Constructing Quotient Inductive-Inductive Types

AMBRUS KAPOSI, Eötvös Loránd University, Hungary
ANDRÁS KOVÁCS, Eötvös Loránd University, Hungary
THORSTEN ALTENKIRCH, University of Nottingham, United Kingdom

Quotient inductive-inductive types (QIITs) generalise inductive types in two ways: a QIIT can have more than one sort and the later sorts can be indexed over the previous ones. In addition, equality constructors are also allowed. We work in a setting with uniqueness of identity proofs, hence we use the term QIIT instead of higher inductive-inductive type. An example of a QIIT is the well-typed (intrinsic) syntax of type theory quotiented by conversion. In this paper first we specify finitary QIITs using a domain-specific type theory which we call the theory of signatures. The syntax of the theory of signatures is given by a QIIT as well. Then, using this syntax we show that all specified QIITs exist and they have a dependent elimination principle. We also show that algebras of a signature form a category with families (CwF) and use the internal language of this CwF to show that dependent elimination is equivalent to initiality.

CCS Concepts: • **Theory of computation** → **Type theory**;

Additional Key Words and Phrases: homotopy type theory, inductive-inductive types, higher inductive types, quotient inductive types, logical relations, category with families, generalised algebraic theory

ACM Reference Format:
Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. 2019. Constructing Quotient Inductive-Inductive Types. *Proc. ACM Program. Lang.* 3, POPL, Article 2 (January 2019), 24 pages. https://doi.org/10.1145/3290315

1 INTRODUCTION

A quotient inductive-inductive type (QIIT) can be seen as a multi-sorted algebraic theory where sorts can be indexed over each other. An example of a QIIT is the following well-typed (intrinsic) syntax of a small type theory.

$$\begin{aligned}
\mathsf{Con} &: \mathsf{Set}\\
\mathsf{Ty} &: \mathsf{Con} \to \mathsf{Set}\\
\cdot &: \mathsf{Con}\\
\rhd &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}\\
\iota &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma\\
\Sigma &: (\Gamma : \mathsf{Con}) \to (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \rhd A) \to \mathsf{Ty}\,\Gamma\\
\mathsf{eq} &: (\Gamma : \mathsf{Con}) \to (A : \mathsf{Ty}\,\Gamma) \to (B : \mathsf{Ty}\,(\Gamma \rhd A)) \to (\Gamma \rhd A \rhd B = \Gamma \rhd \Sigma\,\Gamma\,A\,B)
\end{aligned}$$

It has two sorts: contexts (Con) and types (Ty). The latter is indexed over the former: to talk about a type we need to say which context it lives in. There is an empty context · and context extension

Authors' addresses: Ambrus Kaposi, Department of Programming Languages and Compilers, Eötvös Loránd University, Budapest, Hungary, akaposi@inf.elte.hu; András Kovács, Department of Programming Languages and Compilers, Eötvös Loránd University, Budapest, Hungary, kovacsandras@inf.elte.hu; Thorsten Altenkirch, School of Computer Science, University of Nottingham, United Kingdom, Thorsten.Altenkirch@nottingham.ac.uk.

2475-1421/2019/1-ART2
https://doi.org/10.1145/3290315

$\triangleright$ which takes a context and a type in that context and returns the extended context. Note that we cannot turn this QIIT into two inductive types defined one after the other because the Con-constructor $\triangleright$ refers to Ty, hence Con and Ty must be defined *at the same time*. There is a constructor for a base type ι in any context and a constructor for Σ types. This takes a context (this could be made an implicit parameter), a type A in that context and a type in the context extended by A and returns a type in the original context. The third argument of Σ shows a pattern which does not appear in inductive types or indexed inductive types: a constructor refers to a previous constructor (in our case $\triangleright$). Finally, there is an equality constructor which states the unusual equality (included here for illustration) that extending a context twice is the same as extending by a Σ type. Equality constructors can only target one of the sorts. The constructor eq quotients contexts so that for any Γ, A and B it becomes impossible to distinguish $(\Gamma \triangleright A \triangleright B)$ and $(\Gamma \triangleright \Sigma\,\Gamma\,A\,B)$ – this is ensured by the eliminator of the QIIT as shown below. The equality constructor also refers to previous constructors. This small example can be extended to the full syntax of type theory as shown in [Altenkirch and Kaposi 2016].

In this paper we define the *theory of signatures*, a small type theory. It is itself given as a QIIT, and we show that if a type theory supports this QIIT, then it supports all finitary QIITs. This is analogous to the following results: if a type theory has W-types [Abbott et al. 2005], then it has all inductive types; if a type theory has indexed W-types [Morris and Altenkirch 2009], then it has all indexed inductive types.

The theory of signatures is a restriction of the theory of codes in [Kaposi and Kovács 2018]. A signature for a QIIT is given by a context in this type theory. For example, the context for the above Con-Ty example is $(Con : \mathsf{U}, Ty : Con \to \mathsf{U}, \cdot : Con, \triangleright : (\Gamma : Con) \to Ty\,\Gamma \to Con, \ldots)$ where Con, Ty, $\cdot$, $\triangleright$ are simply variable names.

By induction on the syntax of the theory of signatures, we define what *algebras* are for each signature and we construct the *initial algebra*. Then we define *algebra homomorphisms* and a homomorphism from the initial algebra to any other algebra called the *recursor*. In a similar way, we define *displayed algebras over algebras*, *sections of displayed algebras* and the *eliminator* for each signature.

In the following table we summarise how the above notions correspond to other concepts in the literature:

signature	code, specification, arities of operators
algebra	model, sets with operations and equations
initial algebra	type formation rules and constructors, free algebra
algebra homomorphism	morphism of models
recursor	non-dependent eliminator, iterator, fold, catamorphism
displayed algebra	motives and methods of the eliminator, fibration
section of a displayed algebra	dependent function which respects the operators
eliminator	induction principle, dependent eliminator

Additionally, we show that algebras of a signature form a category with families (CwF), where displayed algebras and sections yield the "F" part of CwF, and these CwFs also support *constant families* in the sense of [Nordvall Forsberg 2013, p. 74] (or equivalently they are *democratic* [Clairambault and Dybjer 2014]) and extensional identity types. This yields a small *internal language* of algebras for each signature, and it allows us to prove that unique recursion (initiality) is equivalent to dependent elimination (induction).

In the rest of the introduction, before giving an overview of the paper, we illustrate our method for deriving the above notions by three examples. We start with a closed type which has a recursive constructor, the natural numbers. Then we move on to a parameterised type with an equality constructor: the integers. Finally, we sketch how our method works for the above Con-Ty example. Our notation below is standard, but we summarise it in Section 2.

1.1 Natural Numbers

The theory of signatures is a small internal type theory with a universe, two restricted function spaces and an identity type. We will use the following notation for the theory of signatures. $\mathsf{Ty}\,\Gamma$ denotes well-formed types in a context Γ. Given $A : \mathsf{Ty}\,\Gamma$, $\mathsf{Tm}\,\Gamma\,A$ denotes well-typed terms in context Γ of type A. The following three-element context is the signature for natural numbers: it has one sort and two operators, zero and successor. [1]

$$\Delta :\equiv (Nat : \mathsf{U},\ zero : \mathsf{El}\,Nat,\ suc : Nat \Rightarrow \mathsf{El}\,Nat)$$

U is the universe (the type of codes), El decodes a code into a type. We call types which come from a code *small*, other types *large*. The function space $\Rightarrow$ in the theory of signatures has a small domain and a large codomain and is itself large. This ensures strict positivity of the operators in the signature.

Algebras, homomorphisms, the initial algebra, the recursor etc. are defined by induction on the syntax of the theory of signatures in later sections. Here we only describe them informally and show their output on the signature Δ.

The operation $-^{\mathsf{A}}$ computes the set of algebras from a signature. A natural number algebra is an iterated Σ-type: a set together with an element of the set and an endofunction on the set.

$$\Delta^{\mathsf{A}} \equiv (N : \mathsf{Set}) \times N \times (N \to N)$$

The operation $-^{\mathsf{A}}$ is the standard interpretation of the syntax which is sometimes called the metacircular interpretation or interpretation into the set model [Altenkirch and Kaposi 2016]. For a context Γ it gives $\Gamma^{\mathsf{A}} : \mathsf{Set}$, for type $A : \mathsf{Ty}\,\Gamma$ it gives $A^{\mathsf{A}} : \Gamma^{\mathsf{A}} \to \mathsf{Set}$ and for a term $t : \mathsf{Tm}\,\Gamma\,A$ it produces $t^{\mathsf{A}} : (\gamma : \Gamma^{\mathsf{A}}) \to A^{\mathsf{A}}\,\gamma$.

The initial Δ-algebra is given by $\mathsf{con}_\Delta : \Delta^{\mathsf{A}}$. The idea is that natural numbers are terms of type Nat in the context Δ, with the intuitive justification that the only way to form terms of type Nat in this context is using $zero$ or suc.

$$\mathsf{con}_\Delta \equiv (\mathsf{Tm}\,\Delta\,(\mathsf{El}\,Nat),\ zero,\ \lambda t.suc \mathbin{@} t)$$

The zero operator is given by the variable $zero$, successor is given by a function which takes a term t and applies it to the variable suc. Application in the theory of signatures is denoted $@$.

A Δ-algebra homomorphism between two Δ-algebras (A, a, f) and (B, b, g) is given by a function between the two sets which respects the operators. Propositional equality is denoted $=$.

$$\Delta^{\mathsf{M}}\,(A, a, f)\,(B, b, g) \equiv (N^M : A \to B) \times (N^M\,a = b) \times ((x : A) \to N^M\,(f\,x) = g\,(N^M\,x))$$

The operation $-^{\mathsf{M}}$ is a modified binary logical relation interpretation [Bernardy et al. 2012]: a context Γ is interpreted as a relation $\Gamma^{\mathsf{M}} : \Gamma^{A} \to \Gamma^{A} \to \mathsf{Set}$.

We use the standard interpretation $-^{A}$ to show weak initiality: given an algebra $(A, a, f) : \Delta^{\mathsf{A}}$, the homomorphism $\mathsf{rec}_\Delta\,(A, a, f) : \Delta^{\mathsf{M}}\,\mathsf{con}_\Delta\,(A, a, f)$ is given by

$$\mathsf{rec}_\Delta\,(A, a, f) \equiv (\lambda t.t^{\mathsf{A}}\,(A, a, f),\ \mathsf{refl}_a,\ \lambda x.\mathsf{refl}_{f\,(x^{\mathsf{A}}\,(A, a, f))}).$$

For a natural number t, its interpretation in the algebra (A, a, f) is given by its standard interpretation at (A, a, f). This has the right type because $(\mathsf{El}\,Nat)^{\mathsf{A}}\,(A, a, f) \equiv A$ by the standard

[1] To improve readability we use named variables to describe contexts in the theory of signatures, while later we formally only define de Bruijn-like combinators. Note that the : is now overloaded, it is used in the metatheory and in the theory of signatures.

interpretation of $\mathsf{El}\,\mathit{Nat}$. The recursor computes a for *zero* as its standard interpretation is just the corresponding component: $\mathit{zero}^{\mathsf{A}}\,(A,a,f) \equiv a$. Similarly we have that $(\mathit{suc} \mathbin{@} x)^{\mathsf{A}}\,(A,a,f) \equiv \mathit{suc}^{\mathsf{A}}\,(A,a,f)\,(x^{\mathsf{A}}\,(A,a,f)) \equiv f\,(x^{\mathsf{A}}\,(A,a,f))$. In essence, the standard interpretation folds over terms, substituting a function for *suc* and a value for *zero* - which is exactly recursion for natural numbers.

A displayed algebra over an algebra (N,z,s) consists of a proof-relevant predicate over N, a witness of the predicate at z and a proof that s respects the predicate. We borrow the term "displayed" from [Ahrens and Lumsdaine 2017], as our notion of displayed algebra is a generalization of the displayed categories of Ibid.[2]

$$\Delta^{\mathsf{D}}\,(N,z,s) \equiv (N^D : N \to \mathsf{Set}) \times N^D\,z \times ((x:N) \to N^D\,x \to N^D\,(s\,x))$$

The operation $-^{\mathsf{D}}$ is the unary logical predicate interpretation [Bernardy et al. 2012]: a context Γ is interpreted as a predicate $\Gamma^{\mathsf{D}} : \Gamma^{\mathsf{A}} \to \mathsf{Set}$. A type $A : \mathsf{Ty}\,\Gamma$ becomes a predicate depending on a witness of Γ^{D}, that is, $A^{\mathsf{D}} : \Gamma^{\mathsf{D}}\,\gamma \to A^{\mathsf{A}}\,\gamma \to \mathsf{Set}$, where we implicitly quantify over γ. A term $t : \mathsf{Tm}\,\Gamma\,A$ is interpreted as $t^{\mathsf{D}} : (\gamma^D : \Gamma^{\mathsf{D}}\,\gamma) \to A^{\mathsf{D}}\,\gamma^D\,(t^{\mathsf{A}}\,\gamma)$.

A section of a displayed algebra (N^D,z^D,s^D) over (N,z,s) is given by a section of the predicate N^D which respects the operations.

$$\begin{aligned}\Delta^{\mathsf{S}}\,(N,z,s)\,(N^D,z^D,s^D) \equiv {} & (N^S : (x:N) \to N^D\,x) \times (N^S\,z = z^D) \times \\ & \big((x:N) \to N^D\,x \to N^S\,(s\,x) = s^D\,x\,(N^S\,x)\big)\end{aligned}$$

The operation $-^{\mathsf{S}}$ is a modified dependent logical relation interpretation: a context Γ is interpreted as a dependent relation $\Gamma^{\mathsf{S}} : (\gamma : \Gamma^{\mathsf{A}}) \to \Gamma^{\mathsf{D}}\,\gamma \to \mathsf{Set}$.

Given a displayed algebra (N^D,z^D,s^D) over the initial algebra con_Δ, we construct a section which we call the eliminator. It has type $\Delta^{\mathsf{S}}\,\mathsf{con}_\Delta\,(N^D,z^D,s^D)$ — the two equations in the definition of sections correspond to the β-rules.

$$\begin{aligned}\mathsf{elim}_\Delta\,(N^D,z^D,s^D) \equiv {} & \big(\lambda t.\mathsf{tr}_{N^D}\,(t^{\mathsf{C}}\,\mathsf{id}^{-1})\,(t^{\mathsf{D}}\,(N^D,z^D,s^D)),\ \mathsf{refl}_{z^D}, \\ & \lambda x\,x^D.\mathsf{J}\,\mathsf{refl}_{s^D\,x\,(x^{\mathsf{D}}\,(N^D,z^D,s^D))}\,(x^{\mathsf{C}}\,\mathsf{id})\big)\end{aligned}$$

We can eliminate any natural number t using the logical predicate interpretation $t^{\mathsf{D}}\,(N^D,z^D,s^D) : N^D\,(t^{\mathsf{A}}\,\mathsf{con}_\Delta)$. The result has to be transported along the equality $t^{\mathsf{C}}\,\mathsf{id} : (t = t^{\mathsf{A}}\,\mathsf{con}_\Delta)$ so that we get something of type $N^D\,t$. The operation $-^{\mathsf{C}}$ is a generalisation of con; as we will see later, con is defined in terms of $-^{\mathsf{C}}$. The usage of $t^{\mathsf{C}}\,\mathsf{id}$ corresponds to the identity extension lemma [Atkey et al. 2014]. The computation rule for zero is definitional (can be proved by refl), but the case for successor requires using J on $x^{\mathsf{C}}\,\mathsf{id}$.

1.2 Integers

Assuming that we have natural numbers in our metatheory (with $\mathbb{N} : \mathsf{Set}$, $+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$), integers are specified by the following signature.

$$\begin{aligned}\Phi :\equiv \big(&\mathit{Int} : \mathsf{U},\ \mathit{pair} : \mathbb{N} \mathrel{\hat{\Rightarrow}} \mathbb{N} \mathrel{\hat{\Rightarrow}} \mathsf{El}\,\mathit{Int}, \\ & \mathit{eq} : (a\,b\,c\,d : \mathbb{N}) \mathrel{\hat{\Rightarrow}} a + d = b + c \mathrel{\hat{\Rightarrow}} \mathsf{Id}\,\mathit{Int}\,(\mathit{pair} \mathbin{\hat{@}} a \mathbin{\hat{@}} b)\,(\mathit{pair} \mathbin{\hat{@}} c \mathbin{\hat{@}} d)\big)\end{aligned}$$

The operator *pair* uses a function space different from the one used for *suc* in section 1.1. An $\Rightarrow$ function has small domain and large codomain, while $\hat{\Rightarrow}$ has metatheoretic domain and large codomain. This lets us specify parameterised types, allowing integers to refer to the set of natural numbers and addition. The *eq* operator takes four natural numbers and a metatheoretic equality

[2]However, we work in a setting with UIP, while Ahrens and Lumsdaine work in homotopy type theory.

between them and returns an identity between the appropriate pairs in the theory of signatures. $\hat{@}$ is application for the $\hat{\Rightarrow}$ function space. Id is the constructor for the identity type. It is indexed by two elements of a small type and produces a large type — this prevents us from writing iterated equality types.

The $\hat{\Rightarrow}$ function space is converted to metatheoretic function space by the $-^{\mathsf{A}}$ operation, and Id is interpreted as metatheoretic equality.

$$\Phi^{\mathsf{A}} \equiv (I : \mathsf{Set}) \times (p : \mathbb{N} \to \mathbb{N} \to I) \times \big((a\,b\,c\,d : \mathbb{N}) \to a + d = b + c \to p\,a\,b = p\,c\,d\big)$$

The initial algebra is given by the terms of type $\mathsf{El}\,\mathit{Int}$ in context Φ. The identity type Id has the equality reflection rule which says that if there is a term of type $\mathsf{Id}\,\mathit{Int}\,t\,u$, then we have $t = u$ (conversion in the theory of signatures is given by propositional equality in the metatheory). Hence, terms of type $\mathsf{El}\,\mathit{Int}$ in Φ are already quotiented by eq, through equality reflection.

$$\mathsf{con}_\Phi \equiv \big(\mathsf{Tm}\,\Phi\,(\mathsf{El}\,\mathit{Int}),\ \lambda a\,b.\mathit{pair}\,\hat{@}\,a\,\hat{@}\,b,\ \lambda a\,b\,c\,d\,e.\mathsf{reflect}\,(eq\,\hat{@}\,a\,\hat{@}\,b\,\hat{@}\,c\,\hat{@}\,d\,\hat{@}\,e)\big) : \Phi^{\mathsf{A}}$$

The pair operator is given by the *pair* variable applied to the two natural number inputs, and the equality is given by equality reflection on the *eq* variable.

A Φ-homomorphism is given by a function which respects the *pair* operators. The component for *eq* is trivial ($\top$) as we have uniqueness of identity proofs (UIP) in the metatheory, so there is no need to relate the equality proofs e and e'.

$$\Phi^{\mathsf{M}}\,(I, p, e)\,(I', p', e') \equiv (I^M : I \to I') \times \big((a\,b : \mathbb{N}) \to I^M\,(p\,a\,b) = p'\,a\,b\big) \times \top$$

The recursor is given using $-^{\mathsf{A}}$ as in the case of natural numbers. It is reflexivity for the *pair* constructor and it just returns the single element of $\top$ for the *eq* constructor.

$$\mathsf{rec}_\Phi\,(I, p, e) \equiv (\lambda t.t^{\mathsf{A}}\,(I, p, e),\ \lambda a\,b.\mathsf{refl}_{p\,a\,b},\ \lambda a\,b\,c\,d\,e.\mathsf{tt}) : \Phi^{\mathsf{A}}.$$

A displayed algebra over a Φ-algebra (I, p, e) is an element of the following set.

$$\begin{aligned}\Phi^{\mathsf{D}}\,(I, p, e) \equiv{} & (I^D : I \to \mathsf{Set}) \times \big(p^D : (a\,b : \mathbb{N}) \to I^D\,(p\,a\,b)\big) \times \\ & \big((a\,b\,c\,d : \mathbb{N}) \to (e : a + d = b + c) \to \mathsf{tr}_{I^D}\,(e\,a\,b\,c\,d\,e)\,(p^D\,a\,b) = p^D\,c\,d\big)\end{aligned}$$

We need a predicate on I, a witness of the predicate at $p\,a\,b$ for all a and b, and a proof that the two witnesses are equal. The types of the witnesses can be shown equal by e, we have to transport over this.

A section of a displayed algebra (I^D, p^D, e^D) over (I, p, e) is an element of the following set.

$$\Phi^{\mathsf{S}}\,(I, p, e)\,(I^D, p^D, e^D) \equiv (I^S : (i : I) \to I^D\,i) \times \big((a\,b : \mathbb{N}) \to I^D\,(p\,a\,b) = p^D\,a\,b\big) \times \top$$

The eliminator can be defined using $-^{\mathsf{D}}$ and $-^{\mathsf{C}}$ as in the case of natural numbers.

1.3 Contexts and Types

The previous Con-Ty example is represented by the context below. We only present here a prefix of the definition which suffices to demonstrate inductive-inductive dependency.

$$\Theta :\equiv \big(\mathit{Con} : \mathsf{U},\ \mathit{Ty} : \mathit{Con} \Rightarrow \mathsf{U},\ \cdot : \mathsf{El}\,\mathit{Con},\ \rhd : (\Gamma : \mathit{Con}) \Rightarrow \mathit{Ty}\,@\,\Gamma \Rightarrow \mathsf{El}\,\mathit{Con}, \ldots\big)$$

Algebras for this signature consist of a set, a family of sets over it, and operators which construct elements of these.

$$\Theta^{\mathsf{A}} \equiv (C : \mathsf{Set}) \times (T : C \to \mathsf{Set}) \times (e : C) \times (f : (\gamma : C) \to T\,\gamma \to C) \times \ldots$$

The initial algebra now has two sorts: one is given by terms of type *Con*, the other takes a *Con*-term as an input and outputs the type of terms of type *El* at the *Con*-term.

$$\mathsf{con}_\Theta \equiv \big(\mathsf{Tm}\,\Theta\,(\mathsf{El}\,\mathit{Con}),\ \lambda t.\mathsf{Tm}\,\Theta\,(\mathsf{El}\,(\mathit{Ty}\,@\,t)),\ \cdot,\ \lambda t\,r.\rhd\,@\,t\,@\,r, \ldots\big) : \Theta^{\mathsf{A}}$$

An algebra homomorphism is given by a function between the *Con* components and a function between the *Ty* components which refers to the first function (this phenomenon is called recursion-recursion in [Nordvall Forsberg 2013] as an analogue to induction-induction).

$$\begin{aligned}\Theta^{\mathsf{M}}\,(C,T,e,f,\dots)\,(C',T',e',f',\dots) \equiv\ & (C^M : C \to C') \times (T^M : (\gamma : C) \to T\,\gamma \to T'\,(C^M\,\gamma)) \times \\ & (e^M : C^M\,e = e') \times \\ & (f^M : (\gamma : C)(\alpha : T\,\gamma) \to C^M\,(f\,\gamma\,\alpha) = f'\,(C^M\,\gamma)\,(T^M\,\gamma\,\alpha)) \times \\ & \dots\end{aligned}$$

The first function in the recursor invokes the standard interpretation on its input, the second one invokes it on its second input. We abbreviate $(C, T, e, f, \dots)$ by γ.

$$\mathsf{rec}_\Phi\,\gamma \equiv (\lambda t.t^{\mathsf{A}}\,\gamma,\ \lambda t\,r.r^{\mathsf{A}}\,\gamma,\ \mathsf{refl}_e,\ \lambda t\,r.\mathsf{refl}_{f\,(t^{\mathsf{A}}\,\gamma)\,(r^{\mathsf{A}}\,\gamma)}, \dots) : \Theta^{\mathsf{A}}$$

A displayed algebra over an algebra $(C, T, e, f, \dots)$ consists of a family over C and a family over $T\,\gamma$ which is also indexed over the first family.

$$\begin{aligned}\Theta^{\mathsf{D}}\,(C,T,e,f,\dots) \equiv\ & (C^D : C \to \mathsf{Set}) \times (T^D : (\gamma : C) \to C^D\,\gamma \to T\,\gamma \to \mathsf{Set}) \times \\ & (e^D : C^D\,e) \times \\ & (f^D : (\gamma : C)(\gamma^D : C^D\,\gamma)(\alpha : T\,\gamma)(\alpha^D : T^D\,\gamma\,\gamma^D\,\alpha) \to C^D\,(f\,\gamma\,\alpha)) \times \dots\end{aligned}$$

As in the case of the homomorphisms, the second function in a section refers to the first one.

$$\begin{aligned}& \Theta^{\mathsf{S}}\,(C,T,e,f,\dots)\,(C^D,T^D,e^D,f^D,\dots) \equiv \\ & \quad (C^S : (\gamma : C) \to C^D\,\gamma) \times (T^S : (\gamma : C)(\alpha : T\,\gamma) \to T^D\,\gamma\,(C^S\,\gamma)\,\alpha) \times \\ & \quad (e^S : C^S\,e = e^D) \times \\ & \quad (f^S : (\gamma : C)(\alpha : T\,\gamma) \to C^S\,(f\,\gamma\,\alpha) = f^D\,\gamma\,(C^S\,\gamma)\,\alpha\,(T^S\,\gamma\,\alpha)) \times \dots\end{aligned}$$

The eliminator is given analogously to the recursor, but using $-^{\mathsf{D}}$ and $-^{\mathsf{C}}$ instead of $-^{\mathsf{A}}$.

1.4 Overview of the Rest of the Paper

After a discussion of related work, we describe the metatheory in Section 2. We define the type theory of signatures in Section 3. We define algebras and the initial algebra for each signature in Section 4, homomorphisms and the recursor in Section 5, displayed algebras, sections and the eliminator in Section 6. In Section 7 we extend algebras and homomorphisms to a categories with families (CwF) model of the theory of signatures and use this to show that initiality is equivalent to dependent elimination. We conclude in Section 8.

In the following table we summarize the operations which we define in Sections 4–6. Γ denotes a signature. The full definitions of these operations are given in Appendix A[3].

[3]Available at https://bitbucket.org/akaposi/finitaryqiit/raw/master/appendix.pdf

Γ^{A}	$: \mathsf{Set}$	the set of Γ-algebras	Sec. 4
$\Gamma^{\mathsf{C}_\Omega}$	$: \mathsf{Sub}\,\Omega\,\Gamma \to \Gamma^{\mathsf{A}}$	helper for initial algebra	
con_Γ	$: \Gamma^{\mathsf{A}}$	initial algebra: $\mathsf{con}_\Gamma :\equiv \Gamma^{\mathsf{C}_\Gamma}\,\mathsf{id}_\Gamma$	
Γ^{M}	$: \Gamma^{\mathsf{A}} \to \Gamma^{\mathsf{A}} \to \mathsf{Set}$	homomorphisms of Γ-algebras	Sec. 5
$\Gamma^{\mathsf{R}_{\Omega,\omega}}$	$: (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^{\mathsf{M}}\,(\nu^{\mathsf{A}}\,\mathsf{con}_\Omega)\,(\nu^{\mathsf{A}}\,\omega)$	helper for the recursor	
rec_Γ	$: (\gamma : \Gamma^{\mathsf{A}}) \to \Gamma^{\mathsf{M}}\,\mathsf{con}_\Gamma\,\gamma$	recursor: $\mathsf{rec}_\Gamma\,\gamma :\equiv \Gamma^{\mathsf{R}_{\Gamma,\gamma}}\,\mathsf{id}_\Gamma$	
Γ^{D}	$: \Gamma^{\mathsf{A}} \to \mathsf{Set}$	displayed algebras over Γ-algebras	Sec. 6
Γ^{S}	$: (\gamma : \Gamma^{\mathsf{A}}) \to \Gamma^{\mathsf{D}}\,\gamma \to \mathsf{Set}$	sections of displayed algebras	
$\Gamma^{\mathsf{E}_{\Omega,\omega^D}}$	$: (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^{\mathsf{S}}\,(\nu^{\mathsf{A}}\,\mathsf{con})\,(\nu^{\mathsf{D}}\,\omega^D)$	helper for the eliminator	
elim_Γ	$: (\gamma^D : \Gamma^{\mathsf{D}}\,\mathsf{con}_\Gamma) \to \Gamma^{\mathsf{S}}\,\mathsf{con}_\Gamma\,\gamma^D$	eliminator: $\mathsf{elim}_\Gamma\,\gamma^D :\equiv \Gamma^{\mathsf{E}_{\Gamma,\gamma^D}}\,\mathsf{id}_\Gamma$	

1.5 Related Work

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 [Abbott et al. 2005]. Morris and Altenkirch [Morris and Altenkirch 2009] extend the notion of container to that of indexed container which specifies indexed inductive types. External schemes for inductive families are given in [Dybjer 1997; Paulin-Mohring 1993], for inductive-recursive types in [Dybjer 2000]. Inductive-inductive types were introduced by Nordvall Forsberg together with an internal coding scheme [Nordvall Forsberg 2013]. A symmetric scheme for both inductive and coinductive types is given in [Basold and Geuvers 2016].

Quotient types as in [Hofmann 1995b] are a precursor to the current development. More recently, an increasing amount of research is concerned with higher inductive types (HITs), motivated by their use cases in homotopy type theory [The Univalent Foundations Program 2013]. The theory of HITs is relevant to the current work as it also has to describe signatures with equality constructors and algebras with equalities. [Basold et al. 2017] define an external syntactic scheme for higher inductive types with only 0-constructors and compute the types of elimination principles. In [van der Weide 2016] a semantics is given for the same class of HITs but with no recursive equality constructors. Sojakova [Sojakova 2015] defines a subset of HITs called W-suspensions by an internal coding scheme similar to W-types. She proves that the induction principle is equivalent to homotopy initiality. Dybjer and Moeneclaey define a syntactic scheme for some HITs and show their existence in a groupoid model [Dybjer and Moeneclaey 2018]. In [Awodey et al. 2018], impredicative encodings of a class of higher inductive types are presented, and we think that this approach could also be modelled within our framework once we assume an impredicative universe.

Lumsdaine and Shulman give a general specification of models of type theory supporting higher inductive types [Lumsdaine and Shulman 2017]. They introduce the notion of cell monad with parameters and characterise the class of models which have intial algebras for a cell monad with parameters. [Coquand et al. 2018] develop semantics for several HITs (sphere, torus, suspensions, truncations, pushouts) in certain presheaf toposes, and extend the syntax of cubical type theory [Cohen et al. 2016] with these HITs.

[Altenkirch et al. 2018] also concerns QIITs. There, signatures are given by a scheme which builds up complete categories of algebras from lists of functors. This notion of signature is more semantic than ours, with more overhead in encoding signatures. It also does not enforce strict positivity, hence the paper does not construct initial algebras. Nonetheless, it is shown that well-behaved QIIT signatures are covered by this scheme. Also, induction is shown to be equivalent to initiality, but the notion of induction is given only up to isomorphisms of algebras, in contrast to the current work, where it is computed strictly.

Cartmell [Cartmell 1986] also used a type-theoretic syntax for a class of theories, called generalised algebraic theories (GATs). GATs cover roughly the same theories as this work does, with the difference that GATs additionally allow equations between sorts. Besides this, there are two main differences to the current work.

First, regarding the definitions of the signatures: GATs are given by a nameful presyntax along with well-formedness relations, while our signatures are intrinsically typed and given by structured categories. This greatly simplifies formal development, makes machine-checked formalisation feasible, and also clarifies the notion of induction over signatures.

Second, the focus of the current work differs. Cartmell's main result was the establishment of contextual categories as classifying categories of GATs. In contrast, we do not discuss classifying categories. Instead, we present more constructions involving algebras; these include an explicit initial algebra construction and the CwF model, which yields a small internal type theory of algebras for each signature, and which in particular allows us to exactly compute eliminators and homomorphisms as types in the meta type theory.[4]

Our notion of displayed algebra is analogous to displayed categories in [Ahrens and Lumsdaine 2017], and our definition of total algebras and reindexing is also analogous to the corresponding notions for displayed categories.

2 METATHEORY AND FORMALISATION

Our metatheory is Martin-Löf type theory with functional extensionality and uniqueness of identity proofs (UIP). In this section we describe the notation used in this paper and the accompanying technical appendix and Agda formalisation.

Definitional equality is denoted by $\equiv$. We have a cumulative hierarchy of Russell-style universes Set_i where we usually omit indices (we don't assume any impredicativity). Dependent function space is denoted $(\alpha : T) \to T'$. $T \to T'$ stands for $(\alpha : T) \to T'$ if T' does not depend on α. We use $(\alpha : T)(\alpha' : T') \to T''$ as a shorthand for iterated function spaces. We sometimes omit certain arguments of functions or write them in subscript to lighten the notation. Σ types are denoted by $(\alpha : T) \times T'$ with left-associative $\times$. The constructor is $(-,-)$ and the eliminators are written proj_1 and proj_2. The one-element type $\top$ has constructor tt. We have the identity type $=$ with constructor refl, eliminator J. The notation is $\mathsf{J}_{x.z.P}\, pr\, e : P[x \mapsto \alpha', z \mapsto e]$ for $\alpha, \alpha' : T$ and $x : T, z : \alpha = x \vdash P : \mathsf{Set}$ and $pr : P[x \mapsto \alpha, z \mapsto \mathsf{refl}]$ and $e : \alpha = \alpha'$. We write $\mathsf{tr}_P\, e\, u : P\, \alpha'$ for transport of $u : P\, \alpha$ along $e : \alpha = \alpha'$. Sometimes we omit the parameters in subscript. We also have coercion $\mathsf{coe}\, e\, \alpha : T'$ whenever $e : T = T'$ and the inverse $e^{-1} : \alpha' = \alpha$ for $e : \alpha = \alpha'$.

In this paper we use the notation of extensional type theory, that is, after proving an equality $p : \alpha = \alpha'$, in later proofs depending on this equality we treat it as definitional, thus $\alpha \equiv \alpha'$ and $p \equiv \mathsf{refl}$. This makes the notation much lighter, as transports disappear: $\mathsf{tr}\, p\, u \equiv u$ for any proof p. Although we use extensional notation, this can be translated to intensional type theory extended with functional extensionality and UIP, as we know from [Hofmann 1995a; Oury 2005; Winterhalter et al. 2018]. In our Agda formalisation, we use these axioms along with rewrite rules [Cockx et al. 2014]. A rewrite rule allows to turn a previously proven (or postulated) equality into a definitional one. This provides an alternative way to add inductive types to Agda: one can postulate the constructors, the eliminator and the computation rules and make the computation rules definitional by marking them as rewrite rules.

We denote the functional extensionality axiom by $\mathsf{funext} : (\alpha : A) \to f\, \alpha = g\, \alpha \to f = g$. We also use the term UIP which has type $e = e'$ whenever $e, e' : \alpha = \beta$. We write equational reasoning proofs by writing the proofs of equalities above the $=$ symbol, e.g. $u \overset{e}{=} u'$ when $e : t = t'$ and $u \equiv f\, t, u' \equiv f\, t'$.

[4]This is demonstrated in [Kaposi and Kovács 2018], by a Haskell program which takes a signature as input, type checks it, then outputs the corresponding induction methods and eliminators as an Agda file.

We also assume the existence of one QIIT, namely the syntax of the type theory of signatures (Section 3). In our formalisation we postulate its constructors and dependent elimination principle, adding the computation rules as rewrite rules. This QIIT has equality constructors for the substitution calculus (see the next section). In this paper, we treat these equalities as definitional (we don't write transports along them). Analogously, we add rewrite rules for them in the formalisation.

The Agda formalisation has been checked using Agda 2.5.4. It is available at https://bitbucket.org/akaposi/finitaryqiit, together with a technical appendix. The Agda formalisation covers sections 3.1, 3.3, 4, the definition of homomorphisms from 5, and also 6, 7.1, 7.2, 7.3, and part of 7.4. The formalisation also includes additional documentation concerning technicalities.

3 THE TYPE THEORY OF SIGNATURES

In this section we define the syntax for a domain-specific type theory. A context in this type theory is a signature for a QIIT. We use intrinsic syntax (that is, we only have well-typed terms), de Bruijn variables and explicit substitutions (substitution is a constructor instead of an operation). The syntax is given by a QIIT and conversion rules are given by equality constructors in the style of [Altenkirch and Kaposi 2016]. Our definition of the syntax can be seen as an unfolding of the initial category with families (CwF) [Dybjer 1996] with certain type formers.

3.1 The Syntax

We have a QIIT with four sorts. Types are indexed over contexts: $\mathsf{Ty}\,\Gamma$ denotes the well-formed types which have free variables in Γ. An element of $\mathsf{Sub}\,\Gamma\,\Delta$ can be viewed as a list of terms: it contains one term for each type in Δ and each of these terms has free variables in Γ. An element of $\mathsf{Tm}\,\Gamma\,A$ is a term of type A with free variables in Γ.

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

The following constructors of the above sorts give the *substitution calculus* part of the syntax.

$\cdot$	$: \mathsf{Con}$	empty context
$- \triangleright -$	$: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$	context extension
$-[-]$	$: \mathsf{Ty}\,\Delta \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma$	substitution of types
id	$: \mathsf{Sub}\,\Gamma\,\Gamma$	identity substitution
$- \circ -$	$: \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Gamma\,\Theta \to \mathsf{Sub}\,\Gamma\,\Delta$	composition
ϵ	$: \mathsf{Sub}\,\Gamma\,\cdot$	empty substitution
$-,-$	$: (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma]) \to \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)$	substitution extension
π_1	$: \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A) \to \mathsf{Sub}\,\Gamma\,\Delta$	first projection
π_2	$: (\sigma : \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)) \to \mathsf{Tm}\,\Gamma\,(A[\pi_1\,\sigma])$	second projection
$-[-]$	$: \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma])$	substitution of terms
$[\mathsf{id}]$	$: A[\mathsf{id}] = A$	
$[\circ]$	$: A[\sigma \circ \delta] = A[\sigma][\delta]$	
ass	$: (\sigma \circ \delta) \circ \nu = \sigma \circ (\delta \circ \nu)$	
idl	$: \mathsf{id} \circ \sigma = \sigma$	

$$\begin{array}{ll}
\mathsf{idr} & : \sigma \circ \mathsf{id} = \sigma \\
\cdot\eta & : \{\sigma : \mathsf{Sub}\,\Gamma\,\cdot\} \to \sigma = \epsilon \\
\triangleright\beta_1 & : \pi_1\,(\sigma, t) = \sigma \\
\triangleright\beta_2 & : \pi_2\,(\sigma, t) = t \\
\triangleright\eta & : (\pi_1\,\sigma, \pi_2\,\sigma) = \sigma \\
,\circ & : (\sigma, t) \circ \delta = (\sigma \circ \delta, t[\delta])
\end{array}$$

$- \triangleright -$ and $-, -$ are left-associative binary operators. Note that $\triangleright\beta_2$ is only well-typed because of a previous equality constructor: the left hand side has type $\mathsf{Tm}\,\Gamma\,(A[\pi_1\,(\sigma, t)])$ and the right hand side has type $\mathsf{Tm}\,\Gamma\,(A[\sigma])$, but these types are equal by $\triangleright\beta_1$. The case of $,\circ$ is similar, here $t[\delta]$ needs to have type $\mathsf{Tm}\,\Gamma\,(A[\sigma \circ \delta])$, however, it has type $\mathsf{Tm}\,\Gamma\,(A[\sigma][\delta])$, but these are equal by $[\circ]$.

Using the terminology of CwFs, the substitution calculus can be summarized as follows: we have a category (Con, Sub, id, $- \circ -$, ass, idl, idr) with a terminal object ($\cdot$, ϵ, $\cdot\eta$), a contravariant functor from this category to the category of families of sets (action on objects given by Ty and Tm, action on morphisms given by the $-[-]$ operators, the functor laws are [id] and $[\circ]$; the functor laws for term substitution are derivable, see below) and a natural isomorphism between $(\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \times \mathsf{Tm}\,\Gamma\,(A[\sigma])$ and $\mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)$, called comprehension ($- \triangleright -$, $-, -$, π_1, π_2, $\triangleright\beta_1$, $\triangleright\beta_2$, $\triangleright\eta$, $,\circ$). It is shown in [Kaposi 2017, p. 63] that this formulation of CwF is equivalent to the original one [Dybjer 1996].

Usual syntactic constructions such as weakenings, variables (de Bruijn indices) and liftings of substitutions can be recovered by the definitions below.

$$\begin{array}{lll}
\mathsf{wk} & : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma & :\equiv \pi_1\,\mathsf{id} \\
\mathsf{vz} & : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{wk}]) & :\equiv \pi_2\,\mathsf{id} \\
\mathsf{vs}\,(x : \mathsf{Tm}\,\Gamma\,A) & : \mathsf{Tm}\,(\Gamma \triangleright B)\,(A[\mathsf{wk}]) & :\equiv x[\mathsf{wk}] \\
\langle t : \mathsf{Tm}\,\Gamma\,A \rangle & : \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A) & :\equiv (\mathsf{id}, t) \\
(\sigma : \mathsf{Sub}\,\Gamma\,\Delta)^{\uparrow} & : \mathsf{Sub}\,(\Gamma \triangleright A[\sigma])\,(\Delta \triangleright A) & :\equiv (\sigma \circ \mathsf{wk}, \mathsf{vz})
\end{array}$$

We will denote variables by natural numbers, e.g. $3 :\equiv \mathsf{vs}\,(\mathsf{vs}\,(\mathsf{vs}\,\mathsf{vz}))$.

As an example of using the substitution calculus, we prove the functor laws for terms using equational reasoning. We use the same names for these laws as for type subsitution: [id] and $[\circ]$.

$$\begin{array}{lll}
[\mathsf{id}] : & t[\mathsf{id}] & \\
& & \triangleright\beta_2{}^{-1} \\
& = \pi_2\,(\mathsf{id}, t[\mathsf{id}]) & \\
& & \mathsf{idl}^{-1} \\
& = \pi_2\,(\mathsf{id} \circ \mathsf{id}, t[\mathsf{id}]) & \\
& & ,\circ^{-1} \\
& = \pi_2\,((\mathsf{id}, t) \circ \mathsf{id}) & \\
& & \mathsf{idr} \\
& = \pi_2\,(\mathsf{id}, t) & \\
& & \triangleright\beta_2 \\
& = t &
\end{array}$$

$$\begin{aligned}
[\circ] \ :\ & t[\sigma \circ \delta] \\
&& \triangleright\beta_2{}^{-1} \\
=\ & \pi_2\,(\sigma \circ \delta, t[\sigma \circ \delta]) \\
&& \mathsf{idl}^{-1} \\
=\ & \pi_2\,(\mathsf{id} \circ (\sigma \circ \delta), t[\sigma \circ \delta]) \\
&& ,\circ^{-1} \\
=\ & \pi_2\,((\mathsf{id}, t) \circ (\sigma \circ \delta)) \\
&& \mathsf{ass}^{-1} \\
=\ & \pi_2\,(((\mathsf{id}, t) \circ \sigma) \circ \delta) \\
&& ,\circ \\
=\ & \pi_2\,((\mathsf{id} \circ \sigma, t[\sigma]) \circ \delta) \\
&& ,\circ \\
=\ & \pi_2\,((\mathsf{id} \circ \sigma) \circ \delta, t[\sigma][\delta]) \\
&& \triangleright\beta_2 \\
=\ & t[\sigma][\delta]
\end{aligned}$$

In addition to the substitution calculus, we have an empty *universe* given by the following constructors. This allows us to add sorts to a signature.

$$\begin{aligned}
&\mathsf{U} && : \mathsf{Ty}\,\Gamma \\
&\mathsf{El} && : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma \\
&\mathsf{U}[] && : \mathsf{U}[\sigma] = \mathsf{U} \\
&\mathsf{El}[] && : (\mathsf{El}\,a)[\sigma] = \mathsf{El}\,(a[\sigma])
\end{aligned}$$

We also have a dependent *function space with small domain.* This can be used to add inductive arguments to operators in a signature, for example the argument of sucessor for natural numbers. We have constructors for Π, a categorical application rule and substitution laws.

$$\begin{aligned}
&\Pi && : (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Ty}\,(\Gamma \triangleright \mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma \\
&\mathsf{app} && : \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B) \to \mathsf{Tm}\,(\Gamma \triangleright \mathsf{El}\,a)\,B \\
&\Pi[] && : (\Pi\,a\,B)[\sigma] = \Pi\,(a[\sigma])\,(B[\sigma^\uparrow]) \\
&\mathsf{app}[] && : (\mathsf{app}\,t)[\sigma^\uparrow] = \mathsf{app}\,(t[\sigma])
\end{aligned}$$

By restricting the domain to be small, we can only write strictly positive operators (we cannot write $\Pi\,(\Pi\,a\,B)\,C$ because the first argument of Π needs to be small but $(\Pi\,a\,B)$ itself is large). We omit lambda abstraction, because there is no essential use for it in signatures, and it follows that this function type only has neutral elements. We define the non-dependent function space by the following abbreviation.

$$(a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \Rightarrow (B : \mathsf{Ty}\,\Gamma) : \mathsf{Ty}\,\Gamma :\equiv \Pi\,a\,(B[\mathsf{wk}])$$

$- @ -$ is left-associative, $- \Rightarrow -$ is right-associative. The usual application can be defined as below.

$$(t : \mathsf{Tm}\,\Gamma\,(\Pi\,a\,B))\,@\,(u : \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a)) : \mathsf{Tm}\,\Gamma\,(B[\langle u \rangle]) :\equiv (\mathsf{app}\,t)[\langle u \rangle]$$

We have an *identity type* for elements of a small type, with equality reflection. The identity type itself is large. This can be used to add equalities to a signature.

$$\begin{aligned}
&\mathsf{Id} &&: (a : \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,a) \to \mathsf{Ty}\,\Gamma \\
&\mathsf{reflect} &&: \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,a\,t\,u) \to t = u \\
&\mathsf{Id}[] &&: (\mathsf{Id}\,a\,t\,u)[\sigma] = \mathsf{Id}\,(a[\sigma])\,(t[\sigma])\,(u[\sigma])
\end{aligned}$$

Transport can be derived using reflection and the metatheoretic transport tr.

$$\begin{aligned}
&\mathsf{transp}\,(P : \mathsf{Ty}\,(\Gamma \rhd \mathsf{El}\,a))(e : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}\,a\,t\,u))\,(w : \mathsf{Tm}\,\Gamma\,(P[\langle t\rangle])) : \mathsf{Tm}\,\Gamma\,(P[\langle u\rangle]) \\
&:\equiv \mathsf{tr}_{\mathsf{Tm}\,\Gamma\,(P[\langle - \rangle])}\,(\mathsf{reflect}\,e)\,w
\end{aligned}$$

We have a *function space with metatheoretic domain.* This can be used to add non-inductive parameters to a sort or an operator in a signature.

$$\begin{aligned}
&\hat{\Pi} &&: (T : \mathsf{Set}) \to (T \to \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,\Gamma \\
&- \mathbin{\hat{@}} - &&: \mathsf{Tm}\,\Gamma\,(\hat{\Pi}\,T\,B) \to (\alpha : T) \to \mathsf{Tm}\,\Gamma\,(B\,\alpha) \\
&\hat{\Pi}[] &&: (\hat{\Pi}\,T\,B)[\sigma] = \hat{\Pi}\,T\,(\lambda\alpha.(B\,\alpha)[\sigma]) \\
&\hat{@}[] &&: (t \mathbin{\hat{@}} \alpha)[\sigma] = (t[\sigma]) \mathbin{\hat{@}} \alpha
\end{aligned}$$

We abbreviate $\hat{\Pi}\,T\,(\lambda\alpha.B)$ where B has type $\mathsf{Ty}\,\Gamma$ as $T \hat{\Rightarrow} B$.

3.2 Example Signatures

The signature for natural numbers (Section 1.1) can be given by the following context on the left hand side. The right hand side is the same context in an informal notation using variable names.

$$\cdot \rhd \mathsf{U} \rhd \mathsf{El}\,0 \rhd (1 \Rightarrow \mathsf{El}\,1) \qquad\qquad \cdot \rhd \mathit{Nat} : \mathsf{U} \rhd \mathit{zero} : \mathsf{El}\,\mathit{Nat} \rhd \mathit{suc} : \mathit{Nat} \Rightarrow \mathsf{El}\,\mathit{Nat}$$

We start with the empty context $\cdot$, use context extension $\rhd$ to add a sort by U. Then we extend the context with an operator which returns in El 0, that is, in the sort just declared before. The last operator uses the function space with small domain $\Rightarrow$. The domain is the sort given earlier (now it is referred to by index 1) and the codomain is the same (the codomain needs to be large, hence the use of El).

The signature for integers is the following (Section 1.2).

$$\begin{aligned}
&\cdot \rhd \mathsf{U} && \cdot \rhd \mathit{Int} \;: \mathsf{U} \\
&\rhd \mathbb{N} \hat{\Rightarrow} \mathbb{N} \hat{\Rightarrow} \mathsf{El}\,0 && \rhd \mathit{pair} : \mathbb{N} \hat{\Rightarrow} \mathbb{N} \hat{\Rightarrow} \mathsf{El}\,\mathit{Int} \\
&\rhd \hat{\Pi}\,\mathbb{N}\,\lambda a.\hat{\Pi}\,\mathbb{N}\,\lambda b.\hat{\Pi}\,\mathbb{N}\,\lambda c.\hat{\Pi}\,\mathbb{N}\,\lambda d. && \rhd \mathit{eq} \;: \hat{\Pi}\,\mathbb{N}\,\lambda a.\hat{\Pi}\,\mathbb{N}\,\lambda b.\hat{\Pi}\,\mathbb{N}\,\lambda c.\hat{\Pi}\,\mathbb{N}\,\lambda d. \\
&\qquad a + d = c + d \hat{\Rightarrow} && \qquad a + d = b + c \hat{\Rightarrow} \\
&\qquad \mathit{Id}\,1\,(0 \mathbin{\hat{@}} a \mathbin{\hat{@}} b)\,(0 \mathbin{\hat{@}} c \mathbin{\hat{@}} d) && \qquad \mathsf{Id}\,\mathit{Int}\,(\mathit{pair} \mathbin{\hat{@}} a \mathbin{\hat{@}} b)\,(\mathit{pair} \mathbin{\hat{@}} c \mathbin{\hat{@}} d)
\end{aligned}$$

In this example we use the function space with metatheoretic domain $\hat{\Rightarrow}$ to express arguments in the metatheoretic set of natural numbers $\mathbb{N}$. Note the usage of metatheoretic λs for binding the parameters of the *eq* constructor.

One might wonder why we need the function space $\hat{\Pi}$ when we could just define natural numbers and integers as one context ($\mathit{Nat} : \mathsf{U}$, $\mathit{zero} : \mathsf{El}\,\mathit{Nat}$, $\mathit{suc} : \mathit{Nat} \Rightarrow \mathsf{El}\,\mathit{Nat}$, $\mathit{Int} : \mathsf{U}$, $\mathit{pair} : \mathit{Nat} \Rightarrow \mathit{Nat} \Rightarrow \mathsf{El}\,\mathit{Int}, \ldots$). However, we would not be able to extend this signature with the *eq* constructor, as it uses + on natural numbers, and the + operation is defined using the recursor for natural numbers which is not available at this stage.

The signature for the Con-Ty example is given below (see beginning of Section 1 and Section 1.3).

$$
\begin{aligned}
&\cdot \rhd \mathsf{U} \\
&\quad \rhd 0 \Rightarrow \mathsf{U} \\
&\quad \rhd \mathsf{El}\, 1 \\
&\quad \rhd \Pi\, 2\, (2 \mathbin{@} 0 \Rightarrow \mathsf{El}\, 3) \\
&\quad \rhd \Pi\, 3\, (3 \mathbin{@} 0) \\
&\quad \rhd \Pi\, 4\, (\Pi\, (4 \mathbin{@} 0) \\
&\qquad\qquad (5 \mathbin{@} (3 \mathbin{@} 1 \mathbin{@} 0) \Rightarrow \mathsf{El}\, (5 \mathbin{@} 1))) \\
&\quad \rhd \Pi\, 5\, (\Pi\, (5 \mathbin{@} 0) \\
&\qquad\qquad (\Pi\, (6 \mathbin{@} (4 \mathbin{@} 1 \mathbin{@} 0)) \\
&\qquad\qquad\quad (\mathsf{Id}\, 7\, (4 \mathbin{@} (4 \mathbin{@} 2 \mathbin{@} 1) \mathbin{@} 0) \\
&\qquad\qquad\qquad (4 \mathbin{@} 2 \mathbin{@} (3 \mathbin{@} 2 \mathbin{@} 1 \mathbin{@} 0))))))
\end{aligned}
\qquad
\begin{aligned}
&\cdot \rhd \mathit{Con} : \mathsf{U} \\
&\quad \rhd \mathit{Ty} \;\; : \mathit{Con} \Rightarrow \mathsf{U} \\
&\quad \rhd \mathit{nil} \;\; : \mathsf{El}\, \mathit{Con} \\
&\quad \rhd \mathit{ext} \;\; : \Pi(\Gamma : \mathit{Con}).\mathit{Ty} \mathbin{@} \Gamma \Rightarrow \mathsf{El}\, \mathit{Con} \\
&\quad \rhd U \;\; : \Pi(\Gamma : \mathit{Con}).\mathsf{El}\, (\mathit{Ty} \mathbin{@} \Gamma) \\
&\quad \rhd \Sigma \;\; : \Pi(\Gamma : \mathit{Con}).\Pi(A : \mathit{Ty} \mathbin{@} \Gamma). \\
&\qquad\qquad \mathit{Ty} \mathbin{@} (\mathit{ext} \mathbin{@} \Gamma \mathbin{@} A) \Rightarrow \mathsf{El}\, (\mathit{Ty} \mathbin{@} \Gamma) \\
&\quad \rhd \mathit{eq} \;\; : \Pi(\Gamma : \mathit{Con}).\Pi(A : \mathit{Ty} \mathbin{@} \Gamma). \\
&\qquad\qquad \Pi(B : \mathit{Ty} \mathbin{@} (\mathit{ext} \mathbin{@} \Gamma \mathbin{@} A). \\
&\qquad\qquad \mathsf{Id}\, \mathit{Con}\, (\mathit{ext} \mathbin{@} (\mathit{ext} \mathbin{@} \Gamma \mathbin{@} A) \mathbin{@} B) \\
&\qquad\qquad\qquad (\mathit{ext} \mathbin{@} \Gamma \mathbin{@} (\Sigma \mathbin{@} \Gamma \mathbin{@} A \mathbin{@} B))
\end{aligned}
$$

3.3 Defining Functions from the Syntax

In the following sections, we will define functions by induction on the syntax of the theory of signatures. This amounts to saying what Con, Ty, Sub and Tm are mapped to and providing cases for all of the 35 constructors from $\cdot$ to $\hat{@}[]$. Examples of full definitions of some of these functions are given in Appendix A, the simplest one being the standard interpretation $-^{\mathsf{A}}$.

We refer to [Altenkirch and Kaposi 2016] for a full definition of the elimination principle of a similar type theory. Alternatively, the elimination principle for the theory of signatures can be mechanically generated by the methods described in [Kaposi and Kovács 2018].

4 ALGEBRAS AND THE INITIAL ALGEBRA

In this section we define the notion of algebras for signatures and show that an algebra exists for every signature. We will prove the initiality of these algebras in Section 7.4.

We first compute notions of algebras by induction on the syntax of the theory of signatures. We denote the operation for this as $-^{\mathsf{A}}$. For types, substitution and terms, $-^{\mathsf{A}}$ works as follows.

$$
\begin{aligned}
&(\Gamma : \mathsf{Con})^{\mathsf{A}} && : \mathsf{Set} \\
&(A : \mathsf{Ty}\,\Gamma)^{\mathsf{A}} && : \Gamma^{\mathsf{A}} \to \mathsf{Set} \\
&(\sigma : \mathsf{Sub}\,\Gamma\,\Delta)^{\mathsf{A}} && : \Gamma^{\mathsf{A}} \to \Delta^{\mathsf{A}} \\
&(t : \mathsf{Tm}\,\Gamma\,A)^{\mathsf{A}} && : (\gamma : \Gamma^{\mathsf{A}}) \to A^{\mathsf{A}}\,\gamma
\end{aligned}
$$

The $-^{\mathsf{A}}$ operation is the interpretation into the standard model (set-theoretic model, metacircular model). Object-theoretic constructs are mapped to their metatheoretic counterparts: contexts become sets, types become famillies of sets, terms become dependent functions.

The above four lines are the specification of the operation $-^{\mathsf{A}}$. Its definition amounts to describing what it does on all the constructors of the theory of signatures. We start by saying that $-^{\mathsf{A}}$ on the empty context returns the unit type $\cdot^{\mathsf{A}} :\equiv \top$, context extension is mapped to Σ, that is $(\Gamma \rhd A)^{\mathsf{A}} :\equiv (\gamma : \Gamma^{\mathsf{A}}) \times A^{\mathsf{A}}\,\gamma$. π_1 and π_2 are interpreted as first and second projections, respectively, so a variable projects out the corresponding component, e.g. $2^{\mathsf{A}}\,(\gamma, \alpha, \alpha', \alpha'') \equiv \alpha$. U is interpreted as Set, that is, $\mathsf{U}^{\mathsf{A}}\,\gamma :\equiv \mathsf{Set}$, a type coming from a code uses the interpretation of the code: $(\mathsf{El}\,a)^{\mathsf{A}}\,\gamma :\equiv a^{\mathsf{A}}\,\gamma$

which has the right type as $\mathsf{U}^{\mathsf{A}}\,\gamma \equiv \mathsf{Set}$. Both function spaces are mapped to metatheoretic function space: $(\Pi\, a\, B)^{\mathsf{A}}\,\gamma :\equiv (\alpha : a^{\mathsf{A}}\,\gamma) \to B^{\mathsf{A}}\,(\gamma, \alpha)$ and $(\hat{\Pi}\, T\, B)^{\mathsf{A}}\,\gamma :\equiv (\alpha : T) \to (B\,\alpha)^{\mathsf{A}}\,\gamma$. The identity type is mapped to the metatheoretic identity type: $(\mathsf{Id}\, a\, t\, u)^{\mathsf{A}}\,\gamma :\equiv (t^{\mathsf{A}}\,\gamma = u^{\mathsf{A}}\,\gamma)$. The standard interpretation justifies equality reflection when the metatheory supports functional extensionality: we need to provide $(\mathsf{reflect}\, e)^{\mathsf{A}} : t^{\mathsf{A}} = u^{\mathsf{A}}$, and we have $e^{\mathsf{A}} : (\gamma : \Gamma^{\mathsf{A}}) \to t^{\mathsf{A}}\,\gamma = u^{\mathsf{A}}\,\gamma$. Hence $(\mathsf{reflect}\, e)^{\mathsf{A}}$ is just given by funext. All the other equality constructors are interpreted by refl. The full definition for every constructor of the theory of signatures is given in Appendix A.

In the example of natural numbers the carrier of the initial algebra is given by $\mathsf{Tm}\,\Delta\,(\mathsf{El}\, \mathit{Nat})$ where Δ is the signature for natural numbers. We could naively attempt to define the initial algebra $\mathsf{con}_\Delta : \Delta^{\mathsf{A}}$ by induction on Δ, however, this is not possible, because we need access to the full Δ when writing down the carrier $\mathsf{Tm}\,\Delta\,(\mathsf{El}\, \mathit{Nat})$. Hence, we need to first fix a signature $\Omega : \mathsf{Con}$ and define an operation $-^{\mathsf{C}_\Omega}$, which for a context Γ takes a substitution from Ω to Γ and returns an element of Γ^{A}. Then, we can recover the initial Ω-algebra as $\mathsf{con}_\Omega : \Omega^{\mathsf{A}} :\equiv \Omega^{\mathsf{C}_\Omega}\,\mathsf{id}$ where id is the identity substitution.

The motives of $-^{\mathsf{C}_\Omega}$ are given as follows (we omit the Ω subscripts from now on).

$$\begin{aligned}
&(\Gamma : \mathsf{Con})^{\mathsf{C}} && : \mathsf{Sub}\,\Omega\,\Gamma \to \Gamma^{\mathsf{A}}\\
&(A : \mathsf{Ty}\,\Gamma)^{\mathsf{C}} && : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \mathsf{Tm}\,\Omega\,(A[\nu]) \to A^{\mathsf{A}}\,(\Gamma^{\mathsf{C}}\,\nu)\\
&(\sigma : \mathsf{Sub}\,\Gamma\,\Delta)^{\mathsf{C}} && : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^{\mathsf{C}}\,(\sigma \circ \nu) = \sigma^{\mathsf{A}}\,(\Gamma^{\mathsf{C}}\,\nu)\\
&(t : \mathsf{Tm}\,\Gamma\,A)^{\mathsf{C}} && : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^{\mathsf{C}}\,\nu\,(t[\nu]) = t^{\mathsf{A}}\,(\Gamma^{\mathsf{C}}\,\nu)
\end{aligned}$$

Once provided with a substitution into the context, contexts are interpreted as algebras. The interpretation of types is determined by the need for context extension: $(\Gamma \rhd A)^{\mathsf{C}}\,(\nu, t)$ has type $(\Gamma \rhd A)^{\mathsf{A}} \equiv (\gamma : \Gamma^{\mathsf{A}}) \times A^{\mathsf{A}}\,\gamma$. We can provide the γ part by $\Gamma^{\mathsf{C}}\,\nu$, so A^{C} has to provide something of type $A^{\mathsf{A}}\,(\Gamma^{\mathsf{C}}\,\nu)$. Similarly, the interpretation of substitutions and terms are determined by the requirements of type substitution and substitution extension.

The universe is interpreted by $\mathsf{U}^{\mathsf{C}}\,\nu\, a :\equiv \mathsf{Tm}\,\Omega\,(\mathsf{El}\, a)$ which results in the sorts being modelled by terms in the initial algebra. For El, we coerce along the equality $a^{\mathsf{C}}\,\nu : \mathsf{Tm}\,\Omega\,(\mathsf{El}\, a) = a^{\mathsf{A}}\,(\Gamma^{\mathsf{C}}\,\nu)$, and thus $(\mathsf{El}\, a)^{\mathsf{C}}\,\nu\, t :\equiv \mathsf{coe}\,(a^{\mathsf{C}}\,\nu)\, t$. For $\Pi\, a\, B$ types, we define a function which coerces the input along a^{C} and uses B^{C} to produce the result in the initial algebra:

$$(\Pi\, a\, B)^{\mathsf{C}}\,\nu\, t :\equiv \lambda\alpha.B^{\mathsf{C}}\,\big(\nu, \mathsf{coe}\,(a^{\mathsf{C}}\,\nu^{-1})\,\alpha\big)\,\big(t \mathbin{@} \mathsf{coe}\,(a^{\mathsf{C}}\,\nu^{-1})\,\alpha\big)$$

Equalities of terms and substitutions are trivial in the $-^{\mathsf{C}}$ interpretation, as they are equalities between equalities and these can be just given by UIP.

We refer the interested reader to Appendix A for the full definition of this and later operations.

In the following sections, we will make use of the following two special cases of $-^{\mathsf{C}}$ for terms. First, we know that the set of terms of a small type is equal to the standard interpretation of the type at the initial algebra. That is, given a term $a : \mathsf{Tm}\,\Omega\,\mathsf{U}$, we have

$$a^{\mathsf{C}}\,\mathsf{id} : \mathsf{Tm}\,\Omega\,(\mathsf{El}\, a) = a^{\mathsf{A}}\,\mathsf{con}_\Omega.$$

Second, we know that every term of a small type is equal to its standard interpretation at the initial algebra. That is, given a $t : \mathsf{Tm}\,\Omega\,(\mathsf{El}\, a)$, we have

$$t^{\mathsf{C}}\,\mathsf{id} : t = t^{\mathsf{A}}\,\mathsf{con}_\Omega.$$

This equality comes from $(\mathsf{El}\, a)^{\mathsf{C}}\,\mathsf{id}\, t = t^{\mathsf{C}}\,(\Omega^{\mathsf{C}}\,\mathsf{id})$ which computes to $\mathsf{coe}\,(a^{\mathsf{C}}\,\mathsf{id})\, t = t^{\mathsf{A}}\,\mathsf{con}_\Omega$, and forgetting the coercion we get the above equality.

5 HOMOMORPHISMS AND THE RECURSOR

Given two Γ-algebras $\gamma, \gamma' : \Gamma^{\mathsf{A}}$, we define the notion of homomorphism between them using a variant of the logical relation interpretation for dependent types [Atkey et al. 2014; Bernardy et al. 2012]. Contexts become binary relations, types become heterogeneous binary relations indexed over a relation for a context, and substitutions and terms are interpreted as the fundamental theorems for the logical relation.

$$\begin{aligned}
&(\Gamma : \mathsf{Con})^{\mathsf{M}} &&: \Gamma^{\mathsf{A}} \to \Gamma^{\mathsf{A}} \to \mathsf{Set}\\
&(A : \mathsf{Ty}\,\Gamma)^{\mathsf{M}} &&: \Gamma^{\mathsf{M}}\,\gamma^0\,\gamma^1 \to A^{\mathsf{A}}\,\gamma^0 \to A^{\mathsf{A}}\,\gamma^1 \to \mathsf{Set}\\
&(\sigma : \mathsf{Sub}\,\Gamma\,\Delta)^{\mathsf{M}} &&: \Gamma^{\mathsf{M}}\,\gamma^0\,\gamma^1 \to \Delta^{\mathsf{M}}\,(\sigma^{\mathsf{A}}\,\gamma^0)\,(\sigma^{\mathsf{A}}\,\gamma^1)\\
&(t : \mathsf{Tm}\,\Gamma\,A)^{\mathsf{M}} &&: (\gamma^M : \Gamma^{\mathsf{M}}\,\gamma^0\,\gamma^1) \to A^{\mathsf{M}}\,\gamma^M\,(t^{\mathsf{A}}\,\gamma^0)\,(t^{\mathsf{A}}\,\gamma^1)
\end{aligned}$$

However, the usual logical relation interpretation does not produce homomorphisms. The motivation of Reynolds [Reynolds 1983] to replace homomorphisms with logical relations was that homomorphisms do not work for higher order functions. In our case there are no higher-order functions because the function space Π is strictly positive. Thus, we are able to interpret the universe by function space instead of relation space. Similarly, the relation for El is the graph of the corresponding function. Below we list the differences: the left hand side ($-^{\mathsf{M}'}$) is the usual logical relation interpretation, the right hand side is the one we use.

$$\begin{aligned}
&\mathsf{U}^{\mathsf{M}'}\,\gamma^{M'}\,T^0\,T^1 &&:\equiv T^0 \to T^1 \to \mathsf{U} &\qquad& \mathsf{U}^{\mathsf{M}}\,\gamma^M\,T^0\,T^1 &&:\equiv T^0 \to T^1\\
&(\mathsf{El}\,a)^{\mathsf{M}'}\,\gamma^{M'}\,\alpha^0\,\alpha^1 &&:\equiv a^{\mathsf{M}'}\,\gamma^{M'}\,\alpha^0\,\alpha^1 && (\mathsf{El}\,a)^{\mathsf{M}}\,\gamma^M\,\alpha^0\,\alpha^1 &&:\equiv a^{\mathsf{M}}\,\gamma^M\,\alpha^0 = \alpha^1\\
&(\Pi\,a\,B)^{\mathsf{M}'}\,\gamma^{M'}\,f^0\,f^1 &&:\equiv (\alpha^{M'} : a^{\mathsf{M}'}\,\gamma^{M'}\,\alpha^0\,\alpha^1) \to && (\Pi\,a\,B)^{\mathsf{M}}\,\gamma^M\,f^0\,f^1 &&:\equiv (\alpha^0 : a^{\mathsf{A}}\,\gamma^0) \to\\
& && \quad B^{\mathsf{M}'}\,(\gamma^{M'}, \alpha^{M'})\,(f^0\,\alpha^0) && && \quad B^{\mathsf{M}}\,(\gamma^M, \mathsf{refl})\,(f^0\,\alpha^0)\\
& && \qquad (f^1\,\alpha^1) && && \qquad (f^1\,(a^{\mathsf{M}}\,\gamma^M\,\alpha^0))\\
&(t \mathbin{@} u)^{\mathsf{M}'}\,\gamma^{M'} &&:\equiv t^{\mathsf{M}'}\,(u^{\mathsf{M}'}\,\gamma^{M'}) && (t \mathbin{@} u)^{\mathsf{M}}\,\gamma^M &&:\equiv \mathsf{J}\,(t^{\mathsf{M}}\,\gamma^M\,(u^{\mathsf{A}}\,\gamma))\,(u^{\mathsf{M}}\,\gamma^M)
\end{aligned}$$

For Π types, we could use the usual interpretation, but we have a better choice. E.g. for the successor constructor of the natural numbers, the original formulation would give the condition $(n^0 : Nat^0)(n^1 : Nat^1)(n^M : Nat^M\,n^0 = n^1) \to Nat^M\,(suc^0\,n^0) = suc^1\,n^1$. The right hand side variant strictifies this and results in $(n^0 : Nat^0) \to Nat^M\,(suc^1\,n^0) = suc^1\,(Nat^M\,n^0)$. The price we have to pay is that interpreting application requires usage of J.

$-^{\mathsf{M}}$ is constant $\top$ on the identity type; homomorphisms do not state any conditions between identity proofs in different algebras because of UIP.

For the recursor, we fix a signature Ω and an algebra $\omega : \Omega^{\mathsf{A}}$. We write con for $\mathsf{con}_\Omega \equiv \Omega^{\mathsf{C}}\,\mathsf{id}$. The operation $-^{\mathsf{R}_\omega}$ specified as follows.

$$\begin{aligned}
&(\Gamma : \mathsf{Con})^{\mathsf{R}} &&: (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^{\mathsf{M}}\,(\nu^{\mathsf{A}}\,\mathsf{con})\,(\nu^{\mathsf{A}}\,\omega)\\
&(A : \mathsf{Ty}\,\Gamma)^{\mathsf{R}} &&: (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(A[\nu])) \to A^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)\,(t^{\mathsf{A}}\,\mathsf{con})\,(t^{\mathsf{A}}\,\omega)\\
&(\sigma : \mathsf{Sub}\,\Gamma\,\Delta)^{\mathsf{R}} &&: (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^{\mathsf{R}}\,(\sigma \circ \nu) = \sigma^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)\\
&(t : \mathsf{Tm}\,\Gamma\,A)^{\mathsf{R}} &&: (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^{\mathsf{R}}\,\nu\,(t[\nu]) = t^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)
\end{aligned}$$

For a context Γ, we get a homomorphism from the constructor (initial algebra) to the given algebra ω, but both of them have to be transported by the standard interpretation of ν from Ω^{A} to Γ^{A}. For types, we get a heterogeneous homomorphism over the interpretation of the context. For substitutions and terms, we get naturality conditions expressing that $-^{\mathsf{M}}$ and $-^{\mathsf{R}}$ commute.

U^{R} is given using the standard interpretation $-^{\mathsf{A}}$. Given $\nu : \mathsf{Sub}\,\Omega\,\Gamma$ and $a : \mathsf{Tm}\,\Omega\,\mathsf{U}$ we need something of type $(\alpha : a^{\mathsf{A}}\,\mathsf{con}) \to a^{\mathsf{A}}\,\omega$. We know by $a^{\mathsf{C}}\,\mathsf{id}$ that $\mathsf{Tm}\,\Omega\,(\mathsf{El}\,a) = a^{\mathsf{A}}\mathsf{con}$, so coercing α along this and applying the standard interpretation we get $(\mathsf{coe}\,(a^{\mathsf{C}}\,\mathsf{id}^{-1})\,\alpha)^{\mathsf{A}}\,\omega : a^{\mathsf{A}}\,\omega$.

For El, we need $(\mathsf{El}\,a)^{\mathsf{R}}\,\nu\,t : a^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)\,(t^{\mathsf{A}}\,\mathsf{con}) = t^{\mathsf{A}}\,\omega$. We use the following equational reasoning to prove this.

$$a^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)\,(t^{\mathsf{A}}\,\mathsf{con}) \overset{t^{\mathsf{C}}\,\mathsf{id}^{-1}}{=} a^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)\,t \overset{a^{\mathsf{R}}\,\nu^{-1}}{=} t^{\mathsf{A}}\,\omega$$

For the function space, $(\Pi\,a\,B)^{\mathsf{R}}$ is defined using a^{R} and B^{R} as follows. $(\Pi\,a\,B)^{\mathsf{R}}\,\nu\,t$ needs to have type $(\alpha : a^{\mathsf{A}}\,(\nu^{\mathsf{A}}\,\mathsf{con})) \to B^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu, \mathsf{refl})\,(t^{\mathsf{A}}\,\mathsf{con}\,\alpha)\,(t^{\mathsf{A}}\,\omega\,(a^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)\,\alpha))$. We can coerce the input along two equalities:

$$a^{\mathsf{A}}\,(\nu^{\mathsf{A}}\,\mathsf{con}) \overset{\nu^{\mathsf{C}}\,\mathsf{id}^{-1}}{=} a^{\mathsf{A}}\,(\Gamma^{\mathsf{C}}\,\nu) \overset{a^{\mathsf{C}}\,\nu^{-1}}{=} \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a[\nu]),$$

so that we get an $u : \mathsf{Tm}\,\Omega\,(\mathsf{El}\,a[\nu])$. Now we use the induction hypothesis for B and get $B^{\mathsf{R}}\,(\nu, u)\,(t \mathbin{@} u) : B^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu, \mathsf{refl})\,(t^{\mathsf{A}}\,\mathsf{con}\,(u^{\mathsf{A}}\,\mathsf{con}))\,(t^{\mathsf{A}}\,\omega\,(u^{\mathsf{A}}\,\omega))$. We can show that this type is equal to the one we need by coercing along the following two equalities.

$$u^{\mathsf{A}}\,\mathsf{con} \overset{u^{\mathsf{C}}\,\mathsf{id}^{-1}}{=} u \equiv \alpha \qquad\qquad u^{\mathsf{A}}\,\omega \overset{a^{\mathsf{R}}\,\nu}{=} a^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)\,u \equiv a^{\mathsf{M}}\,(\Gamma^{\mathsf{R}}\,\nu)\,\alpha$$

After defining $-^{\mathsf{R}}$, we recover the recursor using the identity substitution:

$$\mathsf{rec}_{\Omega}\,(\omega : \Omega^{\mathsf{A}}) : \Omega^{\mathsf{M}}\,\mathsf{con}_{\Omega}\,\omega :\equiv \Omega^{\mathsf{R}_{\omega}}\,\mathsf{id}$$

In Section 7 we prove uniqueness of the recursor from the existence of the eliminator (which is proved in Section 6).

6 DISPLAYED ALGEBRAS, SECTIONS AND THE ELIMINATOR

Displayed algebras are given by the unary logical predicate interpretation [Atkey et al. 2014; Bernardy et al. 2012]. Contexts become predicates over algebras, types become dependent predicates, and substitutions and terms produce witnesses of the logical predicates once the logical predicate is witnessed at the context.

$$\begin{aligned}
(\Gamma : \mathsf{Con})^{\mathsf{D}} &: \Gamma^{\mathsf{A}} \to \mathsf{Set}\\
(A : \mathsf{Ty}\,\Gamma)^{\mathsf{D}} &: \Gamma^{\mathsf{D}}\,\gamma \to A^{\mathsf{A}}\,\gamma \to \mathsf{Set}\\
(\sigma : \mathsf{Sub}\,\Gamma\,\Delta)^{\mathsf{D}} &: \Gamma^{\mathsf{D}}\,\gamma \to \Delta^{\mathsf{D}}\,(\sigma^{\mathsf{A}}\,\gamma)\\
(t : \mathsf{Tm}\,\Gamma\,A)^{\mathsf{D}} &: (\gamma^{D} : \Gamma^{\mathsf{D}}\,\gamma) \to A^{\mathsf{D}}\,\gamma^{D}\,(t^{\mathsf{A}}\,\gamma)
\end{aligned}$$

Here the interpretation of U, El and Π are the usual ones, U becomes predicate space ($\mathsf{U}^{\mathsf{D}}\,\gamma^{D}\,T :\equiv T \to \mathsf{Set}$), $\mathsf{El}\,a$ is just a witness of the predicate for a, and function space is interpreted as the predicate which expresses preservation of logical predicates. The details can be found in Appendix A.

Sections are dependent variants of the homomorphism interpretation $-^{\mathsf{M}}$ described in the previous section. A context becomes a dependent binary relation where the second argument of the relation depends on the first one. A type becomes a dependent relation over a dependent relation, and substitutions and terms become fundamental lemmas.

$$\begin{aligned}
(\Gamma : \mathsf{Con})^{\mathsf{S}} &: (\gamma : \Gamma^{\mathsf{A}}) \to \Gamma^{\mathsf{D}}\,\gamma \to \mathsf{Set}\\
(A : \mathsf{Ty}\,\Gamma)^{\mathsf{S}} &: \Gamma^{\mathsf{S}}\,\gamma\,\gamma^{D} \to (\alpha : A^{\mathsf{A}}\,\gamma) \to A^{\mathsf{D}}\,\gamma^{D}\,\alpha \to \mathsf{Set}\\
(\sigma : \mathsf{Sub}\,\Gamma\,\Delta)^{\mathsf{S}} &: \Gamma^{\mathsf{S}}\,\gamma\,\gamma^{D} \to \Delta^{\mathsf{S}}\,(\sigma^{\mathsf{A}}\,\gamma)\,(\sigma^{\mathsf{D}}\,\gamma^{D})\\
(t : \mathsf{Tm}\,\Gamma\,A)^{\mathsf{S}} &: (\gamma^{S} : \Gamma^{\mathsf{S}}\,\gamma\,\gamma^{D}) \to A^{\mathsf{S}}\,\gamma^{S}\,(t^{\mathsf{A}}\,\gamma)\,(t^{\mathsf{D}}\,\gamma^{D})
\end{aligned}$$

For the eliminator we fix a signature Ω and a displayed algebra over the initial algebra ω^D : $\Omega^\mathsf{D}\,\mathsf{con}_\Omega$. Then we define the operation $-^\mathsf{E}$ by induction on the syntax. The specification is as follows.

$$\begin{aligned}
&(\Gamma : \mathsf{Con})^\mathsf{E} && : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Gamma^\mathsf{S}\,(\nu^\mathsf{A}\,\mathsf{con})\,(\nu^\mathsf{D}\,\omega^D)\\
&(A : \mathsf{Ty}\,\Gamma)^\mathsf{E} && : (\nu : \mathsf{Sub}\,\Omega\,\Gamma)(t : \mathsf{Tm}\,\Omega\,(A[\nu])) \to A^\mathsf{S}\,(\Gamma^\mathsf{E}\,\nu)\,(t^\mathsf{A}\,\mathsf{con})\,(t^\mathsf{D}\,\omega^D)\\
&(\sigma : \mathsf{Sub}\,\Gamma\,\Delta)^\mathsf{E} && : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to \Delta^\mathsf{E}\,(\sigma \circ \nu) = \sigma^\mathsf{S}\,(\Gamma^\mathsf{E}\,\nu)\\
&(t : \mathsf{Tm}\,\Gamma\,A)^\mathsf{E} && : (\nu : \mathsf{Sub}\,\Omega\,\Gamma) \to A^\mathsf{E}\,\nu\,(t[\nu]) = t^\mathsf{S}\,(\Gamma^\mathsf{E}\,\nu)
\end{aligned}$$

The definition is analogous to that of $-^\mathsf{R}$. One difference is that for U we have an additional transport. The type that we need is $(\alpha : a^\mathsf{A}\,\mathsf{con}) \to (a^\mathsf{D}\,\omega^D\,\alpha)$. The expression $\mathsf{coe}\,(a^\mathsf{C}\,\mathsf{id}^{-1})\,\alpha$ has type $\mathsf{Tm}\,\Omega\,(\mathsf{El}\,a)$. Now, instead of applying the standard interpretation, we apply the logical predicate interpretation: $(\mathsf{coe}\,(a^\mathsf{C}\,\mathsf{id}^{-1})\,\alpha)^\mathsf{D}\,\omega^D$ has type $a^\mathsf{D}\,\omega^\mathsf{D}\,(\alpha^\mathsf{A}\,\mathsf{con})$ which is almost right, so we need to coerce along $\alpha^\mathsf{C}\,\mathsf{id}$ which witnesses $\alpha^\mathsf{A}\,\mathsf{con} = \alpha$, and we are finished. For details see Appendix A and the Agda formalisation.

Once we defined the $-^\mathsf{E}$ operation, we recover the eliminator using the identity substitution:

$$\mathsf{elim}_\Omega\,(\omega^D : \Omega^\mathsf{D}\,\mathsf{con}_\Omega) : \Omega^\mathsf{S}\,\mathsf{con}\,\omega^D :\equiv \Omega^{\mathsf{E}_{\omega^D}}\,\mathsf{id}$$

7 THE $\mathsf{CWF}^\mathsf{K}_\mathsf{Eq}$ MODEL OF THE THEORY OF SIGNATURES

In the previous sections, we have given a part of the semantics of QIITs, in order to make precise notions of algebras and induction. However, much is still missing:

- A *category* of algebras and homomorphisms, with the assorted category operations and laws.
- A proof that the properties of being initial and having induction are equivalent.

If we are aiming to get both, then having just a category of algebras is not sufficient, since it does not account for displayed algebras and their sections. We need additional structure. Thus, following the framework of [Nordvall Forsberg 2013], for each signature we construct a category with families, extended with constant families and extensional equality types. We call such a structure $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$. In any $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$, there is a simple native definition of induction, and it can be shown to be equivalent to initiality. In the following, we

- define what a $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$ is
- explain how the previously given operations on the theory of signatures yield part of a $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$ model, and present parts of the $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$ of natural number algebras as example
- show the equivalence of initiality and induction in any $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$
- construct a model of the theory of signatures, where every Γ : Con is interpreted as a $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$.

7.1 Defining $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$

The core of a $\mathsf{CwF}^\mathsf{K}_\mathsf{Eq}$ is just a CwF, of which the complete definition is already given in Section 3.1 as the substitution calculus of the theory of codes, consisting of twenty-four components from Con to , $\circ$. We extend this with two additional structures: *constant families* and *extensional equality*. As we shall see shortly, these are required for the proof that having induction for an object implies its initiality.

Constant families internalize every object (i.e. Con) as a family (Ty and Tm). The rules are as follows:

$$\begin{aligned}
&\mathsf{K} &&: \mathsf{Con} \to \mathsf{Ty}\,\Gamma\\
&\mathsf{K}[] &&: \mathsf{K}\,\Gamma\,[\sigma] = \mathsf{K}\,\Gamma\\
&\mathsf{mk} &&: \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta)\\
&\mathsf{unk} &&: \mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta) \to \mathsf{Sub}\,\Gamma\,\Delta\\
&\mathsf{K}\beta &&: \mathsf{unk}\,(\mathsf{mk}\,\sigma) = \sigma\\
&\mathsf{K}\eta &&: \mathsf{mk}\,(\mathsf{unk}\,t) = t\\
&\mathsf{mk}[] &&: (\mathsf{mk}\,\sigma)\,[\delta] = \mathsf{mk}\,(\sigma \circ \delta)
\end{aligned}$$

The above can be summarized by saying that there is a natural isomorphism between $\mathsf{Sub}\,\Gamma\,\Delta$ and $\mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta)$. An alternative definition is given by *democratic* CwFs [Clairambault and Dybjer 2014], and it was shown in [Nordvall Forsberg 2013] that the two definitions are interderivable.

Extensional equality has a standard definition, although we omit refl for now.

$$\begin{aligned}
&\mathsf{Eq} &&: \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma\\
&\mathsf{Eq}[] &&: \mathsf{Eq}\,t\,u\,[\sigma] = \mathsf{Eq}\,(t[\sigma])\,(u[\sigma])\\
&\mathsf{eqreflect} &&: \mathsf{Tm}\,\Gamma\,(\mathsf{Eq}\,t\,u) \to t = u
\end{aligned}$$

7.2 $\mathbf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s of Algebras

Previously given operations on the theory of signatures yield a fragment of a full $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ model. Algebras become the objects of a $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$, homomorphisms become the morphisms, and displayed algebras and sections together yield families. However, there are numerous other components in a $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ which we will consider in Section 7.4, where we construct the model.

To provide some intuition about $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s of algebras, we give here a tour of some of the definitions of components of the $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of natural number algebras. In the following, we shall use the syntactic names (such as Con, Ty, Tm) for the components of this $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$. This might be somewhat confusing at first, but we believe that there are advantages to thinking in terms of internal languages and using familiar type-theoretic syntax when reasoning about arbitrary $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s.

We start by defining Con as the type of $\mathbb{N}$-algebras, Ty as displayed $\mathbb{N}$-algebras, Sub as $\mathbb{N}$-homomorphisms and Tm as displayed $\mathbb{N}$-algebra sections, using the same definitions as in section 1.1.

An **empty context** corresponds to the terminal algebra. For $\mathbb{N}$-algebras, it is just the terminal set with trivial operations: $(\top, \mathsf{tt}, \lambda x.\, x)$.

Substitution composition and **identity substitution** respectively correspond to composition and identity for homomorphisms. We omit the exact definitions here.

Context extension is taking total algebras of displayed algebras. This is analogous to total categories of displayed categories [Ahrens and Lumsdaine 2017]. More concretely, we construct the Σ-type of the carrier set N and the family N^D, and glue together the algebra operators with their displayed counterparts.

$$\begin{aligned}
&- \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}\\
&(N, z, s) \triangleright (N^D, z^D, s^D) :\equiv (((n : N) \times N^D\,n),\ (z,\ z^D),\ (\lambda(x,\ x^D).\,(s\,x,\ s^D\,x\,x^D)))
\end{aligned}$$

The **first projection of a substitution** takes as input a homomorphism into a total algebra, and returns a homomorphism into the base algebra. The implementation is given by postcomposing the function on carrier sets with proj_1.

$$\pi_1 : \mathsf{Sub}\,\Gamma\,(\Delta \rhd A) \to \mathsf{Sub}\,\Gamma\,\Delta$$
$$\pi_1\,(N^M, z^M, s^M) :\equiv ((\lambda n.\,\mathsf{proj}_1\,(N^M\,n)),\ \mathsf{ap}\,\mathsf{proj}_1\,z^M,\ (\lambda n.\,\mathsf{ap}\,\mathsf{proj}_1\,(s^M\,n)))$$

Substitution is defined as reindexing of displayed algebras. For the first component of the implementation, we can turn a predicate on a carrier set to a predicate on another one, by precomposing a function, and implementations for the other components follow accordingly. Substitution is also functorial, as witnessed by the definitions of [id] and [$\circ$] which we omit here.

$$-[-] : \mathsf{Ty}\,\Delta \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma$$
$$(N^D, z^D, s^D)\,[\,(N^M, z^M, s^M)\,] :\equiv$$
$$((\lambda n.\,N^D\,(N^M\,n)),\ \mathsf{tr}_{N^D}\,(z^{M\,-1})\,z^D,\ (\lambda\,n\,n^D.\,\mathsf{tr}_{N^D}\,(s^M\,n^{-1})\,(s^D\,(N^M\,n)\,n^D)))$$

The **second projection of a substitution** takes as input a homomorphism into a total algebra, and returns a section of the displayed algebra part. Analogously to the first projection, the implementation is given by postcomposing the function part of the morphism with proj_2, but here the type is more complicated because of the necessary reindexing of the output (recall that $\pi_2 : (\sigma : \mathsf{Sub}\,\Gamma\,(\Delta \rhd A)) \to \mathsf{Tm}\,\Gamma\,(A[\pi_1\,\sigma])$).

We shall not elaborate the rest of the CwF components. For the current example of natural number algebras, it is helpful to keep in mind that if we take all the definitions of first components (corresponding to the carrier set), we just get the definition of the CwF of sets, and from there the definitions for the other components follow fairly mechanically.

Constant families for natural numbers are displayed algebras with a constant function for the predicate. In this case, the type of sections of constant families ($\mathsf{Tm}\,\Gamma\,(\mathsf{K}\,\Delta)$) is definitionally equal to the type of homomorphisms ($\mathsf{Sub}\,\Gamma\,\Delta$), so mk and unk could be both defined as identity functions.

$$\mathsf{K} : \mathsf{Con} \to \mathsf{Ty}\,\Gamma$$
$$\mathsf{K}\,(N, z, s) :\equiv ((\lambda\,\_.\,N),\ z,\ (\lambda\,\_.\,s))$$

Equality types are displayed algebras which carry information expressing the equality of two displayed algebra sections. For the function components of a section, the definition is just pointwise propositional equality of the functions. For the equality components, the definition is given by composition ($- \cdot -$) of equalities. Then, reflexivity and equality reflection can be given for this definition.

$$\mathsf{Eq} : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma$$
$$\mathsf{Eq}\,(N^{S0}, z^{S0}, s^{S0})\,(N^{S1}, z^{S1}, s^{S1}) :\equiv \big((\lambda n.\,N^{S0}\,n = N^{S1}\,n),\ z^{S0} \cdot z^{S1\,-1},$$
$$\lambda n\,p.\,s^{S0}\,n \cdot \mathsf{ap}\,(s^D\,n)\,p \cdot s^{S1}\,n^{-1}\big)$$

7.3 Equivalence of Initiality and Induction

First, we define initiality and induction in an arbitrary $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$.

$$\mathit{Initial} : \mathsf{Con} \to \mathsf{Set}$$
$$\mathit{Initial}\,\Gamma :\equiv (\Delta : \mathsf{Con}) \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \times ((\delta : \mathsf{Sub}\,\Gamma\,\Delta) \to \sigma = \delta)$$
$$\mathit{Induction} : \mathsf{Con} \to \mathsf{Set}$$
$$\mathit{Induction}\,\Gamma :\equiv (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Tm}\,\Gamma\,A$$

Initiality implies induction. Assume that $\Gamma : \mathsf{Con}$ and Γ is initial, and also $A : \mathsf{Ty}\,\Gamma$. We aim to inhabit $\mathsf{Tm}\,\Gamma\,A$. By initiality we get a unique $\sigma : \mathsf{Sub}\,\Gamma\,(\Gamma \rhd A)$. Now, $\pi_2\,\sigma$ has type $\mathsf{Tm}\,\Gamma\,(A[\pi_1\,\sigma])$, but since $\pi_1\,\sigma$ has type $\mathsf{Sub}\,\Gamma\,\Gamma$, it must be equal to the identity substitution by uniqueness, and then we can additionally transport $\pi_2\,\sigma$ over $[\mathsf{id}]$ to inhabit $\mathsf{Tm}\,\Gamma\,A$.

Induction implies initiality. Assume that $\Gamma : \mathsf{Con}$ and $ind : Induction\,\Gamma$, and $\Delta : Con$. We want to show that there is a unique inhabitant of $\mathsf{Sub}\,\Gamma\,\Delta$. Now, define σ as $\mathsf{unk}\,(ind\,(\mathsf{K}\,\Delta))$. Since σ has the right type, we only need to show its uniqueness. Assume an arbitrary $\delta : \mathsf{Sub}\,\Gamma\,\Delta$. Now, $ind\,(\mathsf{Eq}\,(\mathsf{mk}\,\delta)\,(ind\,(\mathsf{K}\,\Delta))$ has type $\mathsf{Tm}\,\Gamma\,(\mathsf{Eq}\,(\mathsf{mk}\,\delta)\,(ind\,(\mathsf{K}\,\Delta)))$, and it follows by equality reflection and $\mathsf{K}\beta$ that σ is equal to δ.

Induction is equivalent to initiality. We have established the logical equivalence of initiality and induction, but we can also show equivalence. Note that initiality is propositional, so we only need to show the same for induction. For some $\Gamma : \mathsf{Con}$ and $ind, ind' : Induction\,\Gamma$, and $A : \mathsf{Ty}\,\Gamma$, we have $\mathsf{eqreflect}\,(ind\,(\mathsf{Eq}\,(ind\,A)\,(ind'\,A)))$ with type $ind\,A = ind'\,A$, so by functional extensionality $ind = ind'$.

7.4 The $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ Model of the Theory of Signatures

For a QIIT signature Ω, in order to show the equivalence of induction (described by Ω^{D} and Ω^{S}) and initiality (described by Ω^{M}) we need to combine and extend these operations to a full $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ model. That is, to a model of the theory of signatures in which contexts are interpreted by $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s.

This involves a large amount of technical work. A $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ contains $24 + 7 + 3 = 34$ components corresponding to the fields of $\mathsf{CwF} + \mathsf{K} + \mathsf{Eq}$. For all 39 fields of the theory of signatures from Con to $\hat{@}[]$ we have to define these 34 components. We can imagine filling out a table with 39 rows (one for each field of the theory of signatures) and 34 columns (one for each component of $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$), see Figure 1.

There are two ways to present the model: by rows or by columns of the table. Describing the model by rows is the usual way of saying how contexts are given in the model, then how types are given, then substitutions, terms, the empty context, context extension and so on. Describing the model by columns means defining operations which interpret the syntax, and later operations can depend on previous ones. In Sections 4–6 we used the column-based method to describe the first four columns of this table. These were given by $-^{\mathsf{A}}$, $-^{\mathsf{D}}$, $-^{\mathsf{M}}$, and $-^{\mathsf{S}}$ (their full definition can be found in Appendix A).

In this section we follow the row-based approach and describe the rows of the model informally, while giving the full definition of the first 4 rows (Con, Ty, Sub, Tm) and the rows for U, El and Π in Appendix B. These are the most interesting parts of the construction. The rest of the construction is tedious and not very enlightening. The first 4 columns are fully formalised in Agda and most parts of the category-columns from the CwF-columns are also formalised.

The **CwF part** of the theory of signatures has to be interpreted as the CwF of $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s. Hence, Con is interpreted as just $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$, while Sub is interpreted as the set of strict $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$-morphisms (which strictly preserve all structure), Ty is interpreted as the set of displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s, and $- \triangleright -$ is interpreted as the total $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of a displayed one. Previously we illustrated how the CwF of natural number algebras work in section 7.2. Constructing the CwF of $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s is an analogous, albeit much larger work. All in all, this part is a mostly mechanical exercise.

For the **universe** U we have to give a concrete displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$, since U has type $\mathsf{Ty}\,\Gamma$, and we interpret types as displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s. However, U does not depend on Γ, so we can give a non-displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$, and then take the constant displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ for that.

We interpret U as the $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of sets. It contains the usual category of sets, and has families of sets and dependent functions for families, function composition for substitution, and extensional

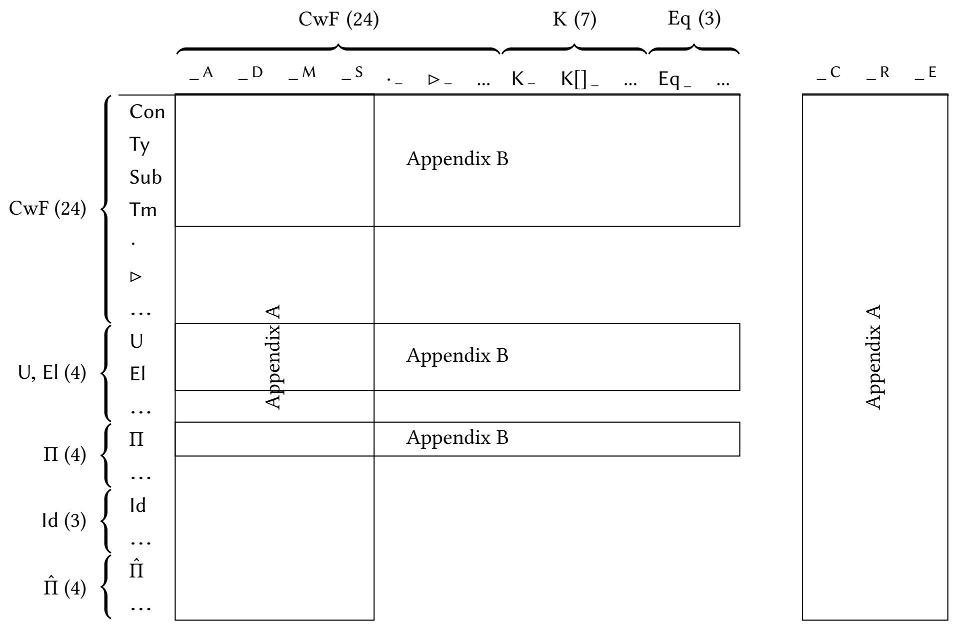

Fig. 1. The components in the $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ model of the theory of signatures. We mark which components are listed in the appendices.

equality and constant families defined in the obvious way. To see why this is the right choice, consider the one-element signature containing just a U. Since U^{A} is Set, the full interpretation of this signature must be the $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of sets.

For interpreting **universe decoding** $\mathsf{El}\,a$, we have $a : \mathsf{Tm}\,\Gamma\,\mathsf{U}$ and we need to construct a semantic $\mathsf{Ty}\,\Gamma$, i.e. a displayed $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$. Here, a is a $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$-section of the interpretation of U. However, we interpreted U as a family which is constantly the $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of sets, so a can be viewed as a morphism from Γ to the $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of sets. Hence, we can define $\mathsf{El}\,a$ as the discrete displayed $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ where there are only identity morphisms.

When interpreting the **function space with small domain**, we need to construct a $\mathsf{Ty}\,\Gamma$ from an $a : \mathsf{Tm}\,\Gamma\,\mathsf{U}$ and a $B : \mathsf{Ty}\,(\Gamma \rhd \mathsf{El}\,a)$. The result has to internalise morphisms from a to B. We know that $(\Pi\,a\,B)^{\mathsf{A}}\,\gamma \equiv (\alpha : a^{\mathsf{A}}\,\gamma) \to B^{\mathsf{A}}\,(\gamma, \alpha)$, which has to be the definition of the displayed objects in the result (a displayed object is a dependent function), since $-^{\mathsf{A}}$ should yield this part of the $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ model. Note that $(\alpha : a^{\mathsf{A}}\,\gamma) \to B^{\mathsf{A}}\,(\gamma, \alpha)$ is just a plain function space, without any conditions for functorality or structure preservation. Fortunately, we don't need such conditions because the domain a is discrete. In the implementation, all of the required displayed $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ structure is inherited from the codomain; for example, $- \rhd -$ is given by using the codomain's $- \rhd -$ pointwise, and π_1 is defined by postcomposing with the codomain's π_1. The interpretation of app generally follows the currying pattern seen in $\mathsf{app}^{\mathsf{A}}$.

The **function space with metatheoretic domain** is straightforward to interpret. We have a metatheoretic T : Set and a function $B : T \to \mathsf{Ty}\,\Gamma$, and we need to create a semantic $\mathsf{Ty}\,\Gamma$. The interpretation of B is a function which returns a displayed $\mathrm{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$, so it can be viewed as a function returning in a large iterated Σ-type. We can just utilize the equivalence of functions returning a Σ, and Σs of functions, by pushing the T parameter inside the result components. In short, displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$s are closed under arbitrary direct products. Then, $@$ is interpreted as pointwise application of each component to a metatheoretic argument.

For the **identity type**, we need to build a displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ representing the equality of t and u sections of some a discrete displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$. Because of the discreteness, a can be viewed as a morphism from a Γ $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ to the $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of sets, and t and u are essentially sections of families of sets. Hence, we can define the displayed objects in the result as pointwise equality of t and u (previously given in $-^{\mathsf{A}}$) and displayed displayed algebras as pointwise equalities over equalities (previously given in $-^{\mathsf{D}}$). Displayed morphisms, sections and all displayed $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ equations become trivial because of UIP. The interpretation of equality reflection can be given using functional extensionality.

This concludes the $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ model of the theory of signatures. Now, it follows that for each signature, there is a $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of algebras, and thus induction is equivalent to initiality by Section 7.3. Since for each signature we have constructed an algebra with induction in Section 4 and Section 6, it follows that the constructed algebras are also initial.

8 CONCLUSIONS AND FURTHER WORK

The present paper develops further the work in [Kaposi and Kovács 2018] where a syntax for HIITs was presented but no construction for HIITs was given. In the present paper we do construct initial algebras and (equivalently) eliminators, although in a restricted setting: we only consider quotient inductive-inductive types, and no higher equalities can be declared.

Note that the current theory of signatures is universal for closed QIITs: this means that the theory of signatures without $\hat{\Pi}$ can describe its own signature. This perhaps opens the way for type theories with *levitated* QIIT codes in the style of [Chapman et al. 2010]. Also, the theory of closed QIIT signatures can be viewed as a fragment of extensional type theory, hence this part of our work can be viewed as a reduction of closed QIITs to the existence of the syntax of extensional type theory.

Another limitation of the current work is that we only allow finitary constructors, i.e. we do not internalise a rule for externally indexed Π-types (Π^*). To obtain infinitary QIITs, we need to add another Π-type:

$$\Pi^* : (T : \mathsf{Set}) \to (T \to \mathsf{Tm}\,\Gamma\,\mathsf{U}) \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$$

such that there is a natural isomorphism between $\mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(\Pi^*\,T\,b))$ and $(\alpha : T) \to \mathsf{Tm}\,\Gamma\,(\mathsf{El}\,(b\,\alpha))$.

Using the current definition, we are unable to interpret this type former in the construction of initial algebras, because where we need to derive a propositional equality, we only get an isomorphism. Since our constructions rely on UIP, we cannot appeal to univalence to solve this. Hence, we are unable to represent QIITs which are infinitely branching such as W-types or the Cauchy reals [The Univalent Foundations Program 2013]. A potential solution would be to replace the propositional equalities in the initial algebra construction (in the interpretation of terms and substitutions) with isomorphisms in the $\mathsf{CwF}^{\mathsf{K}}_{\mathsf{Eq}}$ of algebras. We leave this for future work.

The restriction to QIITs instead of HIITs seems harder to overcome. In any case, since homomorphisms of non-truncated algebras are generally not homotopy sets, we would need to handle higher categories of algebras in the semantics, which poses considerable technical difficulty, and there is no known practical way to encode them in our currently used metatheory (Martin-Löf type theory).

Also, there is a *coherence* problem when interpreting the theory signatures in a setting without UIP: in such a setting, we need to set-truncate the syntax, but the metatheoretic universe is not a set, hence we can't eliminate into it. This prevents us already from defining the $-^{A}$ operation, i.e. the standard model. If we want to omit truncation, we need to add coherence laws, e.g. we need to replace categories with $(\infty, 1)$-categories, which could be defined in a two-level type theory [Capriotti and Kraus 2017], but it is not clear in general what these coherences should be. However, if such a higher syntax for signatures is possible, then perhaps the HIIT of higher signatures would be universal for HIITs.

ACKNOWLEDGMENTS

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.

REFERENCES

Michael Abbott, Thorsten Altenkirch, and Neil Ghani. 2005. Containers — Constructing Strictly Positive Types. *Theoretical Computer Science* 342 (September 2005), 3–27. Applied Semantics: Selected Topics.

Benedikt Ahrens and Peter LeFanu Lumsdaine. 2017. Displayed Categories. arXiv:arXiv:1705.04296

Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. 2018. Quotient inductive-inductive types. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 293–310.

Thorsten Altenkirch and Ambrus Kaposi. 2016. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodik and Rupak Majumdar (Eds.). ACM, 18–29. https://doi.org/10.1145/2837614.2837638

Robert Atkey, Neil Ghani, and Patricia Johann. 2014. A relationally parametric model of dependent type theory. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 503–516. https://doi.org/10.1145/2535838.2535852

Steve Awodey, Jonas Frey, and Sam Speight. 2018. Impredicative Encodings of (Higher) Inductive Types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '18)*. ACM, New York, NY, USA, 76–85. https://doi.org/10.1145/3209108.3209130

Henning Basold and Herman Geuvers. 2016. Type Theory Based on Dependent Inductive and Coinductive Types. In *Proceedings of LICS '16*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 327–336. https://doi.org/10.1145/2933575.2934514

Henning Basold, Herman Geuvers, and Niels van der Weide. 2017. Higher Inductive Types in Programming. *Journal of Universal Computer Science* 23, 1 (jan 2017), 63–88.

Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for Free — Parametricity for Dependent Types. *Journal of Functional Programming* 22, 02 (2012), 107–152. https://doi.org/10.1017/S0956796812000056

Paolo Capriotti and Nicolai Kraus. 2017. Univalent Higher Categories via Complete Semi-Segal Types. *Proc. ACM Program. Lang.* 2, POPL, Article 44 (Dec. 2017), 29 pages. https://doi.org/10.1145/3158132

John Cartmell. 1986. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic* 32 (1986), 209–243.

James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. 2010. The gentle art of levitation. *ACM Sigplan Notices* 45, 9 (2010), 3–14.

Pierre Clairambault and Peter Dybjer. 2014. The biequivalence of locally cartesian closed categories and Martin-Löf type theories. *Mathematical Structures in Computer Science* 24, 6 (2014).

Jesper Cockx, Dominique Devriese, and Frank Piessens. 2014. Pattern Matching Without K. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming (ICFP '14)*. ACM, New York, NY, USA, 257–268. https://doi.org/10.1145/2628136.2628139

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2016. Cubical Type Theory: a constructive interpretation of the univalence axiom. *CoRR* abs/1611.02108 (2016). arXiv:1611.02108 http://arxiv.org/abs/1611.02108

Thierry Coquand, Simon Huber, and Anders Mörtberg. 2018. On Higher Inductive Types in Cubical Type Theory. *LICS '18: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (2018).

Peter Dybjer. 1996. Internal Type Theory. In *Lecture Notes in Computer Science*. Springer, 120–134.

Peter Dybjer. 1997. Inductive Families. *Formal Aspects of Computing* 6 (1997), 440–465.

Peter Dybjer. 2000. A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory. *Journal of Symbolic Logic* 65 (2000), 525–549.

Peter Dybjer and Hugo Moeneclaey. 2018. Finitary Higher Inductive Types in the Groupoid Model. *Electronic Notes in Theoretical Computer Science* 336 (2018), 119 – 134. https://doi.org/10.1016/j.entcs.2018.03.019 The Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII).

Martin Hofmann. 1995a. Conservativity of Equality Reflection over Intensional Type Theory.. In *TYPES 95*. 153–164.

Martin Hofmann. 1995b. *Extensional concepts in intensional type theory.* University of Edinburgh, Department of Computer Science. http://books.google.co.uk/books?id=HK3xtgAACAAJ

Ambrus Kaposi. 2017. *Type theory in a type theory with quotient inductive types.* Ph.D. Dissertation. University of Nottingham.

Ambrus Kaposi and András Kovács. 2018. A Syntax for Higher Inductive-Inductive Types. In *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018) (Leibniz International Proceedings in Informatics (LIPIcs))*, Hélène Kirchner (Ed.), Vol. 108. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 20:1–20:18. https://doi.org/10.4230/LIPIcs.FSCD.2018.20

Peter LeFanu Lumsdaine and Mike Shulman. 2017. Semantics of higher inductive types. arXiv:arXiv:1705.07088

Peter Morris and Thorsten Altenkirch. 2009. Indexed Containers. In *Twenty-Fourth IEEE Symposium in Logic in Computer Science (LICS 2009)*.

Fredrik Nordvall Forsberg. 2013. *Inductive-inductive definitions.* Ph.D. Dissertation. Swansea University.

Nicolas Oury. 2005. *Extensionality in the calculus of constructions.* Springer Berlin Heidelberg, Berlin, Heidelberg, 278–293. https://doi.org/10.1007/11541868_18

Christine Paulin-Mohring. 1993. Inductive Definitions in the system Coq — Rules and Properties. In *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings (Lecture Notes in Computer Science)*, Marc Bezem and Jan Friso Groote (Eds.), Vol. 664. Springer, 328–345. https://doi.org/10.1007/BFb0037116

John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, September 19-23, 1983*, R. E. A. Mason (Ed.). Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 513–523.

Kristina Sojakova. 2015. Higher Inductive Types As Homotopy-Initial Algebras. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15).* ACM, New York, NY, USA, 31–42. https://doi.org/10.1145/2676726.2676983

The Univalent Foundations Program. 2013. *Homotopy type theory: Univalent foundations of mathematics.* Technical Report. Institute for Advanced Study.

Niels van der Weide. 2016. *Higher Inductive Types.* Ph.D. Dissertation. Radboud University, Nijmegen. Master's thesis.

Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. 2018. Using reflection to eliminate reflection. In *24th International Conference on Types for Proofs and Programs, TYPES 2018*, José Espírito Santo and Luís Pinto (Eds.). University of Minho.