SIGNATURES AND INDUCTION PRINCIPLES FOR HIGHER INDUCTIVE-INDUCTIVE TYPES

AMBRUS KAPOSI AND ANDRÁS KOVÁCS

Department of Programming Languages and Compilers, Eötvös Loránd University, Budapest, Hungary
e-mail address: akaposi@inf.elte.hu

Department of Programming Languages and Compilers, Eötvös Loránd University, Budapest, Hungary
e-mail address: kovacsandras@inf.elte.hu

Abstract. Higher inductive-inductive types (HIITs) generalize inductive types of dependent type theories in two ways. On the one hand they allow the simultaneous definition of multiple sorts that can be indexed over each other. On the other hand they support equality constructors, thus generalizing higher inductive types of homotopy type theory. Examples that make use of both features are the Cauchy real numbers and the well-typed syntax of type theory where conversion rules are given as equality constructors. In this paper we propose a general definition of HIITs using a small type theory, named the theory of signatures. A context in this theory encodes a HIIT by listing the constructors. We also compute notions of induction and recursion for HIITs, by using variants of syntactic logical relation translations. Building full categorical semantics and constructing initial algebras is left for future work. The theory of HIIT signatures was formalised in Agda together with the syntactic translations. We also provide a Haskell implementation, which takes signatures as input and outputs translation results as valid Agda code.

1. Introduction

Many dependent type theories support some form of inductive types. An inductive type is given by its constructors, along with an elimination principle which expresses that all inhabitants are constructed using finitely many applications of the constructors.

For example, the inductive type of natural numbers Nat is given by the constructors $\mathsf{zero} : \mathsf{Nat}$ and $\mathsf{suc} : \mathsf{Nat} \to \mathsf{Nat}$, and has the well-known induction principle:

$$\mathsf{ElimNat} : (P : \mathsf{Nat} \to \mathsf{Type})(pz : P\,\mathsf{zero})\big(ps : (n : \mathsf{Nat}) \to P\,n \to P\,(\mathsf{suc}\,n)\big)(n : \mathsf{Nat}) \to P\,n$$

P is a family of types (or: a proof-relevant predicate) over natural numbers, which is called the *induction motive*. The arguments pz and ps are called the *induction methods*. The

Key words and phrases: homotopy type theory, higher inductive types, inductive types.

* This is a journal version of the paper [29].

This work was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002) and COST Action EUTypes CA15123.

behavior of induction is described by a *computation rule* (β-rule) for each constructor and induction method:

$$\begin{aligned}\mathsf{ElimNat}\,P\,pz\,ps\,\mathsf{zero} &\equiv pz \\ \mathsf{ElimNat}\,P\,pz\,ps\,(\mathsf{suc}\,n) &\equiv ps\,n\,(\mathsf{ElimNat}\,P\,pz\,ps\,n)\end{aligned}$$

Indexed families of types can be also considered, for example length-indexed vectors of A-elements $\mathsf{Vec}_A : \mathsf{Nat} \to \mathsf{Type}$. Mutual inductive types are yet another generalization, however, they can be reduced to indexed families where indices classify constructors for each mutual type. Inductive-inductive types [35] are mutual definitions where this reduction does not work: here a type can be defined together with a family indexed over it. An example is the following fragment of a well-typed syntax of a type theory, where the second Ty type constructor is indexed over Con, but constructors of Con also refer to Ty:

Con	$: \mathsf{Type}$	contexts
Ty	$: \mathsf{Con} \to \mathsf{Type}$	types in contexts
$\bullet$	$: \mathsf{Con}$	constructor for the empty context
$- \rhd -$	$: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$	constructor for context extension
ι	$: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma$	constructor for a base type
Π	$: (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \rhd A) \to \mathsf{Ty}\,\Gamma$	constructor for dependent functions

There are two eliminators for this type: one for Con and one for Ty. Both take the same arguments: two motives ($P : \mathsf{Con} \to \mathsf{Type}$ and $Q : (\Gamma : \mathsf{Con}) \to P\,\Gamma \to \mathsf{Ty}\,\Gamma \to \mathsf{Type}$) and four methods (one for each constructor, which we omit).

$$\begin{aligned}\mathsf{ElimCon} &: (P : \dots)(Q : \dots) \to \dots \to (\Gamma : \mathsf{Con}) \;\; \to P\,\Gamma \\ \mathsf{ElimTy} &: (P : \dots)(Q : \dots) \to \dots \to (A : \mathsf{Ty}\,\Gamma) \to Q\,\Gamma\,(\mathsf{ElimCon}\,\Gamma)\,A\end{aligned}$$

Note that the type of ElimTy refers to $\mathsf{ElimCon}$; for this reason this elimination principle is sometimes called “recursive-recursive” (analogously to “inductive-inductive”).

Higher inductive types (HITs, [38, Chapter 6]) generalize inductive types in a different way: they allow constructors expressing equalities of elements of the type being defined. This enables, among others, the definition of types quotiented by a relation. For example, the type of integers Int can be given by a constructor $\mathsf{pair} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Int}$ and an equality constructor $\mathsf{eq} : (a\,b\,c\,d : \mathsf{Nat}) \to a + d =_{\mathsf{Nat}} b + c \to \mathsf{pair}\,a\,b =_{\mathsf{Int}} \mathsf{pair}\,c\,d$ targetting an equality of Int. The eliminator for Int expects a motive $P : \mathsf{Int} \to \mathsf{Type}$, a method for the pair constructor $p : (a\,b : \mathsf{Nat}) \to P\,(\mathsf{pair}\,a\,b)$ and a method for the equality constructor path. This method is a proof that given $e : a + d =_{\mathsf{Nat}} b + c$, $p\,a\,b$ is equal to $p\,c\,d$ (the types of which are equal by e). Thus the method for the equality constructor ensures that all functions defined from the quotiented type respect the relation. Since the integers are supposed to be a set (which means that any two equalities between the same two integers are equal), we would need an additional higher equality constructor $\mathsf{trunc} : (x\,y : \mathsf{Int}) \to (p\,q : x =_{\mathsf{Int}} y) \to p =_{x =_{\mathsf{Int}} y} q$. HITs may have constructors of iterated equality types as well. With the view of types as spaces in mind, point constructors add points to spaces, equality constructors add paths and higher constructors add homotopies between higher-dimensional paths.

Not all constructor expressions make sense. For example [38, Example 6.13.1], given an $f : (X : \mathsf{Type}) \to X \to X$, suppose that an inductive type Ival is generated by the point constructors $\mathsf{a} : \mathsf{Ival}$, $\mathsf{b} : \mathsf{Ival}$ and a path constructor $\sigma : f\,\mathsf{Ival}\,\mathsf{a} =_{\mathsf{Ival}} f\,\mathsf{Ival}\,\mathsf{b}$. The eliminator for this type should take a motive $P : \mathsf{Ival} \to \mathsf{Type}$, two methods $p_a : P\,\mathsf{a}$ and $p_b : P\,\mathsf{b}$, and a path connecting elements of $P\,(f\,\mathsf{lval}\,\mathsf{a})$ and $P\,(f\,\mathsf{lval}\,\mathsf{b})$. However it is not clear what these elements should be: we only have elements $p_a : P\,\mathsf{a}$ and $p_b : P\,\mathsf{b}$, and there is no way in general to transform these to have types $P\,(f\,\mathsf{lval}\,\mathsf{a})$ and $P\,(f\,\mathsf{lval}\,\mathsf{b})$.

Another invalid example is an inductive type Neg with a constructor $\mathsf{con} : (\mathsf{Neg} \to \bot) \to \mathsf{Neg}$ where $\bot$ is the empty type. An eliminator for this type should (at least) yield a projection function $\mathsf{proj} : \mathsf{Neg} \to (\mathsf{Neg} \to \bot)$. Given this, we can define $u :\equiv \mathsf{con}\,(\lambda x.\mathsf{proj}\,x\,x) : \mathsf{Neg}$ and then derive $\bot$ by $\mathsf{proj}\,u\,u$. The existence of Neg would make the type theory inconsistent. A common restriction to avoid such situations is *strict positivity*. It means that the type being defined cannot occur on the left hand side of a function arrow in a parameter of a constructor. This excludes the above constructor con.

In this paper we propose a notion of signatures for higher inductive-inductive types (HIITs) which includes the above valid examples and excludes the invalid ones. Our signatures allow any number of inductive-inductive type constructors, possibly infinitary higher constructors of any dimension and restricts constructors to strictly positive ones. It also allows free usage of J (path induction) and refl in HIIT signatures, and allows mixing type, point and path constructors in any order.

The core idea is to represent HIIT specifications as contexts in a domain-specific type theory which we call the *theory of signatures*. Type formers in the theory of signatures are restricted in order to enforce strict positivity. For example, natural numbers are defined as the three-element context

$$Nat : \mathsf{U},\ \ zero : \underline{Nat},\ \ suc : Nat \to \underline{Nat}$$

where Nat, $zero$ and suc are simply variable names, and underlining denotes El (decoding) for the Tarski-style universe U.

We also show how to derive induction and recursion principles for each signature. We use variants of *syntactic logical relation translations* to compute notions of *algebras*, *homomorphisms*, *displayed algebras* and *displayed algebra sections*, and then define induction and recursion in terms of these notions.

To our knowledge, this is the first proposal for a definition of HIITs. However, we do not provide complete (higher) categorical semantics for HIITs, nor do we show that initial algebras exist for specified HIITs.

The present paper is an expanded version of our conference paper [29]. In this version, we additionally compute notions of homomorphisms from signatures, and compare them to logical relations, in Section 7. We also explain a coherence problem in interpreting syntaxes of type theories, and how this influenced the current paper, in Section 4 and Section 9.

1.1. **Overview of the Paper.**

We start by describing the theory of HIIT signatures in Section 2. Here, we also describe the syntax for an external type theory, which serves as the source of constants which are external to a signature, like natural numbers in the case of length-indexed vectors. In Section 3, we give a general definition of induction and recursion. In Section 4 we explain the choice of using syntactic translations in the rest of the paper. In Sections 5 to 8, we describe four syntactic translations from the theory of signatures to the syntax of the external type theory, respectively computing algebras, displayed algebras, homomorphisms, and sections of displayed algebras. In Section 9, we consider extending the previous translations with additional components of a categorical semantics (e.g. identity and composition for homomorphisms), and explain why the approach in this paper does not make this feasible. Section 10 describes the Agda formalization and the Haskell implementation. We conclude in Section 11.

1.2. **Related Work.** Schemes for inductive families are given in [20, 37], and for inductive-recursive types in [21]. A symmetric scheme for both inductive and coinductive types is given in [9]. Basold et al. [10] define a syntactic scheme for higher inductive types with only 0-constructors and compute the types of induction principles. In [42] a semantics is given for the same class of HITs but with no recursive equality constructors. Dybjer and Moeneclaey define a syntactic scheme for finitary HITs and show their existence in a groupoid model [22].

Internal codes for simple inductive types such as natural numbers, lists or binary trees can be given by containers which are decoded to W-types [1]. Morris and Altenkirch [34] extend the notion of container to that of indexed container which specifies indexed inductive types. Codes for inductive-recursive types are given in [23]. Inductive-inductive types were introduced by Forsberg [35]. Sojakova [40] defines a subset of HITs called W-suspensions by a coding scheme similar to W-types. She proves that the induction principle is equivalent to homotopy initiality.

Quotient types [27] are precursors of higher inductive types (HITs). The notion of HIT first appeared in [38], however only through examples and without a general definition. Lumsdaine and Shulman give a general specification of models of type theory supporting higher inductive types [33]. They introduce the notion of cell monad with parameters and characterize the class of models which have initial algebras for a cell monad with parameters. [17] develop semantics for several HITs (sphere, torus, suspensions, truncations, pushouts) in certain presheaf toposes, and extend the syntax of cubical type theory [16] with these HITs. Kraus [31] and Van Doorn [18] construct propositional truncation as a sequential colimit. The schemes mentioned so far do not support inductive-inductive types.

Cartmell's generalized algebraic theories (GATs) [15] pioneered a type-theoretic notion of algebraic signature. GATs can be viewed as finitary quotient inductive-inductive signatures (QIITs), although GATs support equalities between inductive types (sorts) as well, which have not been so far explored in other works about QIITs and HIITs.

The article of Altenkirch et al. [5] gives specification and semantics of QIITs in a set-truncated setting. Signatures are given as lists of functors which can be interpreted as complete categories of algebras, and completeness is used to talk about notions of induction and recursion. However, no strict positivity restriction is given, nor a construction of initial algebras.

Closely related to the current work is the paper by the current authors and Altenkirch [30], which also concerns QIITs. There, signatures for QIITs are essentially a restriction of the signatures given here, but in contrast to the current work, the restricted quotient setting enables building initial algebras and detailed categorical semantics.

The logical predicate syntactic translation was introduced by Bernardy et al. [11]. The idea that a context can be seen as a signatures and the logical predicate translation can be used to derive the types of induction motives and methods was described in [6, Section 5.3]. Logical relations are used to derive the computation rules in [28, Section 4.3], but only for closed QIITs. Syntactic translations in the context of the calculus of inductive constructions are discussed in [13]. Logical relations and parametricity can also be used to justify the existence of inductive types in a type theory with an impredicative universe [8].

2. Signatures for HIITs

In this section we define signatures for HIITs. First, we list the main motivations behind our definition.

- *Ubiquitous type dependencies.* Recall the inductive-inductive Con-Ty example from Section 1: there, types of constructors may refer back to previous constructors. Additionally, Ty is indexed over the previously declared Con type constructor. This suggests that we should not attempt to stratify signatures, and instead use a fully dependent type theory. At this level of generality, stratification seems to complicate matters and remove the syntax further from familiar type theories.
- *Referring to external types.* We would like to mention types which are external to the signature. For example, length-indexed vectors refer to natural numbers which are supposed to already exist. Hence, we also assume a syntax for an *external type theory*, which is the source of such types, and constructions in the theory of signatures may depend on a context in the external type theory.
- *Strict positivity.* In prior literature, schemes for inductive types usually include structural restrictions for this. In our case, a *universe* can be used to make size restrictions which also entail strict positivity.
- *Iterated equalities and path induction in signatures.* We can support this relatively easily by closing the universe under equality type formation, and also using a standard (although size-restricted) definition of path induction.

2.1. Theory of Signatures.

Theory of Signatures. We list typing rules for the theory of signatures in Figure 1. We consider the following judgments:

$\Gamma \vdash \Delta$	Δ is a context in the external context Γ
$\Gamma; \Delta \vdash A$	A is a type in context Δ and external context Γ
$\Gamma; \Delta \vdash t : A$	t is a term of type A in context Δ and external context Γ

We have the convention that constructions in the external type theory are notated in brick red color. Although every judgement is valid up to a context in the external type theory, note that none of the rules in (1)–(4) depend on or change these assumptions, and even after that, we do not refer to any particular type former from the external theory. We describe the external theory in more detail in Section 2.2. Also, the rules presented here are informal and optimized for readability; we describe the Agda formalizations in Section 10. We explain the rules for the theory of signatures in order below.

(1) The rules for context formation and variables are standard. We build signatures in a well-formed external context. We assume fresh names everywhere to avoid name capture, and leave weakenings implicit.

(2) There is a universe U, with decoding written as an underline instead of usual El, to improve readability. With this part of the syntax, we can already define contexts specifying the empty type, unit type and booleans, or in general, finite sets of finite sets:

$$\cdot,\ \mathit{Empty} : \mathsf{U} \qquad \cdot,\ \mathit{Unit} : \mathsf{U},\ \mathit{tt} : \underline{\mathit{Unit}} \qquad \cdot,\ \mathit{Bool} : \mathsf{U},\ \mathit{true} : \underline{\mathit{Bool}},\ \mathit{false} : \underline{\mathit{Bool}}$$

(3) We have a function space with small domain and large codomain, which we call the inductive function space. This can be used to add inductive parameters to all kinds of constructors. As U is not closed under this function space, these function types cannot

(1) Contexts and variables

$$\frac{\vdash \Gamma}{\Gamma \vdash \cdot} \qquad \frac{\Gamma;\Delta \vdash A}{\Gamma \vdash \Delta, x : A} \qquad \frac{\Gamma;\Delta\vdash A}{\Gamma;\Delta,x:A\vdash x:A} \qquad \frac{\Gamma;\Delta\vdash x:A \qquad \Gamma;\Delta\vdash B}{\Gamma;\Delta,y:B\vdash x:A}$$

(2) Universe

$$\frac{\Gamma; \vdash \Delta}{\Gamma;\Delta\vdash \mathsf{U}} \qquad \frac{\Gamma;\Delta\vdash a:\mathsf{U}}{\Gamma;\Delta\vdash \underline{a}}$$

(3) Inductive parameters

$$\frac{\Gamma;\Delta\vdash a:\mathsf{U}\qquad \Gamma;\Delta,x:\underline{a}\vdash B}{\Gamma;\Delta\vdash (x:a)\to B} \qquad \frac{\Gamma;\Delta\vdash t:(x:a)\to B \qquad \Gamma;\Delta\vdash u:\underline{a}}{\Gamma;\Delta\vdash t\,u : B[x\mapsto u]}$$

(4) Equality

$$\frac{\Gamma;\Delta\vdash a:\mathsf{U} \qquad \Gamma;\Delta\vdash t:\underline{a} \qquad \Gamma;\Delta\vdash u:\underline{a}}{\Gamma;\Delta\vdash t =_a u : \mathsf{U}} \qquad \frac{\Gamma;\Delta\vdash t:\underline{a}}{\Gamma;\Delta\vdash \mathsf{refl} : \underline{t =_a t}}$$

$$\frac{\begin{array}{l}\Gamma;\Delta\vdash t:\underline{a}\\ \Gamma;\Delta,x:\underline{a},z:\underline{t =_a x}\vdash p:\mathsf{U}\\ \Gamma;\Delta\vdash pr:\underline{p[x\mapsto t, z\mapsto \mathsf{refl}]}\\ \Gamma;\Delta\vdash u:\underline{a}\\ \Gamma;\Delta\vdash eq:\underline{t =_a u}\end{array}}{\Gamma;\Delta\vdash \mathsf{J}_{a\,t\,(x.z.p)}\,pr\,{}_u\,eq : \underline{p[x\mapsto u, z\mapsto eq]}}$$

$$\frac{\Gamma;\Delta\vdash t:\underline{a} \qquad \Gamma;\Delta,x:\underline{a},z:\underline{t =_a x}\vdash p:\mathsf{U} \qquad \Gamma;\Delta\vdash pr:\underline{p[x\mapsto t, z\mapsto \mathsf{refl}]}}{\Gamma;\Delta\vdash \mathsf{J}\beta_{a\,t\,(x.z.p)}\,pr : \underline{(\mathsf{J}_{a\,t\,(x.z.p)}\,pr\,{}_t\,\mathsf{refl}) =_{p[x\mapsto t, z\mapsto \mathsf{refl}]} pr}}$$

(5) External parameters

$$\frac{\Gamma\vdash A:\mathsf{Type}_0 \qquad \Gamma;\vdash\Delta \qquad (\Gamma,x:A);\Delta\vdash B}{\Gamma;\Delta\vdash (x:A)\to B}$$

$$\frac{\Gamma;\Delta\vdash t:(x:A)\to B\qquad \Gamma\vdash u:A}{\Gamma;\Delta\vdash t\,u : B[x\mapsto u]}$$

(6) Infinitary parameters

$$\frac{\Gamma\vdash A:\mathsf{Type}_0 \qquad \Gamma;\vdash\Delta \qquad (\Gamma,x:A);\Delta\vdash b:\mathsf{U}}{\Gamma;\Delta\vdash (x:A)\to b:\mathsf{U}}$$

$$\frac{\Gamma;\Delta\vdash t:\underline{(x:A)\to b}\qquad \Gamma\vdash u:A}{\Gamma;\Delta\vdash t\,u : \underline{b[x\mapsto u]}}$$

Figure 1: The theory of HIIT signatures. Weakenings are implicit, we assume fresh names everywhere and consider α-convertible terms equal. The Γ; assumptions are not used or changed in parts (1)–(4).

(recursively) appear in inductive arguments, which ensures strict positivity. When the codomain does not depend on the domain, $a \to B$ can be written instead of $(x : a) \to B$.

Now we can specify the natural numbers as a context:

$$\cdot,\ \ Nat : \mathsf{U},\ \ zero : \underline{Nat},\ \ suc : Nat \to \underline{Nat}$$

We can also encode inductive-inductive definitions such as the fragment of the well-typed syntax of a type theory mentioned in the introduction:

$$\cdot,\ \ Con : \mathsf{U},\ \ Ty : Con \to \mathsf{U},\ \ \bullet : \underline{Con},\ \ - \rhd - : (\Delta : Con) \to Ty\,\Delta \to \underline{Con},$$
$$U : (\Delta : Con) \to \underline{\mathsf{Ty}\,\Delta},\ \ \Pi : (\Delta : Con)(A : Ty\,\Delta)(B : Ty\,(\Delta \rhd A)) \to \underline{Ty\,\Delta}$$

Note that this notion of inductive-inductive types is more general than the one considered in previous works [35], as we allow any number of type constructors, and arbitrary mixing of type and point constructors.

(4) U is closed under the equality type, with eliminator J and a weak (propositional) β-rule. Weakness is required because the translations in Sections 7 and 8 do not preserve this β-rule strictly. We explain this in more detail in Sections 4 and 9. Adding equality to the theory of signatures allows higher constructors and inductive equality parameters as well. We can now define the higher inductive circle as the following context:

$$\cdot,\ \ S^1 : \mathsf{U},\ \ base : \underline{S^1},\ \ loop : \underline{base =_{S^1} base}$$

The J rule allows constructors to mention operations on paths as well. For instance, the definition of the torus depends on path composition, which can be defined using J: given $p : \underline{t =_a u}$ and $q : \underline{u =_a v}$, $p \centerdot q$ abbreviates $\mathsf{J}_{a\,u\,x.z.(t=x)}\,p\,_v\,q : \underline{t =_a v}$. The torus is given as follows.

$$\cdot,\ \ T^2 : \mathsf{U},\ \ b : \underline{T^2},\ \ p : \underline{b =_{T^2} b},\ \ q : \underline{b =_{T^2} b},\ \ t : \underline{p \centerdot q =_{(b =_{T^2} b)} q \centerdot p}$$

With the equality type at hand, we can define a full well-typed syntax of type theory as given e.g. in [6] as an inductive type. Also, see the examples in the Agda formalization described in Section 10.

So far we were only able to define closed HIITs, which do not refer to external types. We add rules which include external types into signatures. A context Δ for which $\Gamma \vdash \Delta$ holds can be seen as a specification of an inductive type which depends on an external Γ signature. For example, in the case of lists for arbitrary external element types, Γ will be $A : \mathsf{Type}_0$.

(5) We have a function space where the domain is a type in the external theory. We distinguish it from (3) by using red brick color in the domain specification. We specify lists and the integers as follows.

$$A : \mathsf{Type}_0 \vdash \cdot,\ List : \mathsf{U},\ \ nil : \underline{List},\ \ cons : (x : A) \to List \to \underline{List}$$
$$\Gamma \vdash \cdot,\ Int : \mathsf{U},\ \ pair : (x\,y : Nat) \to \underline{Int},$$
$$eq : (a\,b\,c\,d : Nat)(p : a{+}d =_{Nat} b{+}c) \to \underline{pair\,a\,b =_{Int} \mathsf{pair}\,c\,d},$$
$$trunc : (xy : Int)(p\,q : a =_{Int} b) \to \underline{p =_{x =_{Int} y} q}$$

In the case of integers, Γ is $Nat : \mathsf{Type}_0, -{+}- : Nat \to Nat \to Nat$, or alternatively, we could require natural numbers in the external theory. As another example, propositional truncation for a type A is specified as follows.

$$A : \mathsf{Type}_0 \vdash \cdot,\ \ tr : \mathsf{U},\ \ emb : (x : A) \to \underline{tr},\ \ eq : (x\,y : tr) \to \underline{x =_{tr} y}$$

The smallness of A is required in (5). It is possible to generalize signatures to arbitrary universe levels, but it is not essential to the current development. Note that the (5) function space preserves strict positivity, since in the external theory there is no way to recursively refer to the inductive type *being defined*. The situation is analogous to the case of W-types [1], where shapes and positions can contain arbitrary types but they cannot recursively refer to the W-type being defined.

(6) U is also closed under a function space where the domain is an external type and the codomain is a small source theory type. We overload the application notation for external parameters, as it is usually clear from context which application is meant. The rules allow types with infinitary constructors, for example trees containing elements of A at the leaves and branching by B (which could be an infinite type):

$$A : \mathsf{Type}_0, B : \mathsf{Type}_0 \vdash \ \cdot, \ \ T : \mathsf{U}, \ \ leaf : (x : A) \to \underline{T}, \ \ node : ((x : B) \to T) \to \underline{T}$$

Here, $leaf$ has a function type (5) and $node$ has a function type (3) with a function type (6) in the domain. More generally, we can define W-types [1] as follows. S describes the "shapes" of the constructors and P the "positions" where recursive arguments can appear.

$$S : \mathsf{Type}_0, P : S \to \mathsf{Type}_0 \vdash \ \cdot, \ \ W : \mathsf{U}, \ \ sup : (s : S) \to ((p : P\, s) \to W) \to \underline{W}$$

For a more complex infinitary example, see the definition of Cauchy reals in [38, Definition 11.3.2]. It can be also found as an example file in our Haskell implementation.

The invalid examples Ival and Neg from Section 1 cannot be encoded by the theory of signatures. For Ival, we can go as far as

$$\cdot, \ \ Ival : \mathsf{U}, \ \ a : \underline{Ival}, \ \ b : \underline{Ival}, \ \ \sigma : \underline{? =_{Ival} ?}.$$

The first argument of the function $f : (X : \mathsf{Type}) \to X \to X$ is an external type, but we only have $Ival : \mathsf{U}$ in the theory of signatures. Neg cannot be typed because the first parameter of the constructor con is a function from a small type to an external type, and no such functions can be formed.

2.2. External type theory.

The external syntax serves two purposes: it is a source of types external to a HIIT signature, and it also serves as the target for the syntactic translations described in Sections 5 to 8. It is not essential that we use the same theory for both purposes; we do so only to simplify the presentation by skipping an additional translation or embedding step. Also, we do not specify the external theory in formal detail, since it is a standard type theory. We only make some assumptions about supported type formers. We generally keep the notation close to Agda, and use brick red color to distinguish from constructions in the theory of signatures.

There is a cumulative Russell-style hierarchy of universes Type_i, with universes closed under Π, Σ, equality and unit types. Importantly, we do not assume uniqueness of identity proofs.

The unit type is denoted $\top$ with constructor tt.

Dependent function space is denoted $(x : A) \to B$. We write $A \to B$ if B does not depend on x, and $\to$ associates to the right, $(x : A)(y : B) \to C$ abbreviates $(x : A) \to (y : B) \to C$ and $(x\, y : A) \to B$ abbreviates $(x : A)(y : A) \to B$. We write $\lambda x.t$ for abstraction and $t\, u$ for left-associative application.

$(x : A) \times B$ stands for Σ types, $A \times B$ for the non-dependent version. We sometimes use a short re-associated notation for left-nested iterated Σ types, for example $(A : \mathsf{Type}_0) \times A \times A$ may stand for the left-nested $(x : (A : \mathsf{Type}_0) \times A) \times \mathsf{proj}_1\, A$. The constructor for Σ is denoted $(t,\,u)$ with eliminators proj_1 and proj_2. Both Π and Σ have definitional β and η rules.

The equality type for a type A and elements $t : A$, $u : A$ is denoted $t =_A u$, and we have the constructor refl_t and the eliminator J with definitional β-rule. The notation is $\mathsf{J}_{A\,t\,P}\,pr_{\,u}\,eq$ for $t : A$, $P : (x : A) \to t =_A x \to \mathsf{Type}_i$, $pr : P\,t\,\mathsf{refl}$ and $eq : t =_A u$. Sometimes we omit parameters in subscripts.

We will use the following functions defined using J in the standard way. We write $\mathsf{tr}_P\,e\,t : P\,v$ for transport of $t : P\,u$ along $e : u = v$. We write $\mathsf{ap}\,f\,e : f\,u = f\,v$ where $f : A \to B$ and $e : u = v$, also $\mathsf{apd}\,f\,e : \mathsf{tr}_P\,e\,(f\,u) = f\,v$ where $f : (x : A) \to B$ and $e : u = v$. Borrowing notation from the homotopy type theory book [38], we write $p \cdot q$ for transitivity and p^{-1} for symmetry. We also make use of the groupoid law $\mathsf{inv}\,p : p^{-1} \cdot p = \mathsf{refl}$.

3. General Definitions for Induction and Recursion

Armed with a definition for HIIT signatures, we would like to have of notions of *induction* and *recursion* for each signature. However, instead of trying to directly extract them from signatures, it is more helpful to have more fundamental (categorical) semantic concepts: *algebras*, *homomorphisms*, *displayed algebras* and *sections of displayed algebras*. Then, we can express induction and recursion using these.

Let us first consider natural numbers, and see how the usual definition of induction arises.

For Nat, algebras are simply a triple consisting of a type, a value and an endofunction. Below, we leave universe indices explicit, and we use the notation of the external type theory described in Section 2.2.

$$\begin{aligned}&\mathsf{Alg} : \mathsf{Type}\\&\mathsf{Alg} \equiv (N : \mathsf{Type}) \times N \times (N \to N)\end{aligned}$$

Displayed Nat-algebras (sometimes called fibered algebras, as in [40]) are likewise triples, but each component depends on the corresponding components of a Nat-algebra. We borrow the term "displayed" from Ahrens and Lumsdaine [4], as our displayed algebras generalize their displayed categories.

$$\begin{aligned}&\mathsf{DisplayedAlg} : \mathsf{Alg} \to \mathsf{Type}\\&\mathsf{DisplayedAlg}\,(N,\,z,\,s) \equiv\\&\qquad (N^D : N \to \mathsf{Type}) \times (z^D : N^D\,z) \times ((n : N) \to N^D\,n \to N^D\,(s\,n))\end{aligned}$$

Homomorphisms, as usual in mathematics, are structure-preserving functions:

$$\begin{aligned}&\mathsf{Morphism} : \mathsf{Alg} \to \mathsf{Alg} \to \mathsf{Type}\\&\mathsf{Morphism}\,(N_0,\,z_0,\,s_0)\,(N_1,\,z_1,\,s_1) \equiv\\&\qquad (N^M : N_0 \to N_1) \times (z^M : N^M\,z_0 = z_1) \times ((n : N_0) \to N^M\,(s_0\,n) = s_1\,(N^M\,n))\end{aligned}$$

Sections of displayed algebras can be viewed as a dependently typed analogue of homomorphisms:

$$\begin{aligned}&\mathsf{Section} : (\alpha : \mathsf{Alg}) \to \mathsf{DisplayedAlg}\ \alpha \to \mathsf{Type}\\ &\mathsf{Section}\,(N,\, z,\, s)\,(N^D,\, z^D,\, s^D) \equiv\\ &\qquad (N^S : (n : N) \to N^D\, n) \times (z^S : N^S\, z = z^D) \times ((n : N) \to N^S\,(s\, n) = s^D\, n\,(N^S\, n))\end{aligned}$$

Now, we can reformulate induction for Nat. First, we assume that there exists a distinguished Nat-algebra, named $\mathsf{InitialAlg}$. The induction principle has the following type:

$$\mathsf{Induction} : (M : \mathsf{DisplayedAlg}\ \mathsf{InitialAlg}) \to \mathsf{Section}\, M$$

Unfolding the definitions, it is apparent that this is the same notion of Nat-induction as we gave before. The initial algebra consists of type and value constructors, the induction motives and methods are bundled into a displayed algebra, and as result we get a section, containing an eliminator function together with its β-rules. Additionally, we can define recursion using homomorphisms:

$$\mathsf{Recursion} : (\alpha : \mathsf{Alg}) \to \mathsf{Morphism}\,\mathsf{InitialAlg}\,\alpha$$

This corresponds to *weak initiality* in the sense of category theory: for each algebra, there is a morphism from the weakly initial algebra to it. Strong initiality in the setting of higher inductive types is called *homotopy initiality* [40], and it is defined as follows for Nat:

$$\mathsf{Initiality} : (\alpha : \mathsf{Alg}) \to \mathsf{isContr}\,(\mathsf{Morphism}\,\mathsf{InitialAlg}\,\alpha)$$

where $\mathsf{isContr}\, A \equiv (a : A) \times ((a' : A) \to a = a')$. Hence, there is a unique morphism from the initial algebra, but in the setting of homotopy type theory, unique inhabitation can be viewed instead as contractibility.

Observe that the definitions for $\mathsf{Induction}$, $\mathsf{Recursion}$ and $\mathsf{Initiality}$ need not refer to natural numbers, and can be used similarly in cases of other structures. Thus, the task in the following is to derive algebras, homomorphisms, displayed algebras and sections from HIIT signatures, in a way which generalizes beyond the current Nat example to indexed types, induction-induction and higher constructors. But even in the general case, displayed algebras yield induction motives and methods, and homomorphisms and sections yield a function for each type constructor and a β-rule for each point or path constructor.

We will compute algebras and the other notions by induction on the syntax of the theory of signatures. However, first we need to clarify the formal foundations of these computations.

4. The Coherence Problem and Syntactic Translations

The next task would be to define a computation which takes as input a $\Gamma \vdash \Delta$ signature, and returns the corresponding type of algebras in some type theory. This would behave as a "standard" model of signatures, which simply maps each construction in the syntax to its counterpart: types to types, universe to universe, functions to functions, and so on.

However, it is important to interpret signatures into a theory without uniqueness of identity proofs (UIP), because we are considering *higher* inductive types, and hence must remain compatible with higher-dimensional interpretations. In particular, we need to interpret type constructors in signatures into type universes which are not truncated to any homotopy level. In this setting, even the mundane $\alpha : (N : \mathsf{Type}) \times N \times (N \to N)$ natural number algebras may have arbitrary higher-dimensional structure.

On first look, we might think that the simplest way to formalize the algebra interpretation is the following:

(1) Assume as metatheory a type theory without UIP.
(2) In this setting, define a formal syntax of the theory of signatures.
(3) Give a standard interpretation of signatures into the UIP-free metatheory.

It may come as a surprise that realizing the above steps is an *open problem*, for any syntax of a dependent type theory. We call the problem of interpreting syntaxes of type theories into a UIP-free metatheory a *coherence problem*. This issue appears to arise with all known ways of defining syntaxes for dependent type theories.

In the following, we first consider the coherence problem in two settings: with intrinsically typed higher inductive-inductive syntaxes, then with conventional syntaxes involving preterms and inductively defined typing and conversion relations. Then, we present syntactic translations as a partial solution to the coherence problem.

4.1. **Interpreting intrinsic syntax.** Following Altenkirch and Kaposi [6], one might define the syntax of a type theory as an initial category with families (CwF) [19] extended with additional type formers. The CwF part provides a calculus and equational theory for explicit substitutions, upon which one can build additional type structure. We present an excerpt below:

Con	$: \mathsf{Set}$	contexts
Ty	$: \mathsf{Con} \to \mathsf{Set}$	types
Sub	$: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$	substitutions
Tm	$: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$	terms
$\cdot$	$: \mathsf{Con}$	empty context
$- \rhd -$	$: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$	context extension
$-[-]$	$: \mathsf{Ty}\,\Delta \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma$	type substitution
id	$: \mathsf{Sub}\,\Gamma\,\Gamma$	identity substitution
$[\mathsf{id}]$	$: A[\mathsf{id}] = A$	action of id on types
$- \circ -$	$: \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Gamma\,\Theta \to \mathsf{Sub}\,\Gamma\,\Delta$	substitution composition
$-[-]$	$: \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma])$	term substitution
...		
U	$: \mathsf{Ty}\,\Gamma$	universe
$\mathsf{U}[]$	$: \mathsf{U}[\sigma] = \mathsf{U}$	substituting the universe
El	$: \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma$	decoding
Π	$: (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \rhd A) \to \mathsf{Ty}\,\Gamma$	functions
...		

This notion of syntax is much more compact and often more convenient to use than extrinsic syntaxes. It is in essence "merely" a formalization of CwFs, which are often used in categorical semantics of type theory. However, its rigorous metatheory is subject to ongoing research (including the current paper). In a set-truncated setting, the current authors and

Altenkirch have previously developed semantics and constructed initial algebras [30], but here we need to work in a non-truncated theory.

4.1.1. *Set-truncation.* Additionally, we need to set-truncate the syntax by adding the following constructors:

$$\begin{aligned}
\mathsf{setTy} &: (A\,B : \mathsf{Ty}\,\Gamma)(p\,q : A = B) \to p = q\\
\mathsf{setTm} &: (t\,u : \mathsf{Tm}\,\Gamma\,A)(p\,q : t = u) \to p = q\\
\mathsf{setSub} &: (\sigma\,\delta : \mathsf{Sub}\,\Gamma\,\Delta)(p\,q : \sigma = \delta) \to p = q
\end{aligned}$$

We can omit the rule for contexts, as it is derivable from the above ones. Set truncation forces definitional equality of the syntax (as defined by equality constructors in the HIIT signature) to be proof irrelevant. If we omit set-truncation, then the defined HIIT becomes very different from what we expect the syntax to be.

We can easily show that the non-truncated syntax does not form sets. For example, $\mathsf{U}[]$ and $[\mathsf{id}]$ are two proofs of $\mathsf{U}[\mathsf{id}] = \mathsf{U}$, and they are not forced to be equal. Assuming univalence, we can give a model where types are interpreted as closed metatheoretic types, U is interpreted as metatheoretic Bool, $[\mathsf{id}]$ is interpreted as $\mathsf{refl} : A = A$ for some metatheoretic A type, and $\mathsf{U}[]$ is interpreted as the Bool negation equivalence, thereby formally distinguishing $\mathsf{U}[]$ and $[\mathsf{id}]$. Hence, by Hedberg's theorem [25], the syntax does not have decidable equality. This also implies that the non-truncated intrinsic syntax is not constructible from set quotients of extrinsic terms (since those always form sets). The non-truncated syntax just does not seem to be a sensible notion. This situation is similar to how categories in homotopy type theory need to have set-truncated morphisms [3].

Unfortunately, set-truncation makes it impossible to directly interpret syntactic types as elements of a UIP-free metatheoretic Type_i universe. This is because we must provide interpretations for all set-truncation constructors, which amounts to shoving that the interpretations of Ty, Tm, and Sub are all sets. However, we cannot show Type_i to be a set without UIP.

A possible solution would be to add *all higher coherences* instead of set-truncating, which would yield something like an $(\omega, 1)$-CwF, but this is also an open research problem [7, 24].

4.2. **Interpreting extrinsic syntax.** An extrinsic syntax for type theory is defined the following way:

(1) We inductively define a *presyntax*: sets of preterms, pretypes, precontexts, and possibly presubstitutions. These are not assumed to be well-formed, and only serve as raw material for expressions.
(2) We give mutual inductive definitions for the following relations on presyntax: well-formedness, typing and conversion.

This is the conventional way of presenting the syntax; see e.g. [43] for a detailed machine-checked formalization in this style. The main advantage compared to the intrinsic syntax is that this only requires conservative inductive definitions in the metatheory, which are also natively supported in current proof assistants, unlike HIITs. The main disadvantage is verbosity, lower level of abstraction, and a difficulty of pinning down a notion of model for the syntax.

What about interpreting extrinsic syntax into a UIP-free universe? It is widely accepted that extrinsic syntaxes have standard interpretations in set-truncated metatheories, although carrying this out in formal detail is technically challenging. Streicher's seminal work [41] laid out a template for doing this: first we construct a family of partial functions from the presyntax to the semantic domain, then we prove afterwards that these functions are total on well-formed input.

However, the coherence problem arises still: it is required that definitional equality is proof irrelevant. In a type-theoretic setting, this means that we need to propositionally truncate the conversion relation. This again prevents us from interpreting the syntax into a UIP-free universe. We have to interpret definitional equality in the syntax as propositional equality in the metatheory, but since the former is propositionally truncated, we can only eliminate it into propositions, and metatheoretic equality types are not generally propositions in the absence of UIP.

Could we define a conversion relation which is propositional, but not truncated? For example, conversion could be defined in terms of a deterministic conversion checking algorithm. But then a complication is that we do not have a proof that conversion checking is *total* and *stable under substitution*, while still in the process of defining the syntax.

Alternatively, we could first define a normalization algorithm for an extrinsic syntax in a UIP-free metatheory, and then try to interpret *only normal forms.* Abel, Öhman and Vezzosi demonstrated a UIP-free conversion checking algorithm in type theory [2], which suggests that UIP-free normalization may be possible as well. But since this normal form interpretation is a major technical challenge, and it has not been carried out yet, we cannot use it to justify constructions in the current paper.

4.3. **Syntactic translations.** We can circumvent the coherence problem in the following way:

(1) Define a *source* and a *target* syntax in any suitable metatheory, where the target theory does not have UIP. The source and target theories do not necessarily need to differ.
(2) Interpret the source syntax into the target syntax.

For extrinsic syntaxes, it is generally understood that a syntactic translation has to preserve definitional equalities in the source syntax. For intrinsic syntaxes, preservation of definitional equality is automatically enforced by the equality constructors. See Boulier et al. [13] for a showcase of syntactic translations.

Now, we can take the source syntax to be the theory of signatures from Section 2.1, and the target syntax to be the external syntax from Section 2.2. Truncation in the source syntax is not an issue here, because the target syntax is likewise truncated.

However, using syntactic translations is also a significant restriction. As always, we must map equal inputs to equal outputs, but now the notion of equality for outputs coincides with definitional equality in the syntax of the target theory, which is far more restrictive than propositional equality. Recall the weak β-rule for J in the theory of signatures in Section 2.1: if we instead used a strict equality, then the translations in Sections 7 and 8 would not work, because they map $\mathsf{J}_{a\,t\,(x.z.p)}\,pr\,{}_t\,\mathsf{refl}$ and pr to terms which are equal propositionally, but not definitionally. This restriction also prevents us from defining more translations which cover other parts of the categorical semantics, e.g. composition of homomorphisms. We return to this topic in Section 9.

In the following three sections we present syntactic translations yielding algebras, homomorphisms, displayed algebras and their sections. The presentation here, like in Section 2.1, is informal and focuses on readability. In particular, we omit interpretations for substitutions and preservation proofs for definitional equalities.

5. Algebras

We use $-^{\mathsf{A}}$ to the denote the translation which computes algebras. It is specified as follows, for contexts, types and terms in the theory of signatures.

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta^{\mathsf{A}} : \mathsf{Type}_1} \qquad \frac{\Gamma;\Delta \vdash A}{\Gamma \vdash A^{\mathsf{A}} : \Delta^{\mathsf{A}} \to \mathsf{Type}_1} \qquad \frac{\Gamma;\Delta \vdash t : A}{\Gamma \vdash t^{\mathsf{A}} : (\gamma : \Delta^{\mathsf{A}}) \to A^{\mathsf{A}}\,\gamma}$$

The $-^{\mathsf{A}}$ translation is essentially the standard interpretation of signatures, where every construction in the source syntax is interpreted with a corresponding brick red construction in the target syntax. The only notable change is in the interpretation of contexts: a source context is interpreted as an iterated Σ-type.

$$\begin{aligned}
\cdot^{\mathsf{A}} &:\equiv \top \\
(\Delta, x : A)^{\mathsf{A}} &:\equiv (\gamma : \Delta^{\mathsf{A}}) \times A^{\mathsf{A}}\,\gamma \\
x^{\mathsf{A}}\,\gamma &:\equiv x^{\text{th}} \text{ component in } \gamma \\
\mathsf{U}^{\mathsf{A}}\,\gamma &:\equiv \mathsf{Type}_0 \\
(\underline{a})^{\mathsf{A}}\,\gamma &:\equiv a^{\mathsf{A}}\,\gamma \\
((x : a) \to B)^{\mathsf{A}}\,\gamma &:\equiv (x : a^{\mathsf{A}}\,\gamma) \to B^{\mathsf{A}}\,(\gamma, x) \\
(t\,u)^{\mathsf{A}}\,\gamma &:\equiv (t^{\mathsf{A}}\,\gamma)\,(u^{\mathsf{A}}\,\gamma) \\
((x : A) \to B)^{\mathsf{A}}\,\gamma &:\equiv (x : A) \to B^{\mathsf{A}}\,\gamma \\
(t\,u)^{\mathsf{A}}\,\gamma &:\equiv (t^{\mathsf{A}}\,\gamma)\,u \\
(t =_a u)^{\mathsf{A}}\,\gamma &:\equiv t^{\mathsf{A}}\,\gamma = u^{\mathsf{A}}\,\gamma \\
\mathsf{refl}^{\mathsf{A}}\,\gamma &:\equiv \mathsf{refl} \\
(\mathsf{J}_{a\,t\,(x.z.p)}\,pr\;_{u}\,eq)^{\mathsf{A}}\,\gamma &:\equiv \mathsf{J}_{(a^{\mathsf{A}}\,\gamma)\,(t^{\mathsf{A}}\,\gamma)\,(\lambda x\,z.p^{\mathsf{A}}\,(\gamma, x, z))}\,(pr^{\mathsf{A}}\,\gamma)\;_{(u^{\mathsf{A}}\,\gamma)}\,(eq^{\mathsf{A}}\,\gamma) \\
(\mathsf{J}\beta_{a\,t\,(x.z.p)}\,pr)^{\mathsf{A}}\,\gamma &:\equiv \mathsf{refl} \\
((x : A) \to b)^{\mathsf{A}}\,\gamma &:\equiv (x : A) \to b^{\mathsf{A}}\,\gamma \\
(t\,u)^{\mathsf{A}}\,\gamma &:\equiv (t^{\mathsf{A}}\,\gamma)\,u
\end{aligned}$$

For example, $-^{\mathsf{A}}$ acts as follows on signature for circles:

$$(\cdot, S^1 : \mathsf{U}, b : \underline{S^1}, loop : \underline{b = b})^{\mathsf{A}} \equiv \top \times (S^1 : \mathsf{Type}_0) \times (b : S^1) \times (loop : b = b)$$

Note that the resulting Σ type is left-nested, and we use the reassociated notation for readability. The result could be written without syntactic sugar the following way:

$$\Big(x'' : (x' : (x : \top) \times \mathsf{Type}_0) \times \mathsf{proj}_2\,x'\Big) \times (\mathsf{proj}_2\,x'' = \mathsf{proj}_2\,x'')$$

We shall keep to the short notation from now on.

6. Displayed Algebras

The $-^{\mathsf{D}}$ translation computes displayed algebras for a signature, which can be viewed as bundles of induction motives and methods. $-^{\mathsf{D}}$ is an unary logical predicate translation over the $-^{\mathsf{A}}$ translation. It is related to the logical predicate translation of Bernardy et al. [12], but our implementation differs by interpreting contexts as Σ-types instead of extended contexts.

We fix a universe level i for the translation. For each context Δ, Δ^{D} is a predicate over the standard interpretation Δ^{A}. For a type $\Delta \vdash A$, A^{D} is a predicate over A^{A}, which also depends on $\gamma : \Delta^{\mathsf{A}}$ and a witness of $\Delta^{\mathsf{D}}\,\gamma$. All of these may refer to a target theory context Γ.

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta^{\mathsf{D}} : \Delta^{\mathsf{A}} \to \mathsf{Type}_{i+1}} \qquad \frac{\Gamma; \Delta \vdash A}{\Gamma \vdash A^{\mathsf{D}} : (\gamma : \Delta^{\mathsf{A}}) \to \Delta^{\mathsf{D}}\,\gamma \to A^{\mathsf{A}}\,\gamma \to \mathsf{Type}_{i+1}}$$

For a term t, t^{D} witnesses that the predicate corresponding to its type holds for t^{A}; this can be viewed as a *fundamental theorem* for the predicate interpretation.

$$\frac{\Gamma; \Delta \vdash t : A}{\Gamma \vdash t^{\mathsf{D}} : (\gamma : \Delta^{\mathsf{A}})(\gamma^D : \Delta^{\mathsf{D}}\,\gamma) \to A^{\mathsf{D}}\,\gamma\,\gamma^D\,(t^{\mathsf{A}}\,\gamma)}$$

The implementation of $-^{\mathsf{D}}$ is given below. We leave γ-s mostly implicit, marking only γ^D witnesses.

$$\begin{array}{ll}
\bullet^{\mathsf{D}}\;\gamma & :\equiv \top \\
(\Delta,\, x : A)^{\mathsf{D}}\,(\gamma,\, t) & :\equiv (\gamma^D : \Delta^D\,\gamma) \times A^D\,\gamma^D\,t \\
x^{\mathsf{D}}\,\gamma^D & :\equiv x^{\text{th}} \text{ component in } \gamma^D \\
\mathsf{U}^{\mathsf{D}}\,\gamma^D\,A & :\equiv A \to \mathsf{Type}_i \\
(\underline{a})^{\mathsf{D}}\,\gamma^D\,t & :\equiv a^{\mathsf{D}}\,\gamma^D\,t \\
((x : a) \to B)^{\mathsf{D}}\,\gamma^D\,f & :\equiv (x : a^{\mathsf{A}}\,\gamma)(x^D : a^{\mathsf{D}}\,\gamma^D\,x) \to B^{\mathsf{D}}\,(\gamma, x)\,(\gamma^D, x^D)\,(f\,x) \\
(t\,u)^{\mathsf{D}}\,\gamma^D & :\equiv (t^{\mathsf{D}}\,\gamma^D)\,(u^{\mathsf{A}}\,\gamma)\,(u^{\mathsf{D}}\,\gamma^D) \\
((x : A) \to B)^{\mathsf{D}}\,\gamma^D\,f & :\equiv (x : A) \to B^{\mathsf{D}}\,\gamma^D\,(f\,x) \\
(t\,u)^{\mathsf{D}}\,\gamma^D & :\equiv t^{\mathsf{D}}\,\gamma^D\,u \\
(t =_a u)^{\mathsf{D}}\,\gamma^D\,e & :\equiv \mathsf{tr}_{(a^{\mathsf{D}}\,\gamma^D)}\,e\,(t^{\mathsf{D}}\,\gamma^D) = u^{\mathsf{D}}\gamma^D \\
(\mathsf{refl}_t)^{\mathsf{D}}\,\gamma^D & :\equiv \mathsf{refl}_{(t^{\mathsf{D}}\,\gamma^D)} \\
(\mathsf{J}_{a\,t\,(x.z.p)}\,pr\;_u\,eq)^{\mathsf{D}}\,\gamma^D & :\equiv \mathsf{J}\,(\mathsf{J}\,(pr^{\mathsf{D}}\,\gamma^D)\,(eq^{\mathsf{A}}\,\gamma))\,(eq^{\mathsf{D}}\,\gamma^D) \\
(\mathsf{J}\beta_{a\,t\,(x.z.p)}\,pr)^{\mathsf{D}}\,\gamma^D & :\equiv \mathsf{refl} \\
((x : A) \to b)^{\mathsf{D}}\,\gamma^D\,f & :\equiv (x : A) \to b^D\,\gamma^D\,(f\,x) \\
(t\,u)^{\mathsf{D}}\,\gamma^D & :\equiv t^{\mathsf{D}}\,\gamma^D\,u
\end{array}$$

The predicate for a context is given by iterating $-^{\mathsf{D}}$ for its constituent types. For a variable, the corresponding witness is looked up from γ^D.

The translation of the universe, given an element of $A : \mathsf{U}^{\mathsf{A}}\,\gamma$ (with $\mathsf{U}^{\mathsf{A}}\,\gamma \equiv \mathsf{Type}_0$) returns the predicate space over A. For $\underline{a}$ types, we just return the translation of a.

The predicate for a function type for inductive parameters expresses preservation of predicates. Witnesses of application are given by recursive application of $-^{\mathsf{D}}$. The definitions for the other (non-inductive) function spaces are similar, except there is no predicate for the domain types, and thus no witnesses are required.

The translation for the equality type $t =_a u$, for each $e : (t =_a u)^{\mathsf{A}}\,\gamma$, i.e. $e : t^{\mathsf{A}}\,\gamma = u^{\mathsf{A}}\,\gamma$, witnesses that t^{D} and u^{D} are equal. As these have different types, we have to transport over the original equality e. Hence, induction methods for path constructors will be *paths over paths* in the sense of homotopy type theory.

refl is interpreted with just reflexivity in the target syntax. The interpretation of J is given by a double J application, borrowing the definition from Lasson [32]. Here, we use a shortened J notation; see the formalization (Section 10) for details.

Again, let us consider the circle example:

$$\begin{aligned}&(\cdot,\ S^1 : \mathsf{U},\ b : \underline{S^1},\ loop : \underline{b = b})^{\mathsf{D}}\ (\mathsf{tt},\ S^1,\ b,\ loop)\\ \equiv\ & \top \times (S^{1D} : S^1 \to \mathsf{Type}_i) \times (b^D : S^{1D}\,b) \times (loop^D : \mathsf{tr}_{S^{1D}}\,loop\,b^D = b^D)\end{aligned}$$

The inputs of $-^{\mathsf{D}}$ here are the signature for the circle (the context in black) and an S^1-algebra consisting of three non-$\top$ components. It returns a family over the type S^1, an element of this family b^D at index b, and and an equality between b^D and b^D which lies over $loop$. This is the same as the usual induction motives and methods for the circle, e.g. as described in [38].

7. From Logical Relations to Homomorphisms

In this section, we specify the $-^{\mathsf{M}}$ translation, which computes homomorphisms of algebras. We do so by first considering a logical relation interpretation, and then refining it towards homomorphisms. For dependent type theories, logical relation models are well-known (see e.g. [8]), so they are certainly applicable to our restricted syntax as well.

Below, we list induction motives for $-^{\mathsf{M}}$; these remain the same as we move from logical relations to homomorphisms. The universe level i was chosen previously for the $-^{\mathsf{D}}$ translation.

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta^{\mathsf{M}} : \Delta^{\mathsf{A}} \to \Delta^{\mathsf{A}} \to \mathsf{Type}_i} \qquad \frac{\Gamma;\Delta \vdash A}{\Gamma \vdash A^{\mathsf{D}} : (\gamma_0\,\gamma_1 : \Delta^{\mathsf{A}}) \to \Delta^{\mathsf{M}}\,\gamma_0\,\gamma_1 \to A^{\mathsf{A}}\,\gamma_0 \to A^{\mathsf{A}}\,\gamma_1 \to \mathsf{Type}_i}$$

$$\frac{\Gamma;\Delta \vdash t : A}{\Gamma \vdash t^{\mathsf{M}} : (\gamma_0\,\gamma_1 : \Delta^{\mathsf{A}})(\gamma^M : \Delta^{\mathsf{M}}\,\gamma_0\,\gamma_1) \to A^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,(t^{\mathsf{A}}\,\gamma_0)\,(t^{\mathsf{A}}\,\gamma_1)}$$

Contexts are mapped to (proof-relevant) relations and types to families of relations depending on interpreted contexts. For terms, we again get a fundamental theorem: every term has related standard interpretations in related semantic contexts.

We present below the logical relation interpretation only for contexts, variables, the universe and the inductive function space.

$$\begin{aligned}\cdot^{\mathsf{M}}\,\gamma_0\,\gamma_1 &:\equiv \top\\ (\Delta, x : A)^{\mathsf{M}}\,\gamma_0\,\gamma_1 &:\equiv (\gamma^M : \Delta^{\mathsf{M}}\,\gamma_0\,\gamma_1) \times A^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\\ x^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M &:\equiv x^{\text{th}} \text{ component in } \gamma^M\end{aligned}$$

$$\begin{aligned}
&\mathsf{U}^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,A_0\,A_1 &&:\equiv A_0 \to A_1 \to \mathsf{Type}_0\\
&(\underline{a})^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,t_0\;t_1 &&:\equiv a^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,t_0\;t_1\\
&((x:a)\to B)^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,f_0\,f_1 &&:\equiv (x_0 : (\underline{a})^{\mathsf{A}}\,\gamma_0)(x_1 : (\underline{a})^{\mathsf{A}}\,\gamma_1)(x^M : (\underline{a})^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M)\\
& &&\quad\to B^{\mathsf{M}}\,(\gamma_0,\,x_0)\,(\gamma_1,\,x_1)\,(\gamma^M,\,x^M)\,(f_0\,x_0)\,(f_1\,x_1)
\end{aligned}$$

We interpret the universe as relation space, and function types as relations expressing pointwise relatedness of functions. This interpretation would work the same way for unrestricted (non-strictly positive) function types as well. For an example, this yields the following definition of logical relations between natural number algebras:

$$\begin{aligned}
&(\bullet, Nat : \mathsf{U}, zero : \underline{Nat}, suc : Nat \to \underline{Nat})^{\mathsf{M}}\,(\mathsf{tt},\,N_0,\,z_0,\,s_0)\,(\mathsf{tt},\,N_1,\,z_1,\,s_1)\\
\equiv\;& \top \times (N^M : N_0 \to N_1 \to \mathsf{Type}_0)\\
&\times (z^M : N^M\,z_0\,z_1)\\
&\times (s^M : (x_0 : N_0)(x_1 : N_1)(x^M : N^M\,x_0\,x_1) \to N^M\,(s_0\,x_0)\,(s_1\,x_1))
\end{aligned}$$

However, we would like to have underlying functions instead of relations in homomorphisms. We take hint from the fact that for classical (simply-typed and single-sorted) algebraic theories, a logical relation is equivalent to a homomorphism if and only if the underlying relation is the graph of a function [26, pg. 5]. Thus we make the following change:

$$\mathsf{U}^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,A_0\,A_1 :\equiv A_0 \to A_1$$

This requires us to change $(\underline{a})^{\mathsf{M}}$ as well, since we need to produce a type as result, but a^M now yields a function. We can view the result of a^{M} as a functional relation, and use its graph to relate t_0 and t_1:

$$(\underline{a})^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,t_0\;t_1 :\equiv a^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,t_0 = t_1$$

At this point we have merely restricted relations to functions, and left the rest of the interpretation unchanged. However, this is not strictly the desired notion of homomorphism. Consider now again the $-^{\mathsf{M}}$ interpretation for natural numbers:

$$\begin{aligned}
&(\bullet, Nat : \mathsf{U}, zero : \underline{Nat}, suc : Nat \to \underline{Nat})^{\mathsf{M}}\,(\mathsf{tt},\,N_0,\,z_0,\,s_0)\,(\mathsf{tt},\,N_1,\,z_1,\,s_1)\\
\equiv\;& \top \times (N^M : N_0 \to N_1)\\
&\times (z^M : N^M\,z_0 = z_1)\\
&\times (s^M : (x_0 : N_0)(x_1 : N_1)(x^M : N^M\,x_0 = x_1) \to N^M\,(s_0\,x_0) = s_1\,x_1)
\end{aligned}$$

In s^M, there is a superfluous x^M equality proof. Fortunately, in the translation of the inductive function space, we can just singleton contract x^M away, yielding an equivalent, but stricter definition:

$$\begin{aligned}
&((x:a)\to B)^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,f_0\,f_1 :\equiv\\
&\quad (x_0 : a^{\mathsf{A}}\,\gamma_0) \to B^{\mathsf{M}}\,(\gamma_0,\,x_0)\,(\gamma_1,\,a^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,x_0)\,(\gamma^M,\,\mathsf{refl})\,(f_0\,x_0)\,(f_1\,(a^{\mathsf{M}}\,\gamma_0\,\gamma_1\,\gamma^M\,x_0))
\end{aligned}$$

Now, the β-rule for successors is as expected:

$$s^M : (x_0 : N_0) \to N^M\,(s_0\,x_0) = s_1\,(N^M\,x_0)$$

Note that this singleton contraction is not possible for a general non-strictly positive function space. We rely on the domain being small: $(\underline{a})^{\mathsf{M}}$ yields an equation, but for general $\Gamma; \Delta \vdash A$ types, A^{M} only yields an unknown relation.

$-^{\mathsf{M}}$ is similar to the translation to displayed algebra sections, which is discussed in the next Section. We will discuss in more detail the interpretations of equalities and the other (external) function types there. A full listing for the homomorphism translation can be found in Appendix A.

8. Displayed Algebra Sections

The operation $-^{\mathsf{S}}$ yields displayed algebra sections. Sections can be viewed as dependent homomorphisms: while homomorphisms are structure-preserving families of functions, sections are structure-preserving families of dependent functions.

Contexts are interpreted as dependent relations between algebras and displayed algebras. We again fix a universe level i.

$$\frac{\Gamma \vdash \Delta}{\Gamma; \Delta^{\mathsf{S}} : (\gamma : \Delta^{\mathsf{A}}) \to \Delta^{\mathsf{D}}\,\gamma \to \mathsf{Type}_i}$$

Types are interpreted as dependent relations which additionally depend on γ, γ^D, γ^S interpretations of the context.

$$\frac{\Gamma; \Delta \vdash A}{\Gamma \vdash A^{\mathsf{S}} : (\gamma : \Delta^{\mathsf{A}})(\gamma^D : \Delta^{\mathsf{D}}\,\gamma)(\gamma^S : \Delta^{\mathsf{S}}\,\gamma\,\gamma^D)(x : A^{\mathsf{A}}\,\gamma) \to A^{\mathsf{D}}\,\gamma\,\gamma^D\,x \to \mathsf{Type}_i}$$

For a term t, t^{S} again witnesses a fundamental theorem.

$$\frac{\Gamma; \Delta \vdash t : A}{\Gamma \vdash t^{\mathsf{S}} : (\gamma : \Delta^{\mathsf{A}})(\gamma^D : \Delta^{\mathsf{D}}\,\gamma)(\gamma^S : \Delta^{\mathsf{S}}\,\gamma\,\gamma^D) \to A^{\mathsf{S}}\,\gamma\,\gamma^D\,\gamma^S\,(t^{\mathsf{A}}\,\gamma)\,(t^{\mathsf{D}}\,\gamma\,\gamma^D)}$$

We present the implementation below. Here, we make γ-s and γ^D-s implicit and mostly notate γ^S parameters and applications.

$$\begin{aligned}
&\bullet^{\mathsf{S}}\,\gamma\,\gamma^D && :\equiv \top \\
&(\Delta, x : A)^{\mathsf{S}}\,(\gamma, t)\,(\gamma^D, t^D) && :\equiv (\gamma^S : \Delta^{\mathsf{S}}\gamma^2) \times A^{\mathsf{S}}\,\gamma^S\,t\,t^D \\
&x^{\mathsf{S}}\,\gamma^S && :\equiv x^{\text{th}} \text{ component in } \gamma^S \\
&\mathsf{U}^{\mathsf{S}}\,\gamma^S\,A\,A^D && :\equiv (x : A) \to A^D\,x \\
&(\underline{a})^{\mathsf{S}}\,\gamma^S\,t\,t^D && :\equiv a^{\mathsf{S}}\,\gamma^S\,t = t^D \\
&((x : a) \to B)^{\mathsf{S}}\,\gamma^S\,f\,f^D && :\equiv (x : a^{\mathsf{A}}\,\gamma) \to B^{\mathsf{S}}\,(\gamma, x)\,(\gamma^D, a^{\mathsf{S}}\,\gamma^S\,x)\,(\gamma^S, \mathsf{refl}) \\
& && \qquad\qquad (f\,x)\,\big(f^D\,x\,(a^{\mathsf{S}}\,\gamma^S\,x)\big) \\
&(t\,u)^{\mathsf{S}}\,\gamma^S && :\equiv \mathsf{J}\,(t^{\mathsf{S}}\,\gamma^S\,(u^{\mathsf{A}}\,\gamma))\,(u^{\mathsf{S}}\,\gamma^S) \\
&((x : A) \to B)^{\mathsf{S}}\,\gamma^S\,f\,f^D && :\equiv (x : A) \to B^{\mathsf{S}}\,\gamma^S\,(f\,x)\,(f^D\,x) \\
&(t\,u)^{\mathsf{S}}\,\gamma^S && :\equiv t^{\mathsf{S}}\,\gamma^S\,u \\
&(t =_a u)^{\mathsf{S}}\,\gamma^S\,e && :\equiv \mathsf{tr}\,(t^{\mathsf{S}}\,\gamma^S)\big(\mathsf{tr}\,(u^{\mathsf{S}}\,\gamma^S)\,(\mathsf{apd}\,(a^{\mathsf{S}}\,\gamma^S)\,e)\big) \\
&(\mathsf{refl}_t)^{\mathsf{S}}\,\gamma^S && :\equiv \mathsf{J}\,\mathsf{refl}\,(t^{\mathsf{S}}\,\gamma^S)
\end{aligned}$$

$$
\begin{aligned}
&(\mathsf{J}_{a\,t\,(x.z.p)}\,pr\,{}_{u}\,eq)^{\mathsf{S}}\,\gamma^{S} &&:\equiv\\
&\quad \mathsf{J}\Big(\mathsf{J}\Big(\mathsf{J}\,(\mathsf{J}\,(\lambda\,p^{D}\,p^{S}\,pr^{D}\,pr^{S}.\,pr^{S})\,(t^{\mathsf{S}}\,\gamma^{S})\\
&\qquad (\mathsf{uncurry}\,p^{\mathsf{D}}\,\gamma^{2})\,(\mathsf{uncurry}\,p^{\mathsf{S}}\,\gamma^{S})\,(pr^{\mathsf{D}}\,\gamma^{2})\,(pr^{\mathsf{S}}\,\gamma^{S}))\,(eq^{\mathsf{A}}\,\gamma)\Big)\,(u^{\mathsf{S}}\,\gamma^{S})\Big)\,(eq^{\mathsf{S}}\,\gamma^{S})\\
&(\mathsf{J}\beta_{a\,t\,(x.z.p)}\,pr)^{\mathsf{S}}\,\gamma^{S} &&:\equiv\\
&\quad \mathsf{J}\,(\mathsf{J}\,(\lambda\,p^{D}\,p^{S}.\,\mathsf{refl})\,(t^{\mathsf{S}}\,\gamma^{S})\,(\mathsf{uncurry}\,p^{\mathsf{D}}\,\gamma^{2})\,(\mathsf{uncurry}\,p^{\mathsf{S}}\,\gamma^{S}))\,(pr^{\mathsf{S}}\,\gamma^{S})\\
&((x:A)\to b)^{\mathsf{S}}\,\gamma^{S}\,f\,t &&:\equiv b^{\mathsf{S}}\,\gamma^{S}\,(f\,t)\\
&(t\,u)^{\mathsf{S}}\,\gamma^{S} &&:\equiv \mathsf{ap}\,(\lambda f.f\,u)\,(t^{\mathsf{S}}\,\gamma^{S})
\end{aligned}
$$

The interpretations follow the same pattern as in the case of $-^{\mathsf{M}}$ up until $((x:a)\to B)^{\mathsf{S}}$, with U^{S} defined as a dependent function instead of a non-dependent one. Also, $\mathsf{U}^{\mathsf{S}}\,\gamma^{S}\,A\,A^{D}$ is precisely the type of sections of the A^{D} type family.

Let us consider now the translation in with the corresponding induction principles in mind, defined as $\mathsf{Induction}$ in Section 3.

The U^{S} rule yields the type of the eliminator function for a type constructor. For natural numbers, the non-indexed $Nat:\mathsf{U}$ is interpreted as $Nat^{S}:(x:Nat)\to Nat^{D}\,x$. For indexed types, the indices are first processed by the $-^{\mathsf{S}}$ cases for inductive and external function parameters, until the ultimate U return type is reached. Hence, we always get an eliminator function for a type constructor.

Analogously, the $-^{\mathsf{S}}$ result type for a point or path constructor is always a β-rule, i.e. a function type returning an equality. That is because $(\underline{a})^{\mathsf{S}}$ expresses that applications of a^{S} eliminators must be equal to the corresponding t^{D} induction methods. Hence, for path and point constructor types, $-^{\mathsf{S}}$ works by first processing all inductive and external parameters, then finally returning an equality type.

Here, we only provide abbreviated definitions for the $t\,u$, $t =_{a} u$, refl, J and $\mathsf{J}\beta$ cases. In the J case, we write $\mathsf{uncurry}\,p^{\mathsf{D}}$ for $\lambda\,\gamma\,\gamma^{D}\,x\,x^{D}\,z\,z^{D}.\,p^{\mathsf{D}}\,(\gamma,x,z)\,(\gamma^{D},x^{D},z^{D})$ and analogously elsewhere, to adjust for the fact that p abstracts over additional x and z variables. The full definitions can be found in the Agda formalization. The definitions are highly constrained by the required types, and not particularly difficult to implement with the help of a proof assistant: they all involve doing successive path induction on all equalities available from induction hypotheses, with appropriately generalized induction motives.

The full $(\mathsf{J}_{a\,t\,(x.z.p)}\,pr\,{}_{u}\,eq)^{\mathsf{S}}$ definition is quite large, and, for instance, yields a very large β-rule for the higher inductive torus definition (the reader can confirm this using the Haskell implementation). One could have an implementation with specialized cases for commonly used operations such as path compositions and inverses, in order to produce smaller translation output.

The circle example is a bit more interesting here:

$$
\begin{aligned}
&(\bullet,\,S^{1}:\mathsf{U},\,b:\underline{S^{1}},\,loop:\underline{b=b})^{\mathsf{S}}\,(\mathsf{tt},\,S^{1},\,b,\,loop)\,(\mathsf{tt},\,S^{1D},\,b^{D},\,loop^{D})\\
\equiv\;& \top\times(S^{1S}:(x:S^{1})\to S^{1D}\,x)\times(b^{S}:S^{1S}\;b=b^{D})\\
&\quad\times(loop^{S}:\mathsf{tr}_{\,(\lambda x.\mathsf{tr}_{\,S^{1D}}\,loop\,x\,=\,b^{D})}\,b^{S}\,(\mathsf{tr}_{\,(\lambda x.\mathsf{tr}_{\,S^{1D}}\,loop\,(S^{1S}\,b)\,=\,x)}\,b^{S}\,(\mathsf{apd}\,S^{1S}\,loop))\\
&\qquad\qquad = \;loop^{D})
\end{aligned}
$$

In homotopy type theory, the β-rule for *loop* is usually just $\mathsf{apd}\, S^{1E}\, loop = loop^D$, but here all β-rules are propositional, so we need to transport with b^S to make the equation well-typed. When computing the type of $loop^S$, we start with $(\underline{b = b})^S\ \gamma^3\ loop\ loop^D$. Next, this evaluates to $(b = b)^{\mathsf{S}}\ \gamma^3\ loop = loop^D$, and then we unfold the left hand side to get the doubly-transported expression in the result.

In Appendix B, we show how the type of displayed algebra sections is computed for the two-dimensional sphere. In Appendix C, we show the same for indexed W-types.

9. Possible Extensions to Categorical Semantics

So far, we were able to compute algebras, displayed algebras, morphisms and sections, and this allows us to state recursion and induction principles. Reiterating Section 3, for a signature $\Gamma \vdash \Delta$ and assuming $\Gamma \vdash \mathsf{InitialAlg} : \Delta^{\mathsf{A}}$, we have the following types for induction and recursion:

$$\Gamma \vdash \mathsf{Induction} : (\gamma^D : \Delta^{\mathsf{D}}\ \mathsf{InitialAlg}) \to \Delta^{\mathsf{S}}\ \mathsf{InitialAlg}\ \gamma^D$$

$$\Gamma \vdash \mathsf{Recursion} : (\gamma : \Delta^{\mathsf{A}}) \to \Delta^{\mathsf{M}}\ \mathsf{InitialAlg}\ \gamma$$

However, this is not the full picture. We would also like to have a *category* of algebras, with homomorphisms as morphisms. This is not without difficulties.

We are working in an UIP-free setting. In such setting, standard definitions of categories feature set-truncated morphisms [38]. But we have non-truncated notions of algebras and homomorphisms, so we cannot use the standard definitions. One solution is to simply use non-truncated categories; these have been previously called "precategories" or "wild categories" [14]. Sojakova demonstrated [40] that working with wild categories internally to type theory is enough to prove equivalence of induction and homotopy initiality for a class of higher inductive types, which suggests that the same might be possible for HIITs. Still, it would be desirable to build semantics of HIITs in a richer $(\omega, 1)$-categorical setting.

However, out current approach does not scale to the point where we have to worry about higher categories. We explain in the following. The natural next step towards a categorical semantics would be defining a $-^{\mathsf{ID}}$ translation, which computes identity homomorphisms:

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta^{\mathsf{ID}} :\ (\gamma : \Delta^{\mathsf{A}}) \to \Delta^{\mathsf{M}}\ \gamma\ \gamma} \qquad \frac{\Gamma ; \Delta \vdash A}{\Gamma \vdash A^{\mathsf{ID}} :\ (\gamma : \Delta^{\mathsf{A}})(t : A^{\mathsf{A}}\ \gamma) \to A^{\mathsf{M}}\ \gamma\ \gamma\ (\Delta^{\mathsf{ID}}\ \gamma)\ t\ t}$$

$$\frac{\Gamma ; \Delta \vdash t : A}{\Gamma \vdash t^{\mathsf{ID}} :\ (\gamma : \Delta^{\mathsf{A}}) \to t^{\mathsf{M}}\ \gamma\ \gamma\ (\Delta^{\mathsf{ID}}\ \gamma) = A^{\mathsf{ID}}\ \gamma\ (t^{\mathsf{A}}\ \gamma)}$$

Here, the interpretation of terms witnesses functoriality: t^{M} maps identity morphisms in Δ to displayed identity morphisms in A. We translate U to identity functions:

$$\mathsf{U}^{\mathsf{ID}}\ \gamma\ A :\equiv \lambda\, (x : A).\, x$$

Assume that σ is a parallel substitution, and note that U and $\mathsf{U}[\sigma]$ are definitionally equal in the theory of signatures. The translation of the latter would be the following (omitting many details, including the handling of substitutions in the translation):

$$(\mathsf{U}[\sigma])^{\mathsf{ID}}\ \gamma\ A \equiv \mathsf{tr}_{(\lambda\, x. A \to A)}\ (\sigma^{\mathsf{ID}}\ \gamma)\ (\lambda\, x.\, x)$$

Here, $\sigma^{\mathsf{ID}}\,\gamma$ yields an equation for functoriality of σ. Since the transport is constant, the result is propositionally equal to $\lambda\, x.\, x$. However, that is not enough, since we need to preserve $\mathsf{U} = \mathsf{U}[\sigma]$ up to definitional equality.

There are numerous unavoidable instances of such failures of strict preservation of definitional equalities for $-^{\mathsf{ID}}$, and they similarly arise when we try to construct composition of homomorphisms. We consider three potential solutions for this strictness problem:

(1) Solving the coherence problem. This would allow us to interpret signatures into the metatheory, allowing preservation of definitional equality up to propositional metatheoretic equality.
(2) Reformulating the syntax of signatures so that definitional equalities become weak propositional equalities. We already use weak β for J, can we do so elsewhere? However, this would result in an unusual and very inconvenient syntax of signatures, because we would need to weaken even basic substitution rules. In contrast, weak β for J seems harmless, because we are not aware of any HIIT signature in the literature that involves any J computation.
(3) Instead of interpreting signatures into an UIP-free type theory, we interpret them into classical combinatorial structures for higher groupoids and categories, e.g. into simplicial sets. The drawback is that now we do not have the convenient synthetic notion of higher structures, provided by the syntax of type theory, and instead we have to manually build up these structures. Needless to say, this makes machine-checked formalization much more difficult. We have found mechanized formalization invaluable for the current paper, and it would be painful to abandon it in further research.

In summary, solutions for the strictness problems require significant deviation from the approach of the current paper, or require significant further research.

10. Formalization and Implementation

There are three additional development artifacts to the current work: a Haskell implementation, a shallow Agda formalization and a deep Agda formalization. All three are available from `https://github.com/akaposi/hiit-signatures`.

The Haskell implementation takes as input a file which contains a of a $\Gamma \vdash \Delta$ signature. Then, it checks the input with respect to the rules in Figure 1, and outputs an Agda-checkable file which contains algebras, homomorphisms, displayed algebras and sections for the input signature.

It comes with examples, including the ones in this paper, the inductive-inductive dense completion [35, Appendix A.1.3] and several HITs from [38] including the definition of Cauchy reals. It can be checked that our implementation computes the expected elimination principles in these cases.

The shallow Agda formalization embeds both the source and target theories shallowly into Agda: it represents types as Agda types, functions as Agda functions, and so on. We also leave the $-^{\mathsf{A}}$ operation implicit. We state each case of translations as Agda functions from all induction hypotheses to the result type of the translation, which lets us "typecheck" the translation. We have found that this style of formalization is conveniently light, but remains detailed enough to be useful. We also generated some of the code of the Haskell implementation from this formalization.

The deep Agda formalization deeply embeds the theory of signatures as a CwF with additional structure, in the style of [6]. However, it embeds the external type theory shallowly as Agda, and we model dependency on external Γ contexts with Agda functions. We formalize $-^{\mathsf{A}}$, $-^{\mathsf{D}}$, $-^{\mathsf{M}}$ and $-^{\mathsf{S}}$ translations as a single model construction in Agda. This setup greatly simplifies formalization, since we do not have to reason explicitly about definitional equality in the external syntax, but we can still reason directly about preservation of definitional equalities in the theory of signatures. This semi-deep formalization can be in principle converted into a fully formal syntactic translation, because we prove all preservations of definitional equalities by refl. Due to technical challenges, this formalization uses transport instead of J in the source theory, but this still covers a rather large class of HIIT definitions.

11. Conclusions

Higher inductive-inductive types are useful in defining the well-typed syntax of type theory in an abstract way [6]. From a universal algebraic point of view, they provide initial algebras for multi-sorted algebraic theories where the sorts can depend on each other. From the perspective of homotopy type theory, they provide synthetic versions of homotopy-theoretic constructions such as higher-dimensional spheres or cell complexes. So far, no general scheme of HIITs have been proposed. To quote Lumsdaine and Shulman [33]:

> "The constructors of an ordinary inductive type are unrelated to each other. But in a higher inductive type each constructor must be able to refer to the previous ones; specifically, the source and target of a path-constructor generally involve the previous point-constructors. No syntactic scheme has yet been proposed for this that covers all cases of interest while remaining meaningful and consistent."

In this paper we proposed such a syntactic scheme which also includes inductive-inductive types. We tackled the problem of complex dependencies on previous type formation rules and constructors by a well-known method of describing intricate dependencies: the syntax of type theory itself. We had to limit the type formers to only allow strictly positive definitions, but these restrictions are the only things that a type theorist has to learn to understand our signatures. Our encoding is also direct in the sense that notions of induction and recursion are computed exactly as required and not merely up to equivalences or isomorphisms, and we also demonstrated that this computation is feasible to implement in computer programs.

We developed an approach where syntactic translations are used to provide semantics for HIIT signatures. Our approach seems to be a sweet spot for computing notions of induction and recursion in a formally verifiable and relatively simple way.

However, there is a coherence problem in the formal treatment of syntaxes and models of type theories internally to type theory. We sidestepped this by considering syntactic translations, but the problem remains, and it prevents extending the current approach to categorical semantics. Our impression is that the true solution will be the development of higher syntaxes and models of type theory in type theory. This may also require adding new features to the meta type theory. In particular, convenient formalization of higher categories seems elusive in conventional homotopy type theory, and we may need two-level type theories [14], or directed type-theories with native notions of higher categories [36, 39].

References

[1] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers — constructing strictly positive types. *Theoretical Computer Science*, 342:3–27, September 2005. Applied Semantics: Selected Topics.

[2] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proceedings of the ACM on Programming Languages*, 2(POPL):23, 2017.

[3] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the rezk completion. *Mathematical Structures in Computer Science*, 25(5):1010–1039, 2015.

[4] Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed categories, 2017.

[5] 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*, pages 293–310, Cham, 2018. Springer International Publishing.

[6] 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.

[7] Thorsten Altenkirch and Nicolai Kraus. Towards the syntax and semantics of higher dimensional type theory. 2018.

[8] Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 503–516. ACM, 2014.

[9] Henning Basold and Herman Geuvers. Type Theory based on Dependent Inductive and Coinductive Types. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of LICS '16*, pages 327–336. ACM, 2016.

[10] Henning Basold, Herman Geuvers, and Niels van der Weide. Higher inductive types in programming. *Journal of Universal Computer Science*, 23(1):63–88, jan 2017.

[11] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In *ACM Sigplan Notices*, volume 45, pages 345–356. ACM, 2010.

[12] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for free — parametricity for dependent types. *Journal of Functional Programming*, 22(02):107–152, 2012.

[13] 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.

[14] Paolo Capriotti and Nicolai Kraus. Univalent higher categories via complete semi-segal types. *Proc. ACM Program. Lang.*, 2(POPL):44:1–44:29, December 2017.

[15] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

[16] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. December 2015.

[17] Thierry Coquand, Simon Huber, and Anders Mrtberg. On higher inductive types in cubical type theory. *LICS '18: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, 2018.

[18] Floris van Doorn. Constructing the propositional truncation using non-recursive hits. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2016, pages 122–129, New York, NY, USA, 2016. ACM.

[19] Peter Dybjer. Internal type theory. In *International Workshop on Types for Proofs and Programs*, pages 120–134. Springer, 1995.

[20] Peter Dybjer. Inductive families. *Formal Aspects of Computing*, 6:440–465, 1997.

[21] Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65:525–549, 2000.

[22] 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 Mathematical Foundations of Programming Semantics (MFPS XXXIII).

[23] Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *Typed Lambda Calculi and Applications, volume 1581 of Lecture Notes in Computer Science*, pages 129–146. Springer, 1999.

[24] Eric Finster. Structure and equality in type systems. 2019.

[25] Michael Hedberg. A coherence theorem for Martin-Löf's type theory. *Journal of Functional Programming*, 8(04):413–436, 1998.

[26] Claudio Hermida, Uday S. Reddy, and Edmund P. Robinson. Logical relations and parametricity – a Reynolds programme for category theory and programming languages. *Electronic Notes in Theoretical Computer Science*, 303(0):149 – 180, 2014. Proceedings of the Workshop on Algebra, Coalgebra and Topology (WACT 2013).

[27] Martin Hofmann. *Extensional concepts in intensional type theory*. Thesis. University of Edinburgh, Department of Computer Science, 1995.

[28] Ambrus Kaposi. *Type theory in a type theory with quotient inductive types*. PhD thesis, University of Nottingham, 2017.

[29] 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.

[30] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proceedings of the ACM on Programming Languages*, 3(POPL):2, 2019.

[31] Nicolai Kraus. Constructions with non-recursive higher inductive types. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 595–604, New York, NY, USA, 2016. ACM.

[32] Marc Lasson. Canonicity of weak ω-groupoid laws using parametricity theory. *Electronic Notes in Theoretical Computer Science*, 308:229 – 244, 2014.

[33] Peter LeFanu Lumsdaine and Mike Shulman. Semantics of higher inductive types, 2017.

[34] Peter Morris and Thorsten Altenkirch. Indexed containers. In *Twenty-Fourth IEEE Symposium in Logic in Computer Science (LICS 2009)*, 2009.

[35] Fredrik Nordvall Forsberg. *Inductive-inductive definitions*. PhD thesis, Swansea University, 2013.

[36] Andreas Nuyts. *Towards a directed homotopy type theory based on 4 kinds of variance*. PhD thesis, MA thesis. KU Leuven, 2015.

[37] Christine Paulin-Mohring. Inductive definitions in the system Coq — rules and properties. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993.

[38] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013.

[39] Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *arXiv preprint arXiv:1705.07442*, 2017.

[40] 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.

[41] Thomas Streicher. *Semantics of type theory: correctness, completeness and independence results*. Springer Science & Business Media, 2012.

[42] Niels van der Weide. Higher inductive types. Master's thesis, Radboud University, Nijmegen, 2016.

[43] Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. Eliminating reflection from type theory. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 91–103. ACM, 2019.

Appendix A. The homomorphism translation

We list below the $-^{\mathsf{M}}$ interpretations for the syntax. For brevity, we mostly leave γ_0 and γ_1 implicit, notating only γ^M where necessary. We also omit some subscript parameters from J applications. The interpretation follows the same pattern as in the case of $-^{\mathsf{S}}$; the main difference is the interpretation of the universe as non-dependent function space. Additionally, the equality type former and reflexivity admit slightly nicer translations, since the lack of type dependency allows us to use composition and inv (which was noted in Section 2.2) instead of raw transports and path induction.

$$
\begin{aligned}
&\cdot^{\mathsf{M}}\,\gamma_0\,\gamma_1 &&:\equiv \top\\
&(\Delta, x : A)^{\mathsf{M}}\,\gamma_0\,\gamma_1 &&:\equiv (\gamma^M : \Delta^{\mathsf{M}}\,\gamma_0\,\gamma_1) \times A^{\mathsf{M}}\,\gamma^M\\
&x^{\mathsf{M}}\,\gamma^M &&:\equiv x^{\text{th}} \text{ component in } \gamma^M\\
&\mathsf{U}^{\mathsf{M}}\,\gamma^M\,A_0\,A_1 &&:\equiv A_0 \to A_1\\
&(\underline{a})^{\mathsf{M}}\,\gamma^M\,t_0\;t_1 &&:\equiv a^{\mathsf{M}}\,\gamma^M\,t_0 = t_1\\
&((x : a) \to B)^{\mathsf{M}}\,\gamma^M\,f_0\,f_1 &&:\equiv\\
&\qquad (x_0 : a^{\mathsf{A}}\,\gamma_0) \to B^{\mathsf{M}}\,(\gamma_0,\,x_0)\,(\gamma_1,\,a^{\mathsf{M}}\,\gamma^M\,x_0)\,(\gamma^M,\,\mathsf{refl})\,(f_0\,x_0)\,(f_1\,(a^{\mathsf{M}}\,\gamma^M\,x_0))\\
&(t\,u)^{\mathsf{M}}\,\gamma^M &&:\equiv \mathsf{J}\,(t^{\mathsf{M}}\,\gamma^M\,(u^{\mathsf{A}}\,\gamma_0))\,(u^{\mathsf{M}}\,\gamma^M)\\
&((x : A) \to B)^{\mathsf{M}}\,\gamma^M\,f_0\,f_1 &&:\equiv (x : A) \to B^{\mathsf{M}}\,\gamma^M\,(f_0\,x)\,(f_1\,x)\\
&(t\,u)^{\mathsf{M}}\,\gamma^M &&:\equiv t^{\mathsf{M}}\,\gamma^M\,u\\
&(t =_a u)^{\mathsf{M}}\,\gamma^M &&:\equiv \lambda\,e.\,(t^{\mathsf{M}}\,\gamma^{M\,-1}) \cdot \mathsf{ap}\,(a^{\mathsf{M}}\,\gamma^M)\,e \cdot u^{\mathsf{M}}\,\gamma^M\\
&(\mathsf{refl}_t)^{\mathsf{M}}\,\gamma^M &&:\equiv \mathsf{inv}\,(t^{\mathsf{M}}\,\gamma^M)\\
&(\mathsf{J}_{a\,t\,(x.z.p)}\,pr\;{}_u\,eq)^{\mathsf{M}}\,\gamma^M &&:\equiv\\
&\quad \mathsf{J}\Big(\mathsf{J}\Big(\mathsf{J}\,(\mathsf{J}\,(\lambda\,p_1\,p^M\,pr_1\,pr^M.\,pr^M)\,(t^{\mathsf{M}}\,\gamma^M)\\
&\qquad (\mathsf{uncurry}\,p^{\mathsf{A}}\,\gamma_1)\,(\mathsf{uncurry}\,p^{\mathsf{M}}\,\gamma^M)\,(pr^{\mathsf{A}}\,\gamma_1)\,(pr^{\mathsf{M}}\,\gamma^M))\,(eq^{\mathsf{A}}\,\gamma_0)\Big)\,(u^{\mathsf{M}}\,\gamma^M)\Big)\,(eq^{\mathsf{M}}\,\gamma^M)\\
&(\mathsf{J}\beta_{a\,t\,(x.z.p)}\,pr)^{\mathsf{M}}\,\gamma^M &&:\equiv\\
&\qquad \mathsf{J}\,(\mathsf{J}\,(\lambda\,p_1\,p^M.\,\mathsf{refl})\,(t^{\mathsf{M}}\,\gamma^M)\,(\mathsf{uncurry}\,p^{\mathsf{A}}\,\gamma_1)\,(\mathsf{uncurry}\,p^{\mathsf{M}}\,\gamma^M))\,(pr^{\mathsf{M}}\,\gamma^M)\\
&((x : A) \to b)^{\mathsf{M}}\,\gamma^M &&:\equiv \lambda f\,x.\;b^{\mathsf{M}}\,\gamma^M\,(f\,x)\\
&(t\,u)^{\mathsf{M}}\,\gamma^M &&:\equiv \mathsf{ap}\,(\lambda f.\,f\,u)\,(t^{\mathsf{M}}\,\gamma^M)
\end{aligned}
$$

Appendix B. Displayed algebra sections for the two-dimensional sphere

The two-dimensional sphere is given by the following signature:

$$\Gamma :\equiv \Big(\cdot,\;\; S^2 : \mathsf{U},\;\; b : \underline{S^2},\;\; surf : \underline{\mathsf{refl}_b =_{(b =_{S^2} b)} \mathsf{refl}_b}\Big)$$

The sphere-algebras are computed as follows.

$$\Gamma^{\mathsf{C}} \equiv \top \times (S^2 : \mathsf{Type}_0) \times (b : S^2) \times (surf : \mathsf{refl}_b =_{(b =_{S^2} b)} \mathsf{refl}_b)$$

Given a sphere-algebra and fixing a universe level i, the motives and methods are computed as follows.

$$\begin{aligned}
&\Gamma^{\mathsf{D}}\;(\mathsf{tt},\,S^2,\,b,\,\mathit{surf})\\
&\equiv \top\times(S^{2D}:S^2\to\mathsf{Type}_i)\\
&\quad\times(b^D:S^{2D}\,b)\\
&\quad\times(\mathit{surf}^D:\mathsf{tr}_{(\mathsf{tr}_{S^{2D}}\;-\;b^D=b^D)}\,\mathit{surf}\,\mathsf{refl}_{b^D}=\mathsf{refl}_{b^D})
\end{aligned}$$

Given a sphere-algebra and a displayed algebra over it, we get the type of sections:

$$\begin{aligned}
&\Gamma^{\mathsf{S}}\;(\mathsf{tt},\,S^2,\,b,\,\mathit{surf})\,(\mathsf{tt},\,S^{2D},\,b^D,\,\mathit{surf}^D)\\
&\equiv \top\times(S^{2S}:(x:S^2)\to S^{2D}\,x)\\
&\quad\times(b^S:S^{2S}\;b=b^D)\\
&\quad\times\Big(\mathit{surf}^S\;:\mathsf{tr}\,\big(\mathsf{J}\,\mathsf{refl}\,b^S\big)\,\big(\mathsf{tr}\,\big(\mathsf{J}\,\mathsf{refl}\,b^S\big)\,\big(\mathsf{apd}\,(\lambda x.\mathsf{tr}\,b^S\,(\mathsf{tr}\,b^S\,(\mathsf{apd}\,S^{2S}\,x)))\,\mathit{surf}\big)\big)\\
&\qquad\qquad=\mathit{surf}^D\Big)
\end{aligned}$$

Note that if b^S is a definitional equality (that is, we have $S^{2S}\;b{\equiv}b^D$), the occurrences of b^S in the type of surf^S can be replaced by refl. In this case the type of surf^S becomes the more usual $\mathsf{apd}\,(\mathsf{apd}\,S^{2S})\,\mathit{surf}=\mathit{surf}^D$.

Appendix C. Displayed algebra sections for indexed W-types

Indexed W-types can describe a large class of inductive definitions [34]. Suppose we have the external context $I:\mathsf{Type}_0, S:\mathsf{Type}_0, P:S\to\mathsf{Type}_0, \mathit{out}:S\to I, \mathit{in}:(s:S)\to P\,s\to I$. Then, the signature for the corresponding indexed W-type is the following:

$$W:\equiv(\cdot,\;\;w:(i:I)\to\mathsf{U},\;\;\mathit{sup}:(s:S)\to((p:P\,s)\to w\,(\mathit{in}\,s\,p))\to\underline{w\,(\mathit{out}\,s)})$$

We pick a universe level j for elimination. The interpretations of W are the following, omitting leading $\top$ components:

$$\begin{aligned}
&W^{\mathsf{A}}\equiv(w:I\to\mathsf{Type}_0)\times((s:S)\to((p:P\,s)\to w\,(\mathit{in}\,s\,p))\to w\,(\mathit{out}\,s))\\
&W^{\mathsf{D}}\,(w,\mathit{sup})\equiv(w^D:(i:I)\to w\,i\to\mathsf{Type}_j)\\
&\qquad\qquad\times\big((s:S)(f:(p:P\,s)\to w\,(\mathit{in}\,s\,p))\\
&\qquad\qquad\quad\to((p:P\,s)\to w^D\,(\mathit{in}\,s\,p)\,(f\,p))\to w^D\,(\mathit{out}\,s)\,(\mathit{sup}\,s\,f)\big)\\
&W^{\mathsf{S}}\,(w,\mathit{sup})\,(w^D,\mathit{sup}^D)\equiv\\
&\qquad\qquad(w^S:(i:I)(x:w\,i)\to w^D\,i\,x)\\
&\qquad\qquad\times\big((s:S)(f:(p:P\,s)\to w\,(\mathit{in}\,s\,p))\\
&\qquad\qquad\quad\to w^S\,(\mathit{out}\,s)\,(\mathit{sup}\;s\;f)=\mathit{sup}^D\;s\,f\,(\lambda p.\,w^S\,(\mathit{in}\,s\,p)\,(f\,p))\big)
\end{aligned}$$