QUOTIENT INDUCTIVE-INDUCTIVE TYPES

THORSTEN ALTENKIRCH, PAOLO CAPRIOTTI, GABE DIJKSTRA, NICOLAI KRAUS, AND FREDRIK NORDVALL FORSBERG

Abstract. Higher inductive types (HITs) in Homotopy Type Theory (HoTT) allow the definition of datatypes which have constructors for equalities over the defined type. HITs generalise quotient types, and allow to define types which are not sets in the sense of HoTT (i.e. do not satisfy uniqueness of equality proofs) such as spheres, suspensions and the torus. However, there are also interesting uses of HITs to define sets, such as the Cauchy reals, the partiality monad, and the well-typed syntax of type theory. In each of these examples we define several types that depend on each other mutually, i.e. they are inductive-inductive definitions. We call those HITs quotient inductive-inductive types (QIITs). Although there has been recent progress on a general theory of HITs, there is not yet a theoretical foundation for the combination of equality constructors and induction-induction, despite having many interesting applications. In the present paper we present a first step towards a semantic definition of QIITs. In particular, we give an initial-algebra semantics and show that this is equivalent to the section induction principle, which justifies the intuitively expected elimination rules.

1. Introduction

This paper is about Type Theory in the sense of Martin-Löf [25], a theory which proof assistants such as Coq [6] and Lean [12] as well as programming languages such as Agda [26] and Idris [7] are based on. Recently, Homotopy Type Theory (HoTT) [29] has been introduced inspired by homotopy theoretic interpretations of Type Theory by Awodey and Warren [4] and Voevodsky [21; 30].

One of the central concepts in Type Theory are inductive definitions, which allow us to define inductive datatypes like the natural numbers, lists and infinite trees just by presenting constructors which use the inductive type in a strictly positive way. Using the propositions as types explanation we can use the same mechanism to inductively define predicates and relations like less than or equal, or the derivability predicate for a logic defined by rules. Conceptually, HoTT changes what we mean by an inductive definition, because we view a type not only as given by its elements (points) but also by its equality types (paths). Hence an inductive definition may not only feature constructors for elements but also for equality. This concept of a higher inductive type (HIT) has been used to represent the homotopical structure of geometric objects, like circles, sphere and tori, and gives rise to synthetic homotopy theory in HoTT [27].

However, as already noted in the HoTT Book [29], higher inductive types have also more mundane applications, such as the definition of the Cauchy reals in a way that avoids the use of the axiom of choice when proving properties like Cauchy completeness of the reals. Here instead of defining the real numbers as a quotient of sequences of rational numbers, a HIT is used to define them as the Cauchy completion of the rational numbers. Similarly, a definition of the partiality monad which represents potentially diverging operations over a given type was given using a HIT [2], again avoiding the axiom of choice when showing for example that the construction is a monad [11].

An important observation is that the idea of generating points and equalities of a type inductively is interesting, even if we do not care about the higher equality structure of types, or if we do not want such non-trivial higher structure, for example if we stay inside the universe Set. To see this, let us look at an example: consider trees branching over an arbitrary type, quotiented by arbitrary permutations of subtrees. We first define the type $T_0(X)$ of X-branching trees, given by the constructors

$$\begin{aligned}\mathsf{leaf}_0 &: T_0(X)\\ \mathsf{node}_0 &: (X \to T_0(X)) \to T_0(X).\end{aligned}$$

We then form the relation $R : T_0(X) \times T_0(X) \to \mathsf{Set}$ that we want to quotient by as follows: R is the smallest relation such that for any auto-equivalence on X (i.e. any $e : X \to X$ which has an inverse) and $f : X \to T_0(X)$, we have a proof $p_{f,e} : R(\mathsf{node}_0(f), \mathsf{node}_0(f \circ e))$, and, secondly, for $g, h : X \to T_0(X)$ such that $(n : X) \to R(g(n), h(n))$, we have a proof $c_{f,g} : R(\mathsf{node}_0(g), \mathsf{node}_0(h))$. We can then form the quotient type $T_0(X)/R$, which is the type of unlabelled trees where each node has an X-indexed family of subtrees, and two trees which agree modulo the "order" of its subtrees are equal. For $X \equiv \mathbf{2}$, these are binary trees where the order of the two subtrees of each node does not matter.

Now, morally, from a family $X \to (T_0(X)/R)$, we should be able to construct an element of the quotient $T_0(X)/R$. This is indeed possible if X is $\mathbf{2}$ or another finite type, by applying the induction principle of the quotient type X times. However, it seems that, for a general type X, this would require the axiom of choice [29], which unfortunately is not a constructive principle [13]. But using a higher inductive type, we can give an alternative definition for the type of A-branching trees modulo permutation of subtrees.

Example 1. Given a type A, we define $T(A) : \mathsf{Set}$ by

$$\begin{aligned}\mathsf{leaf} &: T(A)\\ \mathsf{node} &: (A \to T(A)) \to T(A)\\ \mathsf{mix} &: (e : A \to A) \to \mathsf{isEquivalence}(e) \to (f : A \to T(A))\\ &\qquad \to \mathsf{node}(f) = \mathsf{node}(f \circ e).\end{aligned}$$

Note that in the above example, a set-truncation constructor is implicitly included in the statement $T(A) : \mathsf{Set}$, which ensures that $T(A)$ lives in Set. The construction we were looking for is now directly given by the constructor node. This demonstration of the usefulness of higher inductive constructions to increase the strength of quotients was first discussed in Altenkirch and Kaposi [1], where such set-truncated HITs are called *quotient inductive types* (QITs).

Another example of the use of higher inductive types is *type theory in type theory* [1], where the well-typed syntax of type theory is implemented as a higher inductive-inductive type in type theory itself. A significantly simplified version of this will serve as a running example for us:

Example 2. We define the syntax of a (very basic) type theory by constructing types representing contexts and types as follows. A set $\mathsf{Con} : \mathsf{Set}$ and a type family $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Set}$ are simultaneously defined by giving the constructors

$$\begin{aligned}\varepsilon &: \mathsf{Con}\\ \mathsf{ext} &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}(\Gamma) \to \mathsf{Con}\\ \iota &: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}(\Gamma)\\ \sigma &: (\Gamma : \mathsf{Con}) \to (A : \mathsf{Ty}(\Gamma)) \to \mathsf{Ty}(\mathsf{ext}\,\Gamma\,A) \to \mathsf{Ty}(\Gamma)\\ \sigma_{\mathrm{eq}} &: (\Gamma : \mathsf{Con}) \to (A : \mathsf{Ty}(\Gamma)) \to (B : \mathsf{Ty}(\mathsf{ext}\,\Gamma\,A))\\ &\qquad \to \mathsf{ext}\,(\mathsf{ext}\,\Gamma\,A)\,B =_{\mathsf{Con}} \mathsf{ext}\,\Gamma\,(\sigma\,\Gamma\,A\,B).\end{aligned}$$

For simplicity, we do not consider terms. Contexts are either empty ε, or an extended context $\mathsf{ext}\,\Gamma\,A$ representing the context Γ extended by a fresh variable of type A. Types are either the base type ι (well-typed in any context), or Σ-types represented by $\sigma\,\Gamma\,A\,B$ (well-typed in context Γ if A is well-typed in context Γ, and B is well-typed in the extended context $\mathsf{ext}\,\Gamma\,A$). Type theory in type theory as in [1] has plenty of equality constructors which play a role as soon as terms are introduced. To keep the example simple we instead use another equality, stating that extending a context by A followed by B is equal to extending it by $\Sigma\,A\,B$. This equality is given by σ_{eq}. Note that it is not possible to list the constructors of Con and Ty separately: due to the mutual dependency, the Ty-constructor σ has to be given in between of the two Con-constructors ext and σ_{eq}.

Despite a lot of work in the literature making use of concrete HITs [23; 22; 3; 10; 20; 8; 9], and despite the fact that it is usually on some intuitive level clear for the expert how the elimination principle for such a HIT can be derived, giving a general specification and a theoretical foundation for HITs has turned out to be a major difficulty. Several approaches have been proposed, and they do indeed give a satisfactory specification of HITs in the sense that they cover all HITs which have been used so far (we will discuss related work in a moment). However, to the best of our knowledge, there is no approach which covers *higher inductive-inductive* definitions such as Example 2. In a nutshell, the purpose of the current paper is to remedy this. We restrict ourselves to sets, i.e. to *quotient inductive-inductive types* (QIITs). This of course is a serious restriction since it means that we cannot capture many ordinary HITs such as the circle $\mathbb{S}^1$. At the same time, all higher *inductive-inductive* types that we know of are indeed sets (the Cauchy reals, the surreal numbers, the partiality monad, type theory in type theory, permutable trees), and will be instances of our general specification. Our framework allows arbitrarily complicated dependency structures. In particular, we allow intermixing of constructors as in Example 2.

Contributions. We give a formal specification of *quotient inductive-inductive types* with arbitrary dependency structure. This can be viewed as the generalisation of the usual semantics of inductive types as initial algebras of a functor to the case of quotient inductive-inductive types. We establish conditions on the functorial specification of QIITs that allow us to conclude that the categories of algebras are complete. This is important because it allows us to prove the equivalence of initiality and section-induction, justifying the expected elimination principles.

Related Work. Sojakova [28] shows the correspondence between initiality and induction (a variant of our Theorem 34) for a restricted class of HITs called W-suspensions. Basold, Geuvers and van der Weide [5] introduce a syntactic schema for HITs without higher path constructors, and derive the elimination rules for them. Dybjer and Moeneclaey [16] give a syntactic schema for finitary HITs with at most paths between paths, and give an interpretation in Hofmann and Streicher's groupoid model [19]. Finally, the work by Lumsdaine and Shulman on the semantics of higher inductive types in model categories [24] is somewhat similar to an external version of the approach we take in this paper.

Preliminaries. We work in a standard Martin-Löf style type theory and assume function extensionality. We write Set for a type universe which contains types satisfying UIP (*sets* in the terminology of HoTT), and we mostly work with types of this universe. Univalence is not needed in our development. When we talk of a *category*, we mean a precategory in the sense of [29] (all our categories become univalent categories if we assume the univalence axiom). We write $\mathcal{C} \Rightarrow \mathcal{D}$ for

functors and $X \to Y$ for functions between types. Note that Set refers to both the universe and the obvious category of sets and functions, and consequently, $F : A \to \mathsf{Set}$ is a type family, while $F : \mathcal{C} \Rightarrow \mathsf{Set}$ is a functor. Further, we write $\int^{\mathcal{C}} F$ for the *category of elements* of F. Recall that this is the category which as objects has pairs (X, x) of an object X in $\mathcal{C}$ and an element $x : FX$. For a function $f : X \to Y$ and $z, w : X$, we write $\mathsf{ap}\ f\ : z = w \to f(z) = f(w)$ for the usual "application of a function to paths" [29, Lemma 2.2.1], $^{-1} : x = y \to y = x$ for "path reversal", and $\cdot : x = y \to y = z \to x = z$ for and "path concatenation" [29, Lemmas 2.1.1 and 2.1.2].

2. Sorts

Single inductive (and quotient inductive) sets are simply elements of Set. Inductive families [15] indexed over some fixed type A are families $A \to \mathsf{Set}$. For the inductive-inductive definitions we are considering, the situation is more complicated, since we allow very general dependency structures. Our only requirement is that there is no looping dependency, since this is easily seen to lead to contradictions, e.g. we do not allow the definition of a family $A : B \to \mathsf{Set}$ mutually with a family $B : A \to \mathsf{Set}$ (whatever this would mean). Concretely, we will ensure that the collection of type formation rules (the type signatures) is given in a valid order, and we refer to the types involved as the *sorts* of the definition. Hence our first step towards a specification of general QIITs is to explain what a valid specification of the sorts is.

Sorts do not only determine the formation rules of the inductive definitions, but also the types of the eliminators. To capture this, it is not enough to specify a type of sorts — in order to take the shape of the elimination rules into account, we need to specify a category.

Definition 3 (Sorts)**.** A specification of the *sorts* of a quotient inductive-inductive definition of n types is given by a list

$$H_0, H_1, \ldots, H_{n-1},$$

where each H_i is a functor $H_i : \mathcal{C}_i \Rightarrow \mathsf{Set}$. Here, Set is the category of sets (in the sense of HoTT, i.e. types with trivial equality types) and functions, $\mathcal{C}_0 :\equiv \mathbf{1}$ is the terminal category, and $\mathcal{C}_{i+1}$ is defined as follows:

- objects are pairs (X, P), where X is an object in $\mathcal{C}_i$, and $P : H_i(X) \to \mathsf{Set}$ is a family of sets;
- a morphism $(f, g) : (X, P) \to (Y, Q)$ consists of a morphism $f : X \to Y$ in $\mathcal{C}_i$, and a dependent function $g : (x : H_i(X)) \to P(x) \to Q(H_i(f)\, x)$ (in Set).

We say that $\mathcal{C}_n$ is the *base category* for the sort signature $H_0, \ldots, H_{n-1}$.

The following examples will hopefully make clear the connection between the specification in Definition 3 and common classes of data types.

Example 4 (Permutable trees)**.** For a single inductive type such as the type of trees $T(A)$ in Example 1, the sorts are specified by a single functor $H_0 : \mathcal{C}_0 \to \mathsf{Set}$ which maps the single object $\star$ of $\mathcal{C}_0$ to the unit type $\mathbf{1}$. Objects in the base category $\mathcal{C}_1$ are thus pairs $(\star, W)$, where $W : \mathbf{1} \to \mathsf{Set}$, and morphisms are given by $f : \star \to \star$ in $\mathbf{1}$ (necessarily the identity morphism), together with a dependent function $g : (\star : \mathbf{1}) \to A(\star) \to B(\star)$. It is easy to see that this category $\mathcal{C}_1$ is equivalent to the category Set.

Example 5 (The finite types)**.** Consider the inductive family $\mathsf{Fin} : \mathbb{N} \to \mathsf{Set}$ of finite types. Again, this is a single type family, i.e. we are in the case $n \equiv 1$. We have

$H_0(\star) :\equiv \mathbb{N}$, and the base category $\mathcal{C}_1$ is equivalent to the category of $\mathbb{N}$-indexed families, where objects are families $X : \mathbb{N} \to \mathsf{Set}$ and morphisms $\mathcal{C}_1(X, Y)$ are dependent functions $f : (n : \mathbb{N}) \to X(n) \to Y(n)$.

Example 6 (Contexts and types)**.** Let us consider the QIIT $(\mathsf{Con}, \mathsf{Ty})$ from Example 2. Here, we need two functors H_0, H_1, the first corresponding to Con and the second to Ty. The first is given by $H_0(\star) :\equiv \mathbf{1}$ as in Example 4, since Con is a type on its own. Next, we need $H_1 : \mathcal{C}_1 \to \mathsf{Set}$. Applying the equivalence between $\mathcal{C}_1$ and Set established in Example 4, we define H_1 to be the identity functor $H_1(A) :\equiv A$, since then $\mathsf{Ty} : H_1(\mathsf{Con}) \to \mathsf{Set}$. The base category $\mathcal{C}_2$ is equivalent to the category $\mathsf{Fam}(\mathsf{Set})$, whose objects are pairs (A, B) where $A : \mathsf{Set}$ and $B : A \to \mathsf{Set}$, and whose morphisms (A, B) to (A', B') consist of functions $f : A \to A'$ together with dependent functions $g : (x : A) \to B(x) \to B'(f\, x)$.

Example 7 (the Cauchy reals)**.** Recall that the Cauchy reals in the HoTT book [29] are constructed by simultaneously defining $\mathbb{R} : \mathsf{Set}$ and $\sim : \mathbb{R} \times \mathbb{R} \to \mathsf{Set}$ (we ignore the fact that [29] uses $\mathcal{U}$ instead of Set). This time the sorts H_0, H_1 are given by $H_0(\star) :\equiv \mathbf{1}$ and $H_1(A) :\equiv A \times A$, corresponding to the fact that $\sim$ is indexed *twice* over $\mathbb{R}$. The base category has (up to equivalence) pairs (X, Y) with $Y : X \times X \to \mathsf{Set}$ as objects, and morphisms are defined accordingly.

Example 8 (The full syntax of type theory)**.** Altenkirch and Kaposi [1] give the complete syntax of a basic type theory as a (at that point unspecified) QIIT. Although this construction is far too involved to be treated as an example in the rest of this paper (where we prefer to work with the simplified version of Example 2), we can give the sort signature H_0, H_1, H_2, H_3 of this QIIT. Apart from contexts Con and types Ty, this definition also involves context morphisms Tms and terms Tm:

$$\begin{array}{ll} \mathsf{Con} : \mathsf{Set} & \mathsf{Tms} : \mathsf{Con} \times \mathsf{Con} \to \mathsf{Set} \\ \mathsf{Ty} : \ \mathsf{Con} \to \mathsf{Set} & \mathsf{Tm} : \ \big(\Sigma \Gamma : \mathsf{Con}.\mathsf{Ty}(\Gamma)\big) \to \mathsf{Set}. \end{array}$$

We have:

$$\begin{array}{rl} H_0(\star) :\equiv \mathbf{1} & \mathcal{C}_1 \cong \mathsf{Set} \text{ as in Example 4;} \\ H_1(A) :\equiv A & \mathcal{C}_2 \cong \mathsf{Fam}(\mathsf{Set}) \text{ as in Example 6;} \\ H_2(A, B) :\equiv A \times A & \mathcal{C}_3 \text{ has objects } (A, B, C), \text{ where } C : A \times A \to \mathsf{Set}; \\ H_3(A, B, C) :\equiv \Sigma\, A\, B & \mathcal{C}_4 \text{ has objects } (A, B, C, D), \text{ where } D : \big(\Sigma\, A\, B\big) \to \mathsf{Set}. \end{array}$$

Remark 9. Although we work in type theory also in the meta-theory, we give the presentation informally in natural language. Formally, the specification of sorts and base categories of Definition 3 can be defined as an inductive-recursive definition [17] of the list $H_0, \ldots, H_n$ simultaneously with a function that turns such a list into a category. See Dijkstra [14, Section 4.3] for details.

The main result of this section states that every base category of a is complete, i.e. it has all small limits. By a small limit, we mean a limit of a diagram $D : \mathcal{I} \to \mathcal{C}$ where the shape category $\mathcal{I}$ has a set of objects and each hom-type is a set. This result will be needed later to show that categories of QIIT algebras are complete. Recall that Set has all small limits by a standard construction.

Theorem 10 (Base categories are complete)**.** *For any sort signature $H_0, \ldots, H_{n-1}$, the corresponding base category $\mathcal{C}_n$ has all small limits.*

Proof. We show that each $\mathcal{C}_k$ is complete by induction on k for $0 \leq k \leq n$. Clearly, $\mathcal{C}_0$ is complete. For the step case, assume that $\mathcal{C}_k$ is complete. By definition, the category $\mathcal{C}_{k+1}$ has as objects pairs (X, P), where X is an object of $\mathcal{C}_k$ and

data $\mathbb{Z} = \frac{\mathbb{N}\times\mathbb{N}}{\sim_{\mathbb{Z}}}$; data $\mathbb{Q} = \frac{\mathbb{Z}\times\mathbb{Z}^{\neq}}{\sim_{\mathbb{Q}}}$

$n,n' \sim m,m' \Leftrightarrow n+m' = n+m'$

$x,y \sim x',y' \Leftrightarrow xy' = yx'$

data $\mathbb{R} \coloneqq \sum_{f:\mathbb{Q}\to\mathbb{B}} f$ is cut $\times$ f is inhabited

$\frac{a}{b} \le \frac{c}{d} \Leftrightarrow ad \le cb$

$\forall x,y:\mathbb{Q}.\ x \le y \Rightarrow fx \le fy$

$f = g \Leftrightarrow \forall x:\mathbb{Q}.\ fx = gx$

$P : H_k(X) \to \mathsf{Set}$. By the usual correspondence between families and fibrations,[1] we can replace P by a pair (Y, h) of a type $Y : \mathsf{Set}$ and a function $h : Y \to H_k(X)$. This means $\mathcal{C}_{k+1}$ is equivalent to the category $\mathcal{D}$ with objects triples (X, Y, h). Morphisms between (X, Y, h) and (X', Y', k) are triples (f, g, e), where $f : X \to X'$, $g : Y \to Y'$, and $e : k \circ g = F(f) \circ h$.

Consider a diagram $D : \mathcal{I} \to \mathcal{D}$. We can split this into three components $D_X : \mathcal{I} \to \mathsf{Set}$, $D_Y : \mathcal{I} \to \mathsf{Set}$, and $D_h : (i : \mathcal{I}) \to D_Y(i) \to H_k(D_X(i))$. Consider the cospan

$$\begin{array}{ccc} & & \lim_{\mathcal{I}} D_Y \\ & & \downarrow{\scriptstyle \lim_{\mathcal{I}} D_h} \\ H_k(\lim_{\mathcal{I}} D_X) & \longrightarrow & \lim_{\mathcal{I}}(H_k \circ D_X) \end{array}$$

where all limits are taken in Set, and where the vertical map is the canonical one given by the universal property of the limit. This cospan is itself a diagram in Set, guaranteeing that the pullback of $\lim_{\mathcal{I}} D_h$ exists; let's call it $\tilde{h} : \tilde{Y} \to H_k(\lim_{\mathcal{I}} D_X)$. It is easy to check that $(\lim_{\mathcal{I}} D_X, \tilde{Y}, \tilde{h})$ is the limit of D. □

3. Algebras

Once the sorts of an inductive definition have been established, the next step is to specify the *constructors*. In this section, we will give a very general definition of constructor specifications, although we will mainly focus on two specific kinds: *point constructors*, which can be thought of as the operations of an algebraic signature, and *path constructors*, which correspond to the axioms.

Similarly to how sorts are specified one by one in Section 2, we can construct suitable categories of algebras by starting with a finitely complete category $\mathcal{C}$, such as the one obtained from a sort signature, and iteratively adding one constructor at a time. Therefore, we describe this process by describing how to specify a constructor on a finitely complete category $\mathcal{C}$, and show how to extend $\mathcal{C}$ using this constructor specification to get a new finitely complete category $\mathcal{C}'$. Once all constructors have been added, we obtain the sought-after inductive type as the underlying set of an initial object of the category at the last stage, provided this initial object exists. In the case of the inductive definition of natural numbers, this process will turn out as follows:

- we start with Set as our base category (only one trivial sort, as in Example 4);
- we add a point constructor for the constant corresponding to 0; the category of algebras at this stage is the category of pointed sets;
- we add a second point constructor for the operation corresponding to suc; the objects of the category of algebras at this stage are sets equipped with a point and a unary operation;
- the set of natural numbers, together with its usual structure, can now be regarded as an initial object in the category of algebras just constructed.

3.1. Relative Continuity and Constructor Specifications.

Roughly speaking, constructors at each stage are given by pairs of Set-valued functors F and G on $\mathcal{C}$, where G is continuous (i.e. preserves all small limits). The intuition is that F specifies the arguments of the constructor, while G determines its target. For instance, in the example of the natural numbers when specifying the constructor $\mathsf{suc} : \mathbb{N} \to \mathbb{N}$, $\mathcal{C}$ is the category of pointed sets, and both F and G are the forgetful functor to Set. The continuity condition on G is needed to make sure that the

[1] The correspondence is made precise in [29, Thm 4.8.3], but note that the equivalence of the two categories in consideration does not require univalence.

corresponding category of algebras is complete. Intuitively, this expresses the idea that a constructor should only "construct" elements of one of the sorts, or equalities thereof. In particular, a constant functor is usually not a valid choice for G.

Unfortunately, this simple description falls short of capturing many of the examples of QIITs mentioned in Section 1. The problem is that we want G to be able to depend on the elements of F. However, since F is assumed to be an arbitrary functor, its category of elements is not necessarily complete, and so we need to refine the the notion of G being continuous to this case.

Definition 11 (Relative continuity)**.** Let $\mathcal{C}$ be a category, $\mathcal{C}_0$ a complete category, and $U : \mathcal{C} \Rightarrow \mathcal{C}_0$ a functor. A cone over a small diagram in $\mathcal{C}$ is a *U-limit cone*, or *limit cone relative to U*, if it is mapped to a limit cone in $\mathcal{C}_0$ by U. A functor $\mathcal{C} \Rightarrow \mathsf{Set}$ is *continuous relative to U* if it maps U-limit cones to limit cones in Set.

In particular, the functor U in Definition 11 is continuous relative to itself. Furthermore, if $\mathcal{C}$ is complete and U creates limits, then relative continuity with respect to U reduces to ordinary continuity. If $\mathcal{C}$ is a complete category, and $F : \mathcal{C} \Rightarrow \mathsf{Set}$ is an arbitrary functor, the category $\int^{\mathcal{C}} F$ of elements of F is equipped with a forgetful functor into $\mathcal{C}$. We will implicitly consider relative limit cones and relative continuity with respect to this forgetful functor, unless specified otherwise. Note that if $\mathcal{C}$ is complete and F is continuous, then $\int^{\mathcal{C}} F$ is also complete, and relative continuity of functors on $\int^{\mathcal{C}} F$ is the same as continuity, as observed above.

We can now give a precise definition of what is needed to specify a constructor:

Definition 12 (Constructor specifications)**.** A *constructor specification* on a complete category $\mathcal{C}$ is given by:

- a functor $F : \mathcal{C} \Rightarrow \mathsf{Set}$, called the *argument* functor;
- a relatively continuous functor $G : \int^{\mathcal{C}} F \Rightarrow \mathsf{Set}$, called the *target* functor.

Example 13 (Permutable trees)**.** The constructor $\mathsf{leaf} : T(A)$ from Example 1 can be specified by functors $F_0 : \mathsf{Set} \Rightarrow \mathsf{Set}$ and $G_0 : \int^{\mathsf{Set}} F_0 \Rightarrow \mathsf{Set}$, where $F_0(A) :\equiv \mathbf{1}$ and $G_0(A, x) :\equiv A$. Note how F_0 specifies the (trivial) arguments of leaf, and G_0 the target. Next the constructor $\mathsf{node} : (A \to T(A)) \to T(A)$ can be specified by functors $F_1 : \mathsf{Set}_\bullet \Rightarrow \mathsf{Set}$ and $G_1 : \int^{\mathsf{Set}_\bullet} F_1 \Rightarrow \mathsf{Set}$, where $\mathsf{Set}_\bullet$ is the category of pointed sets (we think of the point as the previous constructor leaf): F_1 and G_1 are defined as $F_1(X, l) :\equiv A \to X$ and $G_1(X, l, f) :\equiv X$, so that

$$\mathsf{node} : (f : F_1(T(A), \mathsf{leaf})) \to G_1(T(A), \mathsf{leaf}, f).$$

Theorem 20 will show that G_0 and G_1 are relatively continuous.

Example 14 (Contexts and types)**.** The constructor σ_{eq} of type

$$(\Gamma : \mathsf{Con})(A : \mathsf{Ty}(\Gamma))(B : \mathsf{Ty}(\mathsf{ext}\,\Gamma\,A)) \to \mathsf{ext}\,(\mathsf{ext}\,\Gamma\,A)\,B =_{\mathsf{Con}} \mathsf{ext}\,\Gamma\,(\sigma\,\Gamma\,A\,B)$$

from Example 2 is specified in the context of the previous constructors ε, ext and σ by functors $F : \mathcal{C} \Rightarrow \mathsf{Set}$ and $G : \int^{\mathcal{C}} F \Rightarrow \mathsf{Set}$, where $\mathcal{C}$ is the category of algebras of the previous constructors, with $F(C, T, \epsilon, ext, s) :\equiv \Sigma\Gamma : C.\Sigma A : T(\Gamma).T(ext\,\Gamma\,A)$, and

$$G(C, T, \epsilon, ext, s, \Gamma, A, B) :\equiv ext\,(ext\,\Gamma\,A)\,B =_C ext\,\Gamma\,(s\,\Gamma\,A\,B).$$

Theorem 25 will show that G is relatively continuous.

Given a constructor specification, we can define a the corresponding category of algebras. In Theorem 27, we will see that the assumptions of Definition 12 guarantee that this category is complete.

Definition 15 (Category of algebras)**.** Let (F, G) be a constructor specification on a complete category $\mathcal{C}$. The *category of algebras* of (F, G) is denoted $\mathcal{C}.(F, G)$, and is defined as follows:

- objects are pairs (X, θ), where X is an object of $\mathcal{C}$, and $\theta : (x : FX) \to G(X, x)$ is a dependent function (in Set);
- morphisms $(X, \theta) \to (Y, \psi)$ are given by morphisms $f : X \to Y$ in $\mathcal{C}$, with the property that for all $x : FX$,

$$\psi(F(f)\, x) = G(\overline{f})(\theta\, x),$$

where $\overline{f} : (X, x) \to (Y, F(f)\, x)$ is the morphism in $\int^{\mathcal{C}} F$ determined by f.

We think of $\mathcal{C}.(F, G)$ as a category of "dependent dialgebras" [18]. Note that there is an obvious forgetful functor $\mathcal{C}.(F, G) \to \mathcal{C}$.

Example 16 (Permutable trees)**.** The category of algebras for the constructor specification (F_1, G_1) for node from Example 13 is equivalent to the category whose objects are triples (A, l, n) where $A : \mathsf{Set}$, $l : A$, and $n : (X \to A) \to A$. After specifying also the mix-constructor, the new category of algebras contains as well a dependent function $p : (f : A \to T) \to (\sigma : A \cong A) \to n(f) = n(f \circ \sigma)$.

Example 17 (Contexts and types)**.** Similarly, the category of algebras for the constructor specification from Example 14 has objects tuples $(C, T, e, c, b, s, s_{\mathrm{eq}})$ where (C, T, e, c, b, s) is an algebra for the previous constructors, and

$$s_{\mathrm{eq}} : (\Gamma : C) \to (A : T(\Gamma)) \to (B : T(c\,\Gamma\, A)) \to c\,(c\,\Gamma\, A)\, B =_C c\,\Gamma\,(s\,\Gamma\, A\, B).$$

3.2. **Point Constructors.** If $\mathcal{C}$ is the base category for a sort signature as in Definition 3, we can define specific target functors $\mathcal{C} \Rightarrow \mathsf{Set}$ which are guaranteed to be relatively continuous. Constructors having those as targets are referred to as *point constructors*. Intuitively, a point constructor is an operation that returns an element (point) of one of the sorts. The corresponding target functor is the forgetful functor that projects out the chosen sort. However, sorts can be dependent, so such a projection needs to be defined on a category of elements.

Specifically, let $\mathcal{C}$ be a finitely complete category, $H : \mathcal{C} \Rightarrow \mathsf{Set}$ a functor, and $\mathcal{C}'$ the extended base category with one more sort indexed over H. Recall that the objects of $\mathcal{C}'$ are pairs (X, P), where X is an object of $\mathcal{C}$, and P is a family of sets indexed over HX. Let $V_H : \mathcal{C}' \Rightarrow \mathcal{C}$ be the forgetful functor. We define the *base target* functor corresponding to H to be the functor $U_H : \int^{\mathcal{C}'} (H \circ V_H) \Rightarrow \mathsf{Set}$ given by

$$U_H(X, P, x) = P(x).$$

In other words, given an object X of $\mathcal{C}$, a family P over HX, and a point x in the base, the functor U_H returns the fibre of the family P over x. The action of U_H on morphisms is the obvious one.

Example 18 (Permutable trees)**.** In Example 13, the functor $G_0 : \int^{\mathsf{Set}} F_0 \Rightarrow \mathsf{Set}$ is the composition of the forgetful $\int^{\mathsf{Set}} F_0 \Rightarrow \mathsf{Set}$ with the base target functor for the only sort, which in this case is the identity $\mathsf{id} : \mathsf{Set} \Rightarrow \mathsf{Set}$.

Note that $U_H = \mathsf{id}$ in Example 18 is relatively continuous, as required by Definition 12. In the rest of this section, we will show that this is true in general. Given a category $\mathcal{C}$ and a functor $F : \mathcal{C} \Rightarrow \mathsf{Set}$, it is well known that the slice category over F of the functor category $\mathcal{C} \Rightarrow \mathsf{Set}$ is equivalent to the functor category $\int^{\mathcal{C}} F \Rightarrow \mathsf{Set}$. Given a functor $G : \mathcal{C} \Rightarrow \mathsf{Set}$ and a natural transformation $\alpha : G \to F$, we will refer to the functor $\overline{G} : \int^{\mathcal{C}} F \Rightarrow \mathsf{Set}$ corresponding to α as the *functor of fibres* of α. Concretely, $\overline{G}$ maps an object (X, x), where $x : FX$, to the fibre of α_X over x.

Lemma 19 (auxiliary, not listed in the main body)**.** *Let $\mathcal{C}$ be complete, $F, G : \mathcal{C} \to \mathsf{Set}$ functors, and $\alpha : G \to F$ a natural transformation, with functor of fibres $\overline{G}$. Then $\overline{G}$ is relatively continuous if and only if, for all small diagrams $X : \mathcal{I} \to \mathcal{C}$ and all limit cones $L \to X$ in $\mathcal{C}$, the following diagram*

$$\begin{array}{ccc} GL & \longrightarrow & \lim GX \\ \downarrow & & \downarrow \\ FL & \longrightarrow & \lim FX \end{array} \tag{1}$$

is a pullback.

Proof. A generic relative limit cone on $\mathcal{C}$ is determined by a limit cone $\pi : L \to X$, where $X : \mathcal{I} \to \mathcal{C}$ is any small diagram, and an element $z : FL$. Relative continuity of $\overline{G}$ is equivalent to the map

$$\overline{G}(L, z) \to \lim_i \overline{G}(X_i, F\pi_i z)$$

being an isomorphism for all such cones π and elements z. By the explicit description of the functor of fibres $\overline{G}$, and the fact that pullbacks commute with limits, we have the pullback squares

$$\begin{array}{ccc} \overline{G}(L, z) & \longrightarrow & GL \\ \downarrow & & \downarrow \\ 1 & \xrightarrow{z} & FL \end{array} \qquad \begin{array}{ccc} \lim_i \overline{G}(X_i, F\pi_i z) & \longrightarrow & \lim GX \\ \downarrow & & \downarrow \\ 1 & \longrightarrow & \lim FX. \end{array} \tag{2}$$

If we assume that (1) is a pullback, we can paste squares to get a pullback

$$\begin{array}{ccc} \overline{G}(L, z) & \longrightarrow & \lim GX \\ \downarrow & & \downarrow \\ 1 & \longrightarrow & \lim FX. \end{array}$$

By uniqueness of limits, it must be that $\overline{G}(L, z) \cong \lim_i \overline{G}(X_i, F\pi_i z)$, and the fact that the isomorphism is given by the canonical map follows from a straightforward diagram chase.

Conversely, if $\overline{G}$ is relatively continuous, it follows from the right square in (2) that the following diagram is a pullback

$$\begin{array}{ccc} \overline{G}(L, z) & \longrightarrow & \lim GX \\ \downarrow & & \downarrow \\ 1 & \longrightarrow & \lim FX. \end{array}$$

By taking a coproduct over FL and using extensivity of Set, we get the pullback square (1), as required. □

Theorem 20 (Base target functors are relatively continuous)**.** *Let $\mathcal{C}$ be a complete category, $H : \mathcal{C} \Rightarrow \mathsf{Set}$ any functor, and $\mathcal{C}'$ the extended base category corresponding to H. Then the base target functor U_H is relatively continuous.*

Proof. Let $\widetilde{U}_H : \mathcal{C}' \to \mathsf{Set}$ be the functor $\widetilde{U}_H(X, P) = (\Sigma x : HX)P(x)$. There is an obvious natural transformation $\theta : \widetilde{U}_H \to H \circ V_H$ given by the first projection. Clearly, U_H is the functor of fibres of θ, hence by Lemma 19 all we need to show is that θ maps limit cones in $\mathcal{C}'$ to pullback squares, which follows immediately from the construction of limits in $\mathcal{C}'$. □

3.3. **Reindexing Target Functors.** In many cases, we can obtain suitable target functors by composing the desired base target functor with the forgetful functor to the appropriate stage of the base category. When building constructors one at a time, it will follow from Theorem 27 and Theorem 10 applied to the previous steps that this forgetful functor is continuous, and the relative continuity of the target functor will follow. In more complicated examples, composing with a forgetful functor is not quite enough. We often want to "substitute into" or reindex a target functor to target a specific element. For example, in the context of Example 2, consider a hypothetical modified σ constructor of the form

$$\sigma' : \big(\Sigma\Gamma : \mathsf{Con}.\Sigma A : \mathsf{Ty}(\Gamma).\mathsf{Ty}(\mathsf{ext}\,\Gamma\,A)\big) \to \mathsf{Ty}(\mathsf{ext}\,\Gamma\,A).$$

We want the target functor to return the set $\mathsf{Ty}(\mathsf{ext}\,\Gamma\,A)$, and not just $\mathsf{Ty}(x)$ for a new argument x, which is the result of the base target functor. We can obtain the desired target functor as a composition

$$\textstyle\int^{\mathcal{C}} F \xrightarrow{S} \int^{\mathsf{Fam}(\mathsf{Set})} \pi_1 \xrightarrow{U_H} \mathsf{Set},$$

where $\mathcal{C}$ is the category with objects tuples (C, T, ϵ, ext), $F : \mathcal{C} \Rightarrow \mathsf{Set}$ is the functor giving the arguments of the constructor σ', U_H is the base target functor corresponding to the second sort, and S is the functor defined by $S(C, T, \epsilon, ext, \Gamma, A, B) :\equiv (C, T, ext\,\Gamma\,A)$.

Since the functors S that we want to compose with in order to "substitute" or reindex are of a special form, the resulting functor will still be relatively continuous when we start with a relatively continuous functor. This is made precise by the following result:

Lemma 21 (Preservation of relative limit cones)**.** *Suppose given a commutative diagram of categories and functors*

$$\begin{array}{ccc}
\mathcal{A} & \xrightarrow{F} & \mathcal{B} \\
{\scriptstyle U'}\downarrow & & \downarrow{\scriptstyle V'} \\
\mathcal{C} & \xrightarrow[G]{} & \mathcal{D} \\
{\scriptstyle U}\downarrow & & \downarrow{\scriptstyle V} \\
\mathcal{C}_0 & & \mathcal{D}_0,
\end{array}$$

where $\mathcal{C}_0$ and D_0 are complete, and G maps U-limit cones to V-limit cones. Then F maps $(U \circ U')$-limit cones to $(V \circ V')$-limit cones. In particular, if $\mathcal{C}$ and $\mathcal{D}$ are complete and G is continuous, then F preserves relative limit cones.

Proof. Immediate from the definition of relative limit cone. □

Example 22. In our example above, we have the following diagram:

$$\begin{array}{ccccc}
\textstyle\int^{\mathcal{C}} F & \xrightarrow{S} & \textstyle\int^{\mathsf{Fam}(\mathsf{Set})} \pi_1 & \xrightarrow{U_H} & \mathsf{Set} \\
\downarrow & & \downarrow & & \\
\mathcal{C} & \xrightarrow{V} & \mathsf{Fam}(\mathsf{Set}) & &
\end{array}$$

where $V : \mathcal{C} \Rightarrow \mathsf{Fam}(\mathsf{Set})$ is the forgetful functor, and hence continuous. It follows from the second statement of Lemma 21 that S preserves relative limit cones, hence $G = U_H \circ S$ is relatively continuous by Theorem 20.

3.4. **Path Constructors.** Path constructors are constructors where the target functor G returns an *equality* type. They can e.g. be used to express laws when constructing an initial algebra of an algebraic theory as a QIT. We saw an example of this in Example 1, where we had a path constructor of the form

$$\mathsf{mix} : (f : A \to T) \to (\sigma : A \cong A) \to \mathsf{node}(f) = \mathsf{node}(f \circ \sigma).$$

The argument functor for mix is entirely unproblematic. However, it is perhaps not so clear that the target functor, which sends (X, l, n, f, σ) to the equality type $n(f) =_X n(f \circ \sigma)$, is relatively continuous. The aim of the current section is to show this for any functor of this form. Our plan is to start with the prototypical example of such an equality functor, observe that it is relatively continuous, and then show that any other target functor that can occur in a path constructor can be obtained by substitution such that Lemma 21 is applicable.

Definition 23. Let $\mathsf{Eq} : \int^{\mathsf{Set}}(\mathsf{id} \times \mathsf{id}) \Rightarrow \mathsf{Set}$ be the functor defined on objects by $\mathsf{Eq}(X, x, y) :\equiv x =_X y$ and on morphisms by $\mathsf{Eq}(f, p_x, p_y) :\equiv p_x \centerdot (\mathsf{ap}\ f\ -) \centerdot p_y^{-1}$.

It is not hard to see that Eq is a functor. Furthermore, Eq is the functor of fibres of the obvious diagonal natural transformation $\Delta : \mathsf{id} \to \mathsf{id} \times \mathsf{id}$.

Lemma 24. *The standard equality functor is relatively continuous.*

Proof. Both id and $\mathsf{id} \times \mathsf{id}$ are representable, and in particular continuous. Therefore, the horizontal maps in the diagram of Lemma 19 are isomorphisms, which implies that the square is a pullback, hence Eq is relatively continuous. □

With the help of this lemma, one can prove that a large class of equality functors are suitable targets for constructors:

Theorem 25 (Equality functors are relatively continuous)**.** *Let $\mathcal{C}$ be a complete category, $F : \mathcal{C} \Rightarrow \mathsf{Set}$ any functor, and $G : \int^{\mathcal{C}} F \Rightarrow \mathsf{Set}$ a relatively continuous functor. Suppose given two global elements l, r of G, i.e. natural transformations $1 \to G$. The map*

$$\mathsf{Eq}_G(l, r) : \int^{\mathcal{C}} F \to \mathsf{Set},$$

defined by $\mathsf{Eq}_G(l, r)(X, x) = (l_{(X,x)} =_{G(X,x)} r_{(X,x)})$, extends to a relatively continuous functor.

Proof. Define $S : \int^{\mathcal{C}} F \Rightarrow \int^{\mathsf{Set}}(\mathsf{id} \times \mathsf{id})$ by $S(X, x) = (G(X, x), l_{(X,x)}, r_{(X,x)})$; this is a functor since l and r are natural. Observe that $\mathsf{Eq}_G(l, r)$ can now be obtained as the composition

$$\int^{\mathcal{C}} F \xrightarrow{S} \int^{\mathsf{Set}}(\mathsf{id} \times \mathsf{id}) \xrightarrow{\mathsf{Eq}} \mathsf{Set},$$

hence the conclusion of the lemma will follow once we establish that S preserves relative limit cones. Consider the diagram:

$$\begin{array}{ccc}
\int^{\mathcal{C}} F & \xrightarrow{S} & \int^{\mathsf{Set}} \mathsf{id} \times \mathsf{id} \\
{\scriptstyle \mathsf{id}}\downarrow & & \downarrow \\
\int^{\mathcal{C}} F & \xrightarrow{G} & \mathsf{Set} \\
\downarrow & & {\scriptstyle \mathsf{id}}\downarrow \\
\mathcal{C} & & \mathsf{Set},
\end{array}$$

which commutes by definition of S. Since G is relatively continuous by assumption, it preserves relative limit cones, hence so does S by Lemma 21, as required. □

Example 26 (Permutable trees)**.** The target of the mix constructor from Example 1 can be obtained as an equality functor in this sense. We take G to be the underlying sort, which is relatively continuous by the results of the previous section. The global elements l and r are defined by $l_{(X,l,n,f,\sigma)} :\equiv n(f)$ and $r_{(X,l,n,f,\sigma)} :\equiv n(f \circ \sigma)$. Their naturality can easily be verified directly.

Iterating equality functors, one can also express *higher* path constructors, but in our limited setting of inductively defined *sets*, there is little reason to go beyond one level of path constructors — higher ones will have no effect on the resulting inductive type. However, we believe that the ease with which Theorem 25 can be applied iteratively will be an important feature when generalising our technique to general higher inductive types. We discuss this further in Section 5.

3.5. **Categories of Algebras are Complete.** If $\mathcal{C}$ is a complete category, and (F, G) is a constructor specification on $\mathcal{C}$, recall that the category of algebras $\mathcal{C}.(F, G)$ from Definition 15 has "dependent (F, G)-dialgebras" as objects, and maps that commute with the dialgebra structures as morphisms. In this section, we will show that $\mathcal{C}.(F, G)$ is complete, and that its forgetful functor is continuous. The significance of this result is twofold:

First of all, it makes it possible to use the power of limits when reasoning about properties of algebras; in particular, we will show in Section 4 how, using products and equalisers, one can extend the classical equivalence between initiality and induction for ordinary inductive types to our setting.

Secondly, it goes a long way towards establishing an existence result for initial algebras; since a category of algebras over $n + 1$ constructors is complete, and the forgetful functor to the category of algebras over the first n preserves limits, it follows from the adjoint functor theorem that this functor has a left adjoint if and only if it satisfies the solution set condition. Since this can be applied to every stage, we get a left adjoint for the forgetful functor down to Set, and in particular an initial object. Given our assumptions on constructors, there is no reason to expect the solution set condition to hold at this generality. However, we expect it to follow from an appropriate "accessibility" condition on the argument functors. This is discussed further in Section 5.

Theorem 27 (Categories of algebras are complete)**.** *Let $\mathcal{C}$ be a complete category, and (F, G) a constructor specification on it. Then $\mathcal{C}.(F, G)$ is complete.*

Proof. Let $\widetilde{G} : \mathcal{C} \to \mathsf{Set}$ be defined by $\widetilde{G}(X) = (x : FX) \times G(X, x)$, and let $p : \widetilde{G} \to F$ be the first projection. Then clearly G is the functor of fibres of p. Now consider a diagram $Y : \mathcal{I} \to \mathcal{C}.(F, G)$. The diagram Y can be decomposed into a diagram $X : \mathcal{I} \to \mathcal{C}$, and a natural transformation $s : FX \to \widetilde{G}X$ which is a section of pX. If $\pi : L \to X$ is a limit cone for X, by Lemma 19 we get a pullback square

$$\begin{array}{ccc} \widetilde{G}L & \longrightarrow & \lim \widetilde{G}X \\ \downarrow & & \downarrow \\ FL & \longrightarrow & \lim FX, \end{array}$$

and s determines a section $\lim s : \lim FX \to \lim \widetilde{G}X$ of the right vertical morphism in the diagram. Let $t : FL \to \widetilde{G}L$ be the section of p_L obtained by pulling back $\lim s$. In particular, t has the form $\langle \mathsf{id}, \theta \rangle$, where

$$\theta : (x : FX) \to G(X, x),$$

and π extends to a cone $(L, \theta) \to Y$ in $\mathcal{C}.(F, G)$. It is now easy to verify that this is a limit cone. □

4. Elimination Principles

So far, we have given rules for specifying a QIIT by giving a sort signature and a list of constructors. As type-theoretical rules, these correspond to the formation and introduction rules for the QIIT. In this section, we introduce he corresponding elimination rules, stating that a QIIT is the smallest type closed under its constructors. We show that a categorical formulation of the elimination rules is equivalent to the universal property of initiality.

4.1. **The Section Induction Principle.** The elimination principle for an algebra X states that *every fibred algebra over X has a section*, where a fibred algebra over X is an algebra family "$Q : X \to \mathsf{Set}$", and a section of it a dependent algebra morphism "$(x : X) \to Q(x)$". The usual correspondence between type families and fibrations extends to algebras (see the examples below), and so we formulate the elimination rule for X as X being section inductive in the category of algebras in the following sense:

Definition 28 (Section inductive)**.** An object X of a category $\mathcal{C}$ is *section inductive* if for every object Y of $\mathcal{C}$ and morphism $p : Y \to X$, there exists $s : X \to Y$ such that $p \circ s = \mathsf{id}_X$.

For an algebra X, the existence of the underlying function(s) $X \to Y$ corresponds to the elimination rules, while the fact that they are algebra morphisms corresponds to the computation rules.

Example 29 (Permutable trees)**.** Consider permutable-tree algebras, e.g. tuples (X, l, n, p) as in Example 16. A fibred permutable-tree algebra over (X, l, n, p) consists of $Q : X \to \mathsf{Set}$ together with $m_l : Q(l)$ and

$$\begin{aligned} m_n &: (f : A \to X) \to (g : (a : A) \to Q(f\,a)) \to Q(n\,f) \\ m_p &: (f : A \to X) \to (g : (a : A) \to Q(f\,a)) \to (\sigma : A \cong A) \\ &\qquad \to m_n\,f\,g \;=\!\![\mathsf{ap}\;Q\;p]\; m_n\,(f \circ \sigma)\,(g \circ \sigma) \end{aligned}$$

Here the type $x \;=\!\![p]\; y$ is the types of equalities between elements $x : A$ and $y : B$ in different types, themselves related by an equality proof $p : A = B$. This data can be arranged into an ordinary algebra $\Sigma x : X.Q(x)$, together with an algebra morphism $\pi_1 : (\Sigma x : X)Q(x) \to X$. A section of this morphism is exactly a dependent function $h : (x : X) \to Q(x)$. Since h comes from an algebra morphism, we further know that e.g. $h(l) = m_l$ and $h(n\,f) = m_n\,f\,(h \circ f)$. Conversely, if we start with an algebra morphism $g : (X', l', n', p') \to (X, l, n, p)$, this gives rise to a fibred algebra (Q, m_l, m_n, m_p) by considering the fibres $Q(x) = \Sigma y : A'.g(y) = x$ of p. The points m_l, m_n and the path m_p arise from the proof that g preserves a', b' and p'.

Example 30 (Contexts and types)**.** For context-and-types algebras from Example 17, a fibred algebra over $(C, T, e, c, b, s, s_{\mathrm{eq}})$ consists of $Q : C \to \mathsf{Set}$ and $R : (x : C) \to T(x) \to Q(x) \to \mathsf{Set}$, together with $m_e : Q(e)$ and

$$\begin{aligned} m_c &: \;(\Gamma : C) \to (x : Q(\Gamma)) \to (A : T(\Gamma)) \to R(\Gamma, A, x) \to Q(c\,\Gamma\,A) \\ m_b &: \;(\Gamma : C) \to (x : Q(\Gamma)) \to R(\Gamma, b\,\Gamma, x) \\ m_s &: \;(\Gamma : C) \to (x : Q(\Gamma)) \to (A : T(\Gamma)) \to (y : R(\Gamma, A, x) \to (B : T(c\,\Gamma\,A)) \\ &\qquad \to (z : R(c\,\Gamma\,A, B, m_c\,\Gamma\,x\,A\,y)) \to R(\Gamma, s\,\Gamma\,A\,B, x) \\ m_{s_{\mathrm{eq}}} &: (\Gamma : C) \to (x : Q(\Gamma)) \to (A : T(\Gamma)) \to (y : R(\Gamma, A, x)) \\ &\qquad \to (B : T(c\,\Gamma\,A)) \to (z : R(c\,\Gamma\,A, B, m_c\,\Gamma\,x\,A\,y)) \\ &\qquad \to m_c\,(c\,\Gamma\,A)\,(m_c\,\Gamma\,x\,A\,y)\,B\,z \;=\!\![\mathsf{ap}\;Q\;(s_{\mathrm{eq}}\,\Gamma\,A\,B)] \\ &\qquad\qquad m_c\,\Gamma\,x\,(s\,\Gamma\,A\,B)\,(m_s\,\Gamma\,x\,A\,y\,B\,z) \end{aligned}$$

Again, this data can be arranged into an ordinary algebra with base $C' : \mathsf{Set}$, $T' : C' \to \mathsf{Set}$, where $C' = \Sigma x : C.Q(x)$ and $T'(x, q) = \Sigma y : T(x).R(x, y, q)$, together with an algebra morphism $(\pi_1, \pi_1) : (C', T') \to (C, T)$. A section of this morphism gives functions $f : (x : C) \to Q(x)$ and $g : (x : C) \to (y : T(x)) \to R(x, y, f\, x)$ that preserve the algebra structure.

A general account of the equivalence between the usual formulation of the elimination rules and the section induction principle is in Dijkstra [14, Section 5.4].

4.2. **Initiality, and its Relation to the Section Induction Principle.** The section induction principle for an algebra X matches our intuitive understanding of the elimination rules for X, but it is perhaps a priori not so clear that e.g. satisfying it defines an algebra uniquely up to equivalence. In this section, we show that this is the case by proving that the section induction principle is equivalent to the categorical property of initiality.

Definition 31 (Initiality)**.** An object X of a category $\mathcal{C}$ is initial if for every object Y of $\mathcal{C}$, the set of morphisms $X \to Y$ is contractible.

It is immediate that the property of being initial is a mere proposition. It is also more or less obvious that initiality implies section induction:

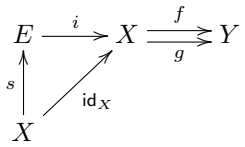
Lemma 32. *If an object X in a category $\mathcal{C}$ is initial, then it is section inductive.*

Proof. Assume X is initial. Given $p : Y \to X$, we need to produce $s : X \to Y$ such that $p \circ s = \mathsf{id}_X$. Since X is initial, there is an arrow $s : X \to Y$. Further $p \circ s : X \to X$, so by uniqueness of morphisms $X \to X$, we have $p \circ s = \mathsf{id}_X$. □

For the converse, a little bit more structure in $\mathcal{C}$ is needed:

Lemma 33. *If an object X in a category $\mathcal{C}$ with finite limits is section inductive, then it is initial.*

Proof. Given an object Y in $\mathcal{C}$, we need to provide a unique arrow $X \to Y$. Consider the projection $\pi_1 : X \times Y \to X$, which is an arrow into X, and therefore has a section $s : X \to X \times Y$. Our candidate arrow is then $\pi_2 \circ s : X \to Y$, which we have to show is unique. Using equalisers, we can show that any two arrows f, g out of X to some other object Y are equal. Let E be the equaliser of f and g, then we get a projection map $i : E \to X$. By the section principle, this map has a section $s : X \to E$:



Hence $f = \mathsf{id}_X \circ f = s \circ i \circ f = s \circ i \circ g = \mathsf{id}_X \circ g = g$ holds. □

Using all these ingredients, we get the main theorem of this section:

Theorem 34 (Initiality $\cong$ section induction)**.** *An object X in a in a category of algebras $\mathcal{C}.(F, G)$ being initial is equivalent to it being section inductive.*

Proof. By Theorem 27, the category $\mathcal{C}.(F, G)$ is complete. Hence by Lemmas 32 and 33, the two statements are logically equivalent. Since a logical equivalence between two mere propositions is automatically an equivalence, and initiality is easily seen to be a mere proposition, all that remains is to show that the section induction property is a mere proposition. For this, we may assume that the type in question is inhabited, and it suffices to show that the set of sections $\big(\Sigma s : X \to Y\big)\big(p \circ s = \mathsf{id}_X\big)$ is a mere proposition for any object Y. But since X is initial by assumption and

Lemma 33, the sets $X \to Y$ and $X \to X$ are contractible, hence so is the path type $p \circ s =_{X \to X} \mathsf{id}_X$, and we are done. □

As an application, we can now reason about QIITs using their categories of algebras. For instance, we get a short proof of the following fact:

Corollary 35. *The interval is equivalent to the unit type.*

Proof. By Theorem 34, the interval is the initial object in the category with objects $\Sigma X : \mathsf{Set}.\Sigma x : X.\Sigma y : X.x =_X y$, while the unit type is the initial object in the category with objects $\Sigma X : \mathsf{Set}.X$. By singleton contractibility, the former is equivalent to the latter, and since initiality is a universal property, the two initial objects coincide up to equivalence. □

5. Conclusions and Further Work

We have developed a semantic framework for QIITs: QIITS give rise to a category of algebras and the initial object of this category represent the types and constructors of the QIIT. This generalises the usual functorial semantics of inductive types to this much more general setting. So far we have verified the appropriateness of this setting by means of examples. In future work, we would like to explicitly relate the syntax of QIITs to the corresponding semantics.

Our category of algebras is complete. This is helpful when developing a metatheory of QIITs, as demonstrated by the proof of equivalence of initiality and section induction (Theorem 34), justifying elimination principles. Of course, completeness is not by itself sufficient to derive the existence of initial algebras, but it suggests that it should be possible to restrict the argument functors enough to guarantee this, possibly by reducing the existence of QIITs to some basic type former playing an analogous role to that of W-types for ordinary inductive types. We believe that completeness of the categories of algebras will allow an existence proof using the adjoint functor theorem.

We have restricted our attention to QIITs, but we believe that our construction is applicable to general HITs (and even HIITs). While at first glance such an extension of our framework seems to require an internal theory of $(\infty, 1)$-categories, we believe that it is enough to keep track of only a very limited number of coherence conditions, making this extension possible even without solving the well-known problem of specifying an infinite tower of coherences in HoTT.

There are other directions of future work one may consider, e.g. the combination of QIITS and induction-recursion, and the possibility of generalising coinductive types along similar lines. In any case these generalisations should be driven by examples, similar to how the examples discussed in the current paper have motivated the need for QIITs.

Acknowledgements. We thank Ambrus Kaposi and Jakob von Raumer for many interesting discussions. This research was supported by EPSRC grants EP/M016994/1 and EP/K023837/1, as well as AFOSR award FA9550-16-1-0029.

References

1. Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Principles of Programming Languages*. ACM, 2016.
2. Thorsten Altenkirch, Nils Anders Danielsson, and Nicolai Kraus. Partiality, revisited. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures*, pages 534–549. Springer, 2017.
3. Carlo Angiuli, Edward Morehouse, Daniel R. Licata, and Robert Harper. Homotopical patch theory. In *International Conference on Functional Programming*, pages 243–256, 2014.

4. Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, 2009.
5. Henning Basold, Herman Geuvers, and Niels van der Weide. Higher inductive types in programming. *Journal of Universal Computer Science*, 23(1):63–88, 2016.
6. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, 2004.
7. Edwin Brady. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, 23:552–593, 9 2013.
8. Guillaume Brunerie. *On the homotopy groups of spheres in homotopy type theory*. PhD thesis, Universitè de Nice, 2016.
9. Ulrik Buchholtz and Egbert Rijke. The real projective spaces in homotopy type theory. In *Logic in Computer Science*, pages 1–8, 2017.
10. Evan Cavallo. Synthetic cohomology in Homotopy Type Theory. Master's thesis, Carnegie-Mellon University, 2015.
11. James Chapman, Tarmo Uustalu, and Niccolò Veltri. Quotienting the delay monad by weak bisimilarity. In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *International Colloquium on Theoretical Aspects of Computing*, volume 9399 of *LNCS*, pages 110–125. Springer, 2015.
12. Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover. In *Conference on Automated Deduction*, 2015.
13. Radu Diaconescu. Axiom of choice and complementation. *Proceedings of the American Mathematical Society*, 51(1):176–178, 1975.
14. Gabe Dijkstra. *Quotient inductive-inductive types*. PhD thesis, University of Nottingham, 2017.
15. Peter Dybjer. Inductive families. *Formal aspects of computing*, 6(4):440–465, 1994.
16. Peter Dybjer and Hugo Moeneclaey. Finitary higher inductive types in the groupoid model. In Alexandra Silva, editor, *Mathematical Foundations of Programming Semantics*, 2017.
17. Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *Typed lambda calculi and applications*, pages 129–146. Springer, 1999.
18. Tatsuya Hagino. *A Categorical Programming Language*. PhD thesis, University of Edinburgh, 1987.
19. Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford University Press, 1998.
20. Kuen-Bang Hou (Favonia), Eric Finster, Daniel R. Licata, and Peter LeFanu Lumsdaine. A mechanization of the Blakers-Massey connectivity theorem in Homotopy Type Theory. In *Logic in Computer Science*, 2016.
21. Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after Voevodsky), 2016. arXiv:1211.2851.
22. Daniel R. Licata and Eric Finster. Eilenberg-Maclane spaces in homotopy type theory. In *Logic in Computer Science*, pages 66:1–66:9, 2014.
23. Daniel R. Licata and Michael Shulman. Calculating the fundamental group of the circle in homotopy type theory. In *Logic in Computer Science*, pages 223–232, 2013.
24. Peter LeFanu Lumsdaine and Mike Shulman. Semantics of higher inductive types, 2017. arXiv:1705.07088.
25. Per Martin-Löf. An intuitionistic theory of types. Published in Twenty-Five Years of Constructive Type Theory, 1972.
26. Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.
27. Michael Shulman. Homotopy type theory: the logic of space, 2017. To appear in *New Spaces for Mathematics and Physics*. arXiv:1703.03007.
28. Kristina Sojakova. Higher inductive types as homotopy-initial algebras. In Sriram K. Rajamani and David Walker, editors, *Principles of Programming Languages*, pages 31–42. ACM, 2015.

29. The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.
30. Vladimir Voevodsky. The equivalence axiom and univalent models of type theory (talk at CMU on February 4, 2010), 2010. arXiv:1402.5556.