis, 1993. http://www.mathematik.tu-darmstadt.de/~streicher/HabilStreicher.pdf.

[Swi08] Wouter Swierstra. Data types à la carte. *J. Funct. Program.*, 18(4):423–436, 2008. doi:10.1017/S0956796808006758.

[TS18] Amin Timany and Matthieu Sozeau. Cumulative inductive types in coq. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSCD.2018.29.

[Uem19] Taichi Uemura. A general framework for the semantics of type theory. *CoRR*, abs/1904.04097, 2019. URL: http://arxiv.org/abs/1904.04097, arXiv:1904.04097.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[VG15] Tamara Von Glehn. *Polynomials and models of type theory*. PhD thesis, University of Cambridge, 2015.

[VMA21] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: A dependently typed programming language with univalence and higher inductive types. *J. Funct. Program.*, 31:e8, 2021. doi:10.1017/S0956796821000034.

[Voe17] Vladimir Voevodsky. Models, Interpretations and the Initiality Conjectures. August 18, 2017. A talk the Special session

on category theory and type theory in honor of Per Martin-Löf on his 75th birthday, August 17–19, 2017, during the Logic Colloquium 2017, August 14-20, in Stockholm. URL: https://www.math-stockholm.se/konferenser-och-akti/logic-in-stockholm-2/logic-colloquium-201/logic-colloquium-2017-august-14-20-1.717648.

[WB18] Pawel Wieczorek and Dariusz Biernacki. A coq formalization of normalization by evaluation for martin-löf type theory. In June Andronick and Amy P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 266–279. ACM, 2018. doi:10.1145/3167091.

[Win20] Théo Winterhalter. *Formalisation and Meta-Theory of Type Theory*. PhD thesis, University of Nantes, 2020.

[YHLJ09] Alexey Rodriguez Yakushev, Stefan Holdermans, Andres Löh, and Johan Jeuring. Generic programming with fixed points for mutually recursive datatypes. In Graham Hutton and Andrew P. Tolmach, editors, *Proceeding of the 14th ACM SIGPLAN international conference on Functional programming, ICFP 2009, Edinburgh, Scotland, UK, August 31 - September 2, 2009*, pages 233–244. ACM, 2009. doi:10.1145/1596550.1596585.