Quotient Inductive-Inductive Types

Thorsten Altenkirch[1], Paolo Capriotti[1], Gabe Dijkstra, Nicolai Kraus[1], and Fredrik Nordvall Forsberg[2]

[1] University of Nottingham
[2] University of Strathclyde

Abstract. Higher inductive types (HITs) in Homotopy Type Theory allow the definition of datatypes which have constructors for equalities over the defined type. HITs generalise quotient types, and allow to define types with non-trivial higher equality types, such as spheres, suspensions and the torus. However, there are also interesting uses of HITs to define types satisfying uniqueness of equality proofs, 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 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. We further derive a *section induction principle*, stating that every algebra morphism into the algebra in question has a section, which is close to the intuitively expected elimination rules.

1 Introduction

This paper is about type theory in the sense of Martin-Löf [29], a theory which proof assistants such as Coq [7] and Lean [14] as well as programming languages such as Agda [31] and Idris [8] are based on. Recently, Homotopy type theory (HoTT) [34] has been introduced inspired by homotopy theoretic interpretations of type theory by Awodey and Warren [5] and Voevodsky [25; 36].

A central concept in type theory is the concept of inductive definitions, which allows us to define inductive datatypes like the natural numbers, lists and trees just by presenting constructors with strictly positive occurrences of the inductive type being defined. Using the propositions as types explanation, we can use the same mechanism to inductively define predicates and relations, like an order on the natural numbers, 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 proofs (paths). Hence an inductive definition may not only feature constructors for elements but also for equalities. This concept of higher inductive types (HITs) has been used to represent the homotopical structure of geometric objects, like circles, spheres and tori, and gives rise to synthetic homotopy theory in HoTT [32].

However, as already noted in the HoTT Book [34], HITs have also more quotidian applications, such as a definition of the Cauchy reals for which the use of the axiom of choice can be avoided when proving e.g. Cauchy completeness. Instead of defining the real numbers as a quotient of sequences of rationals, a HIT is used to define them as the Cauchy completion of the rational numbers, with the quotienting happening simultaneously with the completion definition. Similarly, a definition of the partiality monad, which represents potentially diverging operations over a given type, was given using a HIT [2; 13; 35], again avoiding the axiom of choice when showing e.g. that the construction is a monad [12].

As we see from these examples, 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 it. For example: consider trees branching over an arbitrary type A, quotiented by arbitrary permutations of subtrees. We first define the type $T_0(A)$ of A-branching trees, given by the constructors

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

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

Now, morally, from a family $A \to (T_0(A)/R)$, we should be able to construct an element of the quotient $T_0(A)/R$. This is indeed possible if A is $\mathbf{2}$ or another finite type, by applying the induction principle of the quotient type A times. However, it seems that, for a general type A, this would require the axiom of choice [34], which unfortunately is not a constructive principle [15]. 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 $T(A) : \mathsf{hSet}$ by

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

Note that the fact that $T(A)$ is a homotopy set (see *preliminaries* below) is implicitly included in the statement $T(A) : \mathsf{hSet}$. 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* [30] 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 $\mathsf{Con} : \mathsf{hSet}$ and a type family $\mathsf{Ty} : \mathsf{Con} \to \mathsf{hSet}$ are simultaneously defined by giving the constructors

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

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 $\sigma\,\Gamma\,A\,B$. 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 making use of concrete HITs [27; 26; 4; 11; 23; 9; 10], 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 [33; 6; 18; 28], and they do indeed give a satisfactory specification of HITs in the sense that they cover all HITs which have been used so far (see *related work* below). However, to the best of our knowledge, no approach covers *higher inductive-inductive* definitions such as Example 2. The purpose of the current paper is to remedy this. We restrict ourselves to sets, i.e. to *quotient inductive-inductive types* (QIITs). This is of course a serious restriction, since it means that we cannot capture many ordinary HITs such as e.g. 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 framework, which 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 quotient inductive-inductive types. A QIIT is specified by (i) its *sorts*, which encode the types and type families that it consists of (Section 2), and (ii) by a sequence of *constructors*, that in turn are specified by *argument* and *target functors* (Section 3). This is a very general framework, covering in particular point (Section 3.2) and path constructors (Section 3.4). Each constructor specification gives rise to a category of algebras, and we establish conditions on the target functors that allow us to conclude that these categories of algebras are complete (Section 3.5). This is important, because it allows us to prove the equivalence of initiality and a principle that we call *section induction* (Section 4), stating that every algebra morphism into the algebra in question has a section; this principle is close to the intuitively expected elimination rules.

A full version of the paper, including all proofs, is available on the arXiv [3].

Related Work Sojakova [33] shows the correspondence between initiality and induction (a variant of our Theorem 31) for W-suspensions, a restricted class of HITs. Basold, Geuvers and van der Weide [6] introduce a syntactic schema for HITs without higher path constructors, and derive their elimination rules. Dybjer and Moeneclaey [18] give a syntactic schema for finitary HITs with at most paths between paths, and give an interpretation in Hofmann and Streicher's groupoid model [22]. Finally, Lumsdaine and Shulman's work on the semantics of HITs in model categories [28] is similar to an external version of our approach.

Preliminaries We work in a standard Martin-Löf style type theory and assume function extensionality. We do not assume univalence, but also do not contradict it; in particular, everything we do works in the type theory from the HoTT Book [34]. We write $\mathcal{U}$ for "the" universe of types, omitting universe indices in the *typical ambiguity* style [21]. A type is a set if all its equality proofs are equal, and hSet is defined as $\Sigma(A : \mathcal{U}).\mathsf{is\text{-}set}(A)$; we implicitly treat elements of hSet as their first projections — this allows us to view hSet as a universe. By a *category*, we mean a precategory [34, Def 9.1.1] in the sense of the HoTT Book (all our categories become univalent categories if univalence is assumed). We write $\mathcal{C} \Rightarrow \mathcal{D}$ for functors and $X \to Y$ for functions between types. We denote the obvious category of sets and functions by hSet as well; consequently, $F : A \to \mathsf{hSet}$ denotes a type family, while $F : \mathcal{C} \Rightarrow \mathsf{hSet}$ denotes a functor. For such a functor $F : \mathcal{C} \Rightarrow \mathsf{hSet}$, we write $\int^{\mathcal{C}} F$ for the *category of elements* of F, whose objects are 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 "action of a function to paths", ${}^{-1} : x = y \to y = x$ for "path reversal", and $\cdot : x = y \to y = z \to x = z$ for "path concatenation" [34, Lem 2.2.1, 2.1.1, 2.1.2].

2 Sorts

Single inductive (and quotient inductive) sets are simply elements of hSet. Inductive families [17] indexed over some fixed type A are families $A \to \mathsf{hSet}$.

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 $A : B \to \mathsf{hSet}$ mutually with a family $B : A \to \mathsf{hSet}$ (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 used as family indices 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 (Sort specifications). *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 $H_i : \mathcal{C}_i \Rightarrow \mathsf{hSet}$. Here, $\mathcal{C}_0 :\equiv \mathbb{1}$ 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 $P : H_i(X) \to \mathsf{hSet}$ 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 hSet).*

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 $H_0 : \mathcal{C}_0 \Rightarrow \mathsf{hSet}$ 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 $W : \mathbf{1} \to \mathsf{hSet}$, and morphisms are given by $f : \star \to \star$ in $\mathbb{1}$ (necessarily the identity morphism), together with a dependent function $g : (x : \mathbf{1}) \to W(x) \to V(x)$. It is easy to see that this category $\mathcal{C}_1$ is equivalent to the category hSet.

Example 5 (The finite types). Consider the inductive family $\mathsf{Fin} : \mathbb{N} \to \mathsf{hSet}$ 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 $X : \mathbb{N} \to \mathsf{hSet}$ 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 $H_1 : \mathcal{C}_1 \Rightarrow \mathsf{hSet}$. Applying the equivalence between $\mathcal{C}_1$ and hSet established in Example 4, we define H_1 to be the identity functor $H_1(A) :\equiv A$, since then $\mathsf{Ty} : H_1(\mathsf{Con}) \to \mathsf{hSet}$. The base category $\mathcal{C}_2$ is equivalent to the category $\mathsf{Fam}(\mathsf{hSet})$, whose objects are pairs (A, B) where $A : \mathsf{hSet}$ and $B : A \to \mathsf{hSet}$, 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 [34] are constructed by simultaneously defining $\mathbb{R} : \mathsf{hSet}$ and $\sim : \mathbb{R} \times \mathbb{R} \to \mathsf{hSet}$ (we ignore the fact that [34] uses $\mathcal{U}$ instead of hSet). 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 a binary relation on $\mathbb{R}$. The base category has (up to equivalence) pairs (X, Y) with $Y : X \times X \to \mathsf{hSet}$ 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:

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

We have:

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

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 [19] of the list $H_0, \ldots, H_n$ simultaneously with a function that turns such a list into a category. Details can be found in Dijkstra's thesis [16, Sec 4.3].

The main result of this section states that base categories of sort signatures are complete, i.e. have 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 the collection of morphisms between any two objects is a set. This result will be needed later to show that categories of QIIT algebras are complete. Recall that hSet has all small limits by a standard construction.

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

Proof. All proofs can be found in the arXiv version of the paper [3]. □

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 inductively in Section 2, we construct suitable categories of algebras by starting with a finitely complete category $\mathcal{C}$ such as the one obtained from a sort signature, specify a constructor on $\mathcal{C}$, and then extend $\mathcal{C}$ using this constructor specification to get a new finitely complete category $\mathcal{C}'$. This process is repeated until all constructors have been added, and 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 hSet 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 category of algebras at this stage is the category of 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 hSet-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 hSet. The continuity condition on G is needed for the corresponding category of algebras to be complete. Intuitively, this expresses that a constructor should only "construct" elements of one of the sorts, or equalities thereof.[3] 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.

[3] More concretely, elements of a sort correspond to representable functors for algebras over a single generator for that sort, while equalities correspond to algebras with no generators and the given equality as the only relation. Clearly, representable functors are continuous, and the converse holds for reasonable functors (e.g. accessible ones). However, we do not attempt to make this construction precise here, and the following results do not depend on it.

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. If I is a small category, and $X : I \to \mathcal{C}$ is a diagram, we say that a cone $A \to X$ in $\mathcal{C}$ is a U-*limit cone*, or* limit cone relative to U*, if the induced cone $UA \to UX$ is a limit cone in $\mathcal{C}_0$. A functor $\mathcal{C} \Rightarrow \mathsf{hSet}$ is* continuous relative to U *if it maps U-limit cones to limit cones in hSet.*

In the special case $\mathcal{C}_0 \equiv \mathsf{hSet}$, the functor U in Definition 11 is continuous relative to itself. Also note that 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 $F : \mathcal{C} \Rightarrow \mathsf{hSet}$ 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 $F : \mathcal{C} \Rightarrow \mathsf{hSet}$, called the* argument *functor of the specification;*
- *a relatively continuous functor $G : \int^{\mathcal{C}} F \Rightarrow \mathsf{hSet}$, called the* target *functor.*

Given a constructor specification, we can define the corresponding category of algebras. In Theorem 25, we will see that the assumptions of Definition 12 guarantee that this category is complete.

Definition 13 (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 hSet);*
- *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" [20]. Note that there is an obvious forgetful functor $\mathcal{C}.(F, G) \to \mathcal{C}$.

Similarly to how we defined sort specifications (Definition 3), we now have all the necessary notions in place to be able to give the full definition of a QIIT.

Definition 14 (QIIT descriptions). *A* QIIT description *is given by*

- *a sort specification $H_0, \ldots, H_{n-1}$;*

– *a list of constructor specifications* $(F_0, G_0), \dots, (F_{n-1}, G_{n-1})$ *on* $\mathcal{B}_0, \dots, \mathcal{B}_{n-1}$ *respectively, where* $\mathcal{B}_0$ *is the base category of the given sort specification, and* $\mathcal{B}_{i+1}$ *is the category of algebras of* (F_i, G_i)*.*

For Definition 14 to make sense, the categories $\mathcal{B}_i$ need to be complete, since constructor specifications are only defined on complete categories. This will follow from Theorem 25.

Example 15 (Permutable trees). The constructor $\mathsf{leaf} : T(A)$ from Example 1 can be specified by functors $F_0 : \mathsf{hSet} \Rightarrow \mathsf{hSet}$ and $G_0 : \int^{\mathsf{hSet}} F_0 \Rightarrow \mathsf{hSet}$, where $F_0(X) :\equiv \mathbf{1}$ and $G_0(X, l) :\equiv X$. 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 $F_1 : \mathsf{hSet}_\bullet \Rightarrow \mathsf{hSet}$ and $G_1 : \int^{\mathsf{hSet}_\bullet} F_1 \Rightarrow \mathsf{hSet}$, where $\mathsf{hSet}_\bullet$ 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 18 will show that G_0 and G_1 are relatively continuous.

The corresponding category of algebras for this constructor specification (F_1, G_1) for node is equivalent to the category whose objects are triples (X, l, n) where $X : \mathsf{hSet}$, $l : A$, and $n : (A \to X) \to X$. After specifying also the mix-constructor, the new category of algebras further contains a dependent function $p : (f : A \to X) \to (e : X \cong X) \to n(f) = n(f \circ e)$.

Example 16 (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 $F : \mathcal{C} \Rightarrow \mathsf{hSet}$ and $G : \int^{\mathcal{C}} F \Rightarrow \mathsf{hSet}$, where $\mathcal{C}$ is the category of algebras of the previous constructors, with

$$F(C, T, \varepsilon, \mathsf{ext}, \sigma) :\equiv \Sigma(\Gamma : C).\Sigma(A : T(\Gamma)).T(\mathsf{ext}, \Gamma\,A)$$

and

$$G(C, T, \varepsilon, \mathsf{ext}, \sigma, \Gamma, A, B) :\equiv \mathsf{ext}\,(\mathsf{ext}\,\Gamma\,A)\,B =_C \mathsf{ext}\,\Gamma\,(\sigma\,\Gamma\,A\,B).$$

Theorem 23 will show that G is relatively continuous. The corresponding category of algebras for this constructor specification 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 $\mathcal{C} \Rightarrow \mathsf{hSet}$ 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, $H : \mathcal{C} \Rightarrow \mathsf{hSet}$ a functor, and $\mathcal{C}'$ the extended base category with one more sort indexed over H. Recall from Definition 13 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 $U_H : \int^{\mathcal{C}'}(H \circ V_H) \Rightarrow \mathsf{hSet}$ 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 17 (Permutable trees). In Example 15, the functor $G_0 : \int^{\mathsf{hSet}} F_0 \Rightarrow \mathsf{hSet}$ specifying the target of leaf is the composition of the forgetful $\int^{\mathsf{hSet}} F_0 \Rightarrow \mathsf{hSet}$ with the base target functor for the only sort, in this case the identity $\mathsf{id} : \mathsf{hSet} \Rightarrow \mathsf{hSet}$.

Note that $U_H = \mathsf{id}$ in Example 17 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 $F : \mathcal{C} \Rightarrow \mathsf{hSet}$, it is well known that the slice category over F of the functor category $\mathcal{C} \Rightarrow \mathsf{hSet}$ is equivalent to the functor category $\int^{\mathcal{C}} F \Rightarrow \mathsf{hSet}$ (see for example [24, Prop 1.1.7]). Given a functor $G : \mathcal{C} \Rightarrow \mathsf{hSet}$ and a natural transformation $\alpha : G \to F$, we will refer to the functor $\overline{G} : \int^{\mathcal{C}} F \Rightarrow \mathsf{hSet}$ 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. The following theorem is proved by noting that U_H is a functor of fibres.

Theorem 18 (Base target functors are relatively continuous). *Let $\mathcal{C}$ be a complete category, $H : \mathcal{C} \Rightarrow \mathsf{hSet}$ any functor, and $\mathcal{C}'$ the extended base category corresponding to H. Then the base target functor U_H is relatively continuous.* □

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 25 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

$$\int^{\mathcal{C}} F \xrightarrow{S} \int^{\mathsf{Fam(hSet)}} \pi_1 \xrightarrow{U_H} \mathsf{hSet}, \tag{1}$$

where $\mathcal{C}$ is the category with objects tuples $(C, T, \varepsilon, \mathsf{ext})$, $F : \mathcal{C} \Rightarrow \mathsf{hSet}$ 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 $S(C, T, \varepsilon, \mathsf{ext}, \Gamma, A, B) :\equiv (C, T, \mathsf{ext}\,\Gamma\,A)$.

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

Lemma 19 (Preservation of relative limit cones). *Suppose given is a commutative diagram of categories and functors as shown on the right, where $\mathcal{C}_0$ and $\mathcal{D}_0$ 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.* □

$$\begin{tikzcd}
\mathcal{A} \arrow[r, "F"] \arrow[d, "U'"'] & \mathcal{B} \arrow[d, "V'"] \\
\mathcal{C} \arrow[r, "G"'] \arrow[d, "U"'] & \mathcal{D} \arrow[d, "V"] \\
\mathcal{C}_0 & \mathcal{D}_0
\end{tikzcd}$$

$$\begin{tikzcd}
\int^{\mathcal{C}} F \arrow[r, "S"] \arrow[d] & \int^{\mathsf{Fam(hSet)}} \pi_1 \arrow[r, "U_H"] \arrow[d] & \mathsf{hSet} \\
\mathcal{C} \arrow[r, "V"] & \mathsf{Fam(hSet)} &
\end{tikzcd}$$

Example 20. Starting from the situation in (1) we can form the diagram shown on the left, where $V : \mathcal{C} \Rightarrow \mathsf{Fam(hSet)}$ is the forgetful functor and hence continuous. It follows from the second statement of Lemma 19 that S preserves relative limit cones, hence $G = U_H \circ S$ is relatively continuous by Theorem 18.

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

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

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, e) to the equality type $n(f) =_X n(f \circ e)$, is relatively continuous. The aim of the current section is to show this for any functor of this form. We first observe that the prototypical such equality functor is relatively continuous, and then show that any other target functor for a path constructor can be obtained by substitution using Lemma 19.

Definition 21. *Let* $\mathsf{Eq} : \int^{\mathsf{hSet}}(\mathsf{id} \times \mathsf{id}) \Rightarrow \mathsf{hSet}$ *be the functor defined on objects by* $\mathsf{Eq}(X, x, y) :\equiv x =_X y$ *and on morphisms by* $\mathsf{Eq}(f, p_x, p_y) :\equiv p_x \cdot (\mathsf{ap}\ f\ -) \cdot p_y^{-1}$.

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

Lemma 22. *The standard equality functor* Eq *is relatively continuous.* □

The lemma we have just given is central to the observation that a large class of equality functors are suitable targets for constructors:

Theorem 23 (Equality functors are relatively continuous). *Let $\mathcal{C}$ be a complete category, $F : \mathcal{C} \Rightarrow \mathsf{hSet}$ any functor, and $G : \int^{\mathcal{C}} F \Rightarrow \mathsf{hSet}$ a relatively continuous functor. Suppose given two global elements l, r of G, i.e. natural transformations $l, r : \mathbf{1} \to G$. The map*

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

with $\mathsf{Eq}_G(l, r)(Y) = (l_Y =_{G(Y)} r_Y)$ extends to a relatively continuous functor. □

Example 24 (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 $l_{(X,l,n,f,e)} :\equiv n(f)$ and $r_{(X,l,n,f,e)} :\equiv n(f \circ e)$. 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 23 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

Recall from Definition 13 that the category of algebras $\mathcal{C}.(F, G)$ for a constructor specification (F, G) on a complete category $\mathcal{C}$ has "dependent (F, G)-dialgebras" as objects, and maps that commute with the dialgebra structure 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 enables the use of limits when reasoning about 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 existence of 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, the adjoint functor theorem says that this functor has a left adjoint if and only if it satisfies the solution set condition. Applying this argument at every stage, we get a left adjoint for the forgetful functor down to hSet, and in particular an initial object. There is no reason to expect the solution set condition to hold at this generality, but we expect it to follow from appropriate "accessibility" conditions on the argument functors. This is discussed further in Section 5.

Theorem 25 (Categories of algebras are complete). *Let (F, G) be a constructor specification on a complete category $\mathcal{C}$. Then $\mathcal{C}.(F, G)$ is complete.* $\square$

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 the 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 "$Q : X \to \mathsf{hSet}$", and a section of it a dependent algebra morphism "$(x : X) \to Q(x)$".[4] The usual correspondence between type families and fibrations extends to algebras, and so we formulate the elimination rule for X as X being section inductive in the category of algebras in the following sense:

Definition 26 (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 27 (Permutable trees). Consider permutable-tree algebras, e.g. tuples (X, l, n, p) as in Example 15. A fibred permutable-tree algebra over (X, l, n, p) consists of $Q : X \to \mathsf{hSet}$ together with $m_l : Q(l)$ and

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

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 $\pi_1 : \big(\Sigma(x : X).Q(x)\big) \to X$. A section of π_1 is a dependent function $h : (x : X) \to Q(x)$. Since h comes from an algebra morphism, we further know e.g. $h(l) = m_l$ and $h(n\,f) = m_n\,f\,(h \circ f)$. Conversely, every algebra morphism

[4] See Dijkstra's thesis [16, Sec 5.4] for the general definition of fibred algebras and their morphisms — here we restrict ourselves to examples only for space reasons.

$g : (X', l', n', p') \to (X, l, n, p)$ 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 l', n' and p'.

Example 28 (Contexts and types). For context-and-types algebras from Example 16, a fibred algebra over $(C, T, e, c, b, s, s_{\mathrm{eq}})$ consists of $Q : C \to \mathsf{hSet}$ and $R : (x : C) \to T(x) \to Q(x) \to \mathsf{hSet}$, together with $m_e : Q(e)$ and

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

Again, this data can be arranged into an ordinary algebra with base $C' : \mathsf{hSet}$, $T' : C' \to \mathsf{hSet}$, 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 [16, Sec 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 quite well, 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. Recall that a type is *contractible* if it is equivalent to the unit type [34, Def 3.11.1].

Definition 29 (Initiality). *An object X of a category $\mathcal{C}$ is* (homotopy) initial *if for every object Y of $\mathcal{C}$, the set of morphisms $X \to Y$ is contractible.*

It is easy to see that initiality implies section induction, while the converse requires additional structure on $\mathcal{C}$:

Lemma 30. *If an object X in a category $\mathcal{C}$ is initial, then it is section inductive. If $\mathcal{C}$ has finite limits and X is section inductive, then X is initial.* □

From here, we can show the main theorem of the current section. The proof uses the fact that both statements involved are mere propositions, i.e. they have at most one proof.

Theorem 31 (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.* $\square$

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 32. *The interval is equivalent to the unit type.*

Proof. By Theorem 31, the interval is the initial object in the category with objects $\Sigma(X : \mathsf{hSet}).\Sigma(x : X).\Sigma(y : X).x =_X y$, while the unit type is the initial object in the category with objects $\Sigma(X : \mathsf{hSet}).X$. By contractibility of singleton types [34, Lem 3.11.8], the former is equivalent to the latter, and since initiality is a universal property, the two initial objects coincide up to equivalence. $\square$

5 Conclusions and Further Work

We have developed a semantic framework for QIITs: A QIIT description gives 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 a 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 categories of algebras are complete. This is helpful for the metatheory of QIITs, as demonstrated by the proof of initiality being equivalent to section induction (Theorem 31), 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 to guarantee this, possibly by reducing QIITs to a basic type former playing an analogous role to that of W-types for inductive types. We believe that completeness of the categories of algebras allows 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.

Other possible future directions include the combination of QIITs and induction-recursion, and the possibility of generalising coinductive types along similar lines. These generalisations should be driven by examples, similar to how the examples discussed in the current paper have motivated the need for a theory of QIITs.

Acknowledgements. We thank Ambrus Kaposi and Jakob von Raumer for many interesting discussions, and the anonymous referees for their valuable comments. 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. Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. Quotient inductive-inductive types, 2018. arXiv:1612.02346.
4. Carlo Angiuli, Edward Morehouse, Daniel R. Licata, and Robert Harper. Homotopical patch theory. In *International Conference on Functional Programming*, pages 243–256, 2014.
5. Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1): 45–55, 2009.
6. Henning Basold, Herman Geuvers, and Niels van der Weide. Higher inductive types in programming. *Journal of Universal Computer Science*, 23(1):63–88, 2016.
7. 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.
8. Edwin Brady. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, 23:552–593, 9 2013.
9. Guillaume Brunerie. *On the homotopy groups of spheres in homotopy type theory*. PhD thesis, Universitè de Nice, 2016.
10. Ulrik Buchholtz and Egbert Rijke. The real projective spaces in homotopy type theory. In *Logic in Computer Science*, pages 1–8, 2017.
11. Evan Cavallo. Synthetic cohomology in Homotopy Type Theory. Master's thesis, Carnegie-Mellon University, 2015.
12. 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.
13. James Chapman, Tarmo Uustalu, and Niccolò Veltri. Quotienting the delay monad by weak bisimilarity. *Mathematical Structures in Computer Science*, page 1–26, 2017.
14. Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover. In *Conference on Automated Deduction*, 2015.
15. Radu Diaconescu. Axiom of choice and complementation. *Proceedings of the American Mathematical Society*, 51(1):176–178, 1975.
16. Gabe Dijkstra. *Quotient inductive-inductive types*. PhD thesis, University of Nottingham, 2017.
17. Peter Dybjer. Inductive families. *Formal aspects of computing*, 6(4):440–465, 1994.
18. Peter Dybjer and Hugo Moeneclaey. Finitary higher inductive types in the groupoid model. In Alexandra Silva, editor, *Mathematical Foundations of Programming Semantics*, 2017.
19. Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *Typed lambda calculi and applications*, pages 129–146. Springer, 1999.

20. Tatsuya Hagino. *A Categorical Programming Language.* PhD thesis, University of Edinburgh, 1987.
21. Robert Harper and Robert Pollack. Type checking with universes. *Theoretical Computer Science*, 89(1):107–136, 1991.
22. Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford University Press, 1998.
23. 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.
24. Peter Johnstone. *Sketches of an Elephant: A Topos Theory Compendium.* Oxford University Press, 2002.
25. Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after Voevodsky), 2016. arXiv:1211.2851.
26. Daniel R. Licata and Eric Finster. Eilenberg-Maclane spaces in homotopy type theory. In *Logic in Computer Science*, pages 66:1–66:9, 2014.
27. 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.
28. Peter LeFanu Lumsdaine and Mike Shulman. Semantics of higher inductive types, 2017. arXiv:1705.07088.
29. Per Martin-Löf. An intuitionistic theory of types. Published in Twenty-Five Years of Constructive Type Theory, 1972.
30. Fredrik Nordvall Forsberg. *Inductive-inductive definitions.* PhD thesis, Swansea University, 2013.
31. Ulf Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Chalmers University of Technology, 2007.
32. Michael Shulman. Homotopy type theory: the logic of space, 2017. To appear in *New Spaces for Mathematics and Physics.* arXiv:1703.03007.
33. Kristina Sojakova. Higher inductive types as homotopy-initial algebras. In *Principles of Programming Languages*, pages 31–42. ACM, 2015.
34. The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.
35. Niccolò Veltri. *A Type-Theoretical Study of Nontermination.* PhD thesis, Tallinn University of Technology, 2017.
36. Vladimir Voevodsky. The equivalence axiom and univalent models of type theory (talk at CMU on February 4, 2010), 2010. arXiv:1402.5556.