Parametricity and Semi-Cubical Types

Hugo Moeneclaey
Université de Paris,
Inria Paris, CNRS, IRIF,
France

May 19, 2021

Abstract

We construct a model of type theory enjoying parametricity from an arbitrary one. A type in the new model is a semi-cubical type in the old one, illustrating the correspondence between parametricity and cubes.

Our construction works not only for parametricity, but also for similar interpretations of type theory and in fact similar interpretations of any generalized algebraic theory. To be precise we consider a functor forgetting unary operations and equations defining them recursively in a generalized algebraic theory. We show that it has a right adjoint.

We use techniques from locally presentable category theory, as well as from quotient inductive-inductive types.

Contents

1 Overview

1.1 Introduction to type theory

Martin Löf type theory [21] is a foundational system for constructive mathematics. Most modern proof assistants (for example Agda, Coq, Lean) are based on variants of this system.

In this paper we use a semantical approach: we study models of type theory rather than type theory itself. Such a model supplies suitable notions of types (corresponding to sets and properties) and terms inhabiting them (corresponding to elements of sets and proofs of propositions). It can be intuitively conceived as a world where (constructive) mathematics can take place. We give two basic models:

- The set model is the traditional mathematical world.

- The initial model is inhabited by the elements definable from the syntax of type theory. Its elements and identifications are precisely the ones which can be derived from the axioms of type theory.

Proof assistants implement the initial model. This makes possible a rich interaction between models of type theory and proof assistants, mainly using two principles:

- On the one hand a given model has features absent from the initial one. They can suggest extensions for a proof assistant.

- On the other hand a term in the initial model can be interpreted in any model. So a formal proof gives a multitude of theorems, one for each model.

The abundance of models for type theory makes this approach fruitful. We give an example for each principle.

Perhaps the most striking use of the first principle comes from the Kan cubical set model given in [10]. This model was used to design cubical type theory [13], which is now implemented as Cubical Agda [26]. This proof assistant based on Agda supports (among other features) the axiom of univalence, implying that isomorphic types are equal. This axiom has applications in computer science, where it allows to transport programs along isomorphisms, and in higher mathematics (that is mathematics with everything considered up to homotopy).

Now we give an example for the second principle. Schreier theory classifies group extensions. It can be proven in type theory with univalence using an alternative definition of groups as pointed connected types (a sketch is given by David Myers at `http://davidjaz.com/Talks/DJM_HoTT2020.pdf`). But type theory with univalence can be interpreted in any higher topos [23], so we get a higher Schreier theory classifying group extensions in higher topoi. But the type-theoretic proof is significantly easier than the usual proof for regular Schreier theory, let alone a higher version! The canonical introduction to such synthetic homotopy theory using univalence is called the HoTT book [1].

1.2 Parametricity and cubical structure

Now we introduce a fundamental tool of type theory called parametricity. Originally designed for system F [22], a generalization to type theory can be found in [8]. From our semantical point of view, parametricity consists of some operations $\_^*$ defined recursively in the initial model. Using these operations we can extract *Theorems for free!* [27] for polymorphic terms in the initial model (that is terms taking a type as input). For example given any term t in the initial model of the same type as the polymorphic identity, the term t^* proves that t behaves as the polymorphic identity. Parametricity can be summarized by saying that for any term t in the initial model, t^* proves that t preserves relations.

We define a parametric model as a model of type theory together with operations $\_^*$ showing that terms in this model preserve relations. The main goal of this article is to construct a parametric model from an arbitrary one.

Now we introduce the seemingly unrelated semi-cubical sets. A semi-cubical set consists of a set of points together with:

- For any two points, a set of paths between them.

- For any four points and four paths drawing a square, a set of surfaces having this square as border.

- And so on in higher dimensions (cubes, hypercubes...).

The geometric intuition should be clear from our vocabulary. A cubical set is a semi-cubical set with degeneracies, meaning constant paths, constant surfaces, etc.

It is known that there is a strong connection between cubical structures and parametricity. In [6] a model for non-iterated parametricity (i.e. parametricity where $\_^*$ can only be applied once) by 1-truncated cubical sets (i.e. reflexive graphs) is given. In [16] a model for the so-called proof-relevant parametricity (where $\_^*$ can be applied twice) is given using 2-truncated semi-cubical sets. Alternatively [24] gives a general, foundation-agnostic, definition of a model for non-iterated parametricity in system F using reflexive graphs, and [17] extends this to full parametricity using cubical structures.

Moreover it is known that cubical structures arise when trying to internalize parametricity in type theory. For example [7] gives a model for (the unary

variant of) parametricity in (the unary variant of) cubical sets, and [12] shows how parametricity can be internalized orthogonally to univalence, using very similar cubical techniques for both features.

In this paper we will show how semi-cubical structures arise already from iterated parametricity without internalization, and can be used to construct a parametric model from any given model.

1.3 Content of this paper

We use the language of categories. There is a functor from parametric models to arbitrary models, which forgets the unary operations $\_^*$. This forgetful functor has a poorly behaved left adjoint freely adding parametricity, sending non-parametric models to trivial ones. In this paper we show this forgetful functor also has a better-behaved right adjoint. We will see that this right adjoint sends a model $\mathcal{C}$ to the model of semi-cubical types in $\mathcal{C}$. For example it sends the set model to the semi-cubical set model. Moreover we will see that our method for constructing this right adjoint does not use much features of parametricity. From these two facts a picture emerges: from an interpretation of type theory (for example parametricity), we get:

- A notion of structure on types (for example semi-cubical types).

- Models for this interpretation by types with this structure (for example semi-cubical models for parametricity).

Recall there is a link between cubical structure and internal parametricity. In fact we pursue the idea that univalence is a variant of parametricity, and that we have a similar link between Kan cubical structure and univalence. This is already evoked in [5] which introduces a syntax for a univalent type theory inspired by parametricity but does not prove univalence, and in [25] which unifies parametricity and univalence, although assuming a univalent universe to begin with. To summarize we conjecture a table:

Interpretation	Structure
External parametricity	Semi-cubical types
Internal parametricity	Cubical types
External univalence	Kan semi-cubical types
Internal univalence	Kan cubical types

In this paper we give the procedure supposedly linking the two columns, and study the first line in detail. It is organized as follows:

- In Section 2 we define parametricity for models of type theory. Then we give a general definition of an interpretation, give toy examples, and prove that parametricity is an interpretation. It should be noted that we use unary parametricity (which can also be seen as a form of realizability as first noted in [9]) for notational convenience, so we obtain something a bit different from semi-cubes. But our approach can be extended to binary parametricity and semi-cubes straightforwardly.

4

- In Section 3 we give general conditions implying the existence of right adjoints to forgetful functors for extensions of generalized algebraic theories. To do this we use locally presentable categories, which generalize categories of models for an algebraic theory. The standard textbook is [3]. This short section contains only well-known material, but we include it anyway because the intended audience for this paper might not be familiar with it.

- In Section 4 we prove our main result, constructing a right adjoint from any interpretation. We examine this adjoint for our toy examples of interpretation to help build intuition, and then for parametricity to see it constructs semi-cubical models.

Remark 1. *Three very similar syntactic notions can be used as a basis for the general definition of an interpretation.*

- *Essentially algebraic theories, giving rise to locally presentable categories [3].*

- *Generalized algebraic theories, allowing dependencies [11].*

- *Signatures for quotient inductive-inductive types, where there is a built-in strong induction principle for initial objects [18].*

Technically we define interpretations using generalized algebraic theories, but we freely use the most convenient of these three notions throughout this paper, as they are intuitively equivalent. This should not be interpreted as a claim that these three notions are equivalent in a technical sense, as we do not know any satisfying reference for this, and it is out of scope to prove it here. However our main example (parametricity as an interpretation of type theory) can indeed be defined using any of these three syntactic notions.

2 Parametricity as an interpretation

2.1 Models of type theory

We define a model of type theory as a CwF (Category with Families [14]), so they are algebras for a GAT (Generalized Algebraic Theory in Cartmell's sense [11]). We follow the presentation in [4], which explains how such a GAT can be seen as a signature for a QIIT (Quotient Inductive-Inductive Type). Indeed the induction principle for the initial model of type theory seen as a QIIT will be crucial in our construction.

First we define CwFs. In this definition Ctx stands for contexts, Ty for types, Sub for substitutions and Tm for terms. We use type-theoretic notations, so a set-minded reader should replace $x : A$ by $x \in A$ and $p\ q$ by $p(q)$. Moreover any variable appearing undeclared is in fact universally quantified. We use Agda notations, meaning we use:

$$(x : A) \to B(x) \tag{1}$$

for what is usually denoted $\forall x : A, B(x)$ or $\Pi(x : A).B(x)$.

Definition 2. *A CwF consists of:*

$$\text{Ctx} : \text{Set} \tag{2}$$

$$\text{Ty} : \text{Ctx} \to \text{Set} \tag{3}$$

$$\text{Sub} : \text{Ctx} \to \text{Ctx} \to \text{Set} \tag{4}$$

$$\text{Tm} : (\Gamma : \text{Ctx}) \to \text{Ty } \Gamma \to \text{Set} \tag{5}$$

With constructors for contexts:

$$\cdot : \text{Ctx} \tag{6}$$

$$(\_,\_) : (\Gamma : \text{Ctx}) \to \text{Ty } \Gamma \to \text{Ctx} \tag{7}$$

types:

$$\_[\_] : \text{Ty } \Delta \to \text{Sub } \Gamma \ \Delta \to \text{Ty } \Gamma \tag{8}$$

substitutions:

$$\_ \circ \_ : \text{Sub } \Delta \ \Theta \to \text{Sub } \Gamma \ \Delta \to \text{Sub } \Gamma \ \Theta \tag{9}$$

$$\text{id} : \text{Sub } \Gamma \ \Gamma \tag{10}$$

$$\epsilon : \text{Sub } \Gamma \ \cdot \tag{11}$$

$$(\_,\_) : (\delta : \text{Sub } \Gamma \ \Delta) \to \text{Tm } \Gamma \ (A[\delta])$$
$$\to \text{Sub } \Gamma \ (\Delta, A) \tag{12}$$

$$\pi_1 : \text{Sub } \Gamma \ (\Delta, A) \to \text{Sub } \Gamma \ \Delta \tag{13}$$

and terms:

$$\_[\_] : \text{Tm } \Delta \ A \to (\delta : \text{Sub } \Gamma \ \Delta)$$
$$\to \text{Tm } \Gamma \ (A[\delta]) \tag{14}$$

$$\pi_2 : (\sigma : \text{Sub } \Gamma \ (\Delta, A)) \to \text{Tm } \Gamma \ (A[\pi_1 \ \sigma]) \tag{15}$$

with the following equations for types:

$$A[\sigma \circ \eta] = A[\sigma][\eta] \tag{16}$$

$$A[\text{id}] = A \tag{17}$$

substitutions:

$$(\sigma \circ \nu) \circ \delta = \sigma \circ (\nu \circ \delta) \tag{18}$$

$$\text{id} \circ \sigma = \sigma \tag{19}$$

$$\sigma \circ \text{id} = \sigma \tag{20}$$

$$\sigma = \epsilon \tag{21}$$

$$\pi_1 \ (\sigma, t) = \sigma \tag{22}$$

$$(\pi_1 \ \sigma, \pi_2 \ \sigma) = \sigma \tag{23}$$

$$(\sigma, t) \circ \nu = (\sigma \circ \nu, t[\nu]) \tag{24}$$

and terms:

$$\pi_2\ (\sigma, t) = t \tag{25}$$

Note that some equations in the definition need the previous ones to be well-typed. CwFs are worlds where one can substitute, so they are sometimes called models for the calculus of substitutions. But not much can be done in an arbitrary CwF since there is no way to construct types. We define additional structures on a CwF, which will be assumed in a model of type theory.

Now we introduce some useful notations where w stands for weakening and v for the last variable.

Notation 3. *We define for* $\Gamma : \mathrm{Ctx}$ *and* $A : \mathrm{Ty}\ \Gamma$.

$$\mathrm{w} = \pi_1\ \mathrm{id} : \mathrm{Sub}\ (\Gamma, A)\ \Gamma \tag{26}$$

$$\mathrm{v} = \pi_2\ \mathrm{id} : \mathrm{Tm}\ (\Gamma, A)\ A[\mathrm{w}] \tag{27}$$

Using this notation we have that $\mathrm{v}[\mathrm{w}^n]$ is similar to the de Bruijn index n, where w^n is $\mathrm{w} \circ \cdots \circ \mathrm{w}$ where w is composed n times.

Definition 4. *A unit for a CwF consists of:*

$$\top : \mathrm{Ty}\ \Gamma \tag{28}$$

$$\mathrm{tt} : \mathrm{Tm}\ \Gamma\ \top \tag{29}$$

such that for all $x : \mathrm{Tm}\ \Gamma\ \top$ *we have:*

$$x = \mathrm{tt} \tag{30}$$

with equations for substitutions:

$$\top[\sigma] = \top \tag{31}$$

$$\mathrm{tt}[\sigma] = \mathrm{tt} \tag{32}$$

Definition 5. *Products for a CwF consist of:*

$$\Sigma : (A : \mathrm{Ty}\ \Gamma) \to \mathrm{Ty}\ (\Gamma, A) \to \mathrm{Ty}\ \Gamma \tag{33}$$

$$\_.1 : \mathrm{Tm}\ \Gamma\ (\Sigma\ A\ B) \to \mathrm{Tm}\ \Gamma\ A \tag{34}$$

$$\_.2 : (t : \mathrm{Tm}\ \Gamma\ (\Sigma\ A\ B)) \to \mathrm{Tm}\ \Gamma\ B[\mathrm{id}, t.1] \tag{35}$$

$$(\_,\_) : (t : \mathrm{Tm}\ \Gamma\ A) \to \mathrm{Tm}\ \Gamma\ B[\mathrm{id}, t]$$
$$\to \mathrm{Tm}\ \Gamma\ (\Sigma\ A\ B) \tag{36}$$

such that we have:

$$(s, t).1 = s \tag{37}$$

$$(s, t).2 = t \tag{38}$$

$$(t.1, t.2) = t \tag{39}$$

with equations for substitutions:

$$(\Sigma\ A\ B)[\sigma] = \Sigma\ A[\sigma]\ B[\sigma \circ \mathrm{w}, \mathrm{v}] \tag{40}$$

$$(s, t)[\sigma] = (s[\sigma], t[\sigma]) \tag{41}$$

Remark 6. *We call $\Sigma\ A\ B$ a product because it specializes to the cartesian product $A \times B$ when B does not depend on A. Confusingly it is sometimes called a dependent sum.*

Remark 7. *The notation $(\_,\_)$ is overloaded, as it can be used for contexts, substitutions or terms.*

Definition 8. *Functions for a CwF consist of:*

$$\Pi : (A : \mathrm{Ty}\ \Gamma) \to \mathrm{Ty}\ (\Gamma, A) \to \mathrm{Ty}\ \Gamma \tag{42}$$

$$\mathrm{app} : \mathrm{Tm}\ \Gamma\ (\Pi\ A\ B) \to \mathrm{Tm}\ (\Gamma, A)\ B \tag{43}$$

$$\lambda : \mathrm{Tm}\ (\Gamma, A)\ B \to \mathrm{Tm}\ \Gamma\ (\Pi\ A\ B) \tag{44}$$

such that we have:

$$\mathrm{app}\ (\lambda\ t) = t \tag{45}$$

$$\lambda\ (\mathrm{app}\ t) = t \tag{46}$$

with equations for substitutions:

$$(\Pi\ A\ B)[\sigma] = \Pi\ A[\sigma]\ B[\sigma \circ \mathrm{w}, \mathrm{v}] \tag{47}$$

$$(\lambda\ t)[\sigma] = \lambda\ (t[\sigma \circ \mathrm{w}, \mathrm{v}]) \tag{48}$$

Remark 9. *The previous definitions imply:*

$$(t.1)[\sigma] = t[\sigma].1 \tag{49}$$

$$(t.2)[\sigma] = t[\sigma].2 \tag{50}$$

$$(\mathrm{app}\ t)[\sigma \circ \mathrm{w}, \mathrm{v}] = \mathrm{app}\ (t[\sigma]) \tag{51}$$

So that there are no missing equations for substitutions.

Definition 10. *A universe for a CwF consists of:*

$$\mathcal{U} : \mathrm{Ty}\ \Gamma \tag{52}$$

$$\mathrm{El} : \mathrm{Tm}\ \Gamma\ \mathcal{U} \to \mathrm{Ty}\ \Gamma \tag{53}$$

$$\top_{\mathcal{U}} : \mathrm{Tm}\ \Gamma\ \mathcal{U} \tag{54}$$

$$\Sigma_{\mathcal{U}} : (s : \mathrm{Tm}\ \Gamma\ \mathcal{U}) \to \mathrm{Tm}\ (\Gamma, \mathrm{El}\ s)\ \mathcal{U} \to \mathrm{Tm}\ \Gamma\ \mathcal{U} \tag{55}$$

$$\Pi_{\mathcal{U}} : (s : \mathrm{Tm}\ \Gamma\ \mathcal{U}) \to \mathrm{Tm}\ (\Gamma, \mathrm{El}\ s)\ \mathcal{U} \to \mathrm{Tm}\ \Gamma\ \mathcal{U} \tag{56}$$

such that we have:

$$\mathrm{El}\ \top_{\mathcal{U}} = \top \tag{57}$$

$$\mathrm{El}\ (\Sigma_{\mathcal{U}}\ s\ t) = \Sigma\ (\mathrm{El}\ s)\ (\mathrm{El}\ t) \tag{58}$$

$$\mathrm{El}\ (\Pi_{\mathcal{U}}\ s\ t) = \Pi\ (\mathrm{El}\ s)\ (\mathrm{El}\ t) \tag{59}$$

with equations for substitutions:

$$\mathcal{U}[\sigma] = \mathcal{U} \tag{60}$$

$$(\text{El } t)[\sigma] = \text{El } (t[\sigma]) \tag{61}$$

$$\top_{\mathcal{U}}[\sigma] = \top_{\mathcal{U}} \tag{62}$$

$$(\Sigma_{\mathcal{U}} \ s \ t)[\sigma] = \Sigma_{\mathcal{U}} \ s[\sigma] \ t[\sigma \circ \text{w}, \text{v}] \tag{63}$$

$$(\Pi_{\mathcal{U}} \ s \ t)[\sigma] = \Pi_{\mathcal{U}} \ s[\sigma] \ t[\sigma \circ \text{w}, \text{v}] \tag{64}$$

Definition 11. *A model of type theory is a CwF with a unit, products, functions and a universe.*

There exist many variants of type theory, with fewer or more types. The most common extensions consist in adding some inductive types (for example booleans, natural numbers, identity types, W-types...) with sometimes a hierarchy of universes and maybe a scheme for general inductive families.

Our abstract approach using an interpretation makes it easy to check whether this article works for a given extension. It certainly works for the common ones.

2.2 Parametric models

Now we define what it means for a model of type theory to be parametric. We use unary parametricity, which can be seen as a case of realizability.

Definition 12. *A parametric model is a model of type theory together with:*

$$\_^* : (\Gamma : \text{Ctx}) \to \text{Ty } \Gamma \tag{65}$$

$$\_^* : (A : \text{Ty } \Gamma) \to \text{Ty } (\Gamma, \Gamma^*, A[\text{w}]) \tag{66}$$

$$\_^* : (\sigma : \text{Sub } \Gamma \ \Delta) \to \text{Tm } (\Gamma, \Gamma^*) \ \Delta^*[\sigma \circ \text{w}] \tag{67}$$

$$\_^* : (t : \text{Tm } \Gamma \ A) \to \text{Tm } (\Gamma, \Gamma^*) \ A^*[\text{id}, t[\text{w}]] \tag{68}$$

Such that we have equations for substitutions:

$$\cdot^* = \top \tag{69}$$

$$(\Gamma, A)^* = \Sigma \ \Gamma^*[\text{w}] \ A^*[\text{w}^2, \text{v}, \text{v}[\text{w}]] \tag{70}$$

$$(A[\sigma])^* = A^*[\sigma \circ \text{w}^2, \sigma^*[\text{w}], \text{v}] \tag{71}$$

$$(\sigma \circ \nu)^* = \sigma^*[\nu \circ \text{w}, \nu^*] \tag{72}$$

$$\text{id}^* = \text{v} \tag{73}$$

$$\epsilon^* = \text{tt} \tag{74}$$

$$(\sigma, t)^* = (\sigma^*, t^*) \tag{75}$$

$$(\pi_1 \ \sigma)^* = \sigma^*.1 \tag{76}$$

$$(t[\sigma])^* = t^*[\sigma \circ \text{w}, \sigma^*] \tag{77}$$

$$(\pi_2 \ \sigma)^* = \sigma^*.2 \tag{78}$$

for the unit:

$$\top^* = \top \tag{79}$$

$$\text{tt}^* = \text{tt} \tag{80}$$

for products:

$$(\Sigma \ A \ B)^* = \Sigma \ A^*[\eta_1] \ B^*[\eta_2] \tag{81}$$

$$(t.1)^* = t^*.1 \tag{82}$$

$$(t.2)^* = t^*.2 \tag{83}$$

$$(s,t)^* = (s^*,t^*) \tag{84}$$

where:

$$\eta_1 = (\text{w}, \text{v}.1) \tag{85}$$

$$\eta_2 = (\text{w}^3, \text{v}.1[\text{w}], (\text{v}[\text{w}^2], \text{v}), \text{v}.2[\text{w}]) \tag{86}$$

for functions:

$$(\Pi \ A \ B)^* = \Pi \ A[\sigma_1] \ (\Pi \ A^*[\sigma_2] \ B^*[\sigma_3]) \tag{87}$$

$$(\text{app} \ t)^* = (\text{app} \ (\text{app} \ t^*))[\nu_1] \tag{88}$$

$$(\lambda \ t)^* = \lambda \ (\lambda \ (t^*[\nu_2])) \tag{89}$$

where:

$$\sigma_1 = \text{w}^2 \tag{90}$$

$$\sigma_2 = (\text{w}^2, \text{v}) \tag{91}$$

$$\sigma_3 = (\text{w}^4, \text{v}[\text{w}], (\text{v}[\text{w}^3], \text{v}), (\text{app} \ \text{v})[\text{w}]) \tag{92}$$

$$\nu_1 = (\text{w}^2, \text{v}.1, \text{v}[\text{w}], \text{v}.2) \tag{93}$$

$$\nu_2 = (\text{w}^3, \text{v}[\text{w}], (\text{v}[\text{w}^2], \text{v})) \tag{94}$$

and for the universe:

$$\mathcal{U}^* = \Pi \ (\text{El} \ \text{v}) \ \mathcal{U} \tag{95}$$

$$(\text{El} \ t)^* = \text{El} \ (\text{app} \ t^*) \tag{96}$$

$$(\top_{\mathcal{U}})^* = \lambda \ \top_{\mathcal{U}} \tag{97}$$

$$(\Sigma_{\mathcal{U}} \ s \ t)^* = \lambda \ (\Sigma_{\mathcal{U}} \ (\text{app} \ s^*)[\eta_1] \ (\text{app} \ t^*)[\eta_2]) \tag{98}$$

$$(\Pi_{\mathcal{U}} \ s \ t)^* = \lambda \ (\Pi_{\mathcal{U}} \ s[\sigma_1] \ (\Pi_{\mathcal{U}} \ (\text{app} \ s^*)[\sigma_2]$$
$$(\text{app} \ t^*)[\sigma_3])) \tag{99}$$

The main point of this lengthy definition is that the equations ensure that the operations $\_^*$ are recursively defined in the initial model by the given equations. This will be checked in detail in the Appendix. We will capture this feature in the definition of an interpretation.

Remark 13. *To treat binary parametricity a product of contexts should be added, so that for $\Gamma : \mathrm{Ctx}$ we can define $\Gamma^* : \mathrm{Ty}\ (\Gamma, \Gamma)$.*

Remark 14. *A parametric model needs products and a unit in the equations for $\cdot^*$ and $(\Gamma, A)^*$. Moreover a parametric model with a universe needs functions in the equation for $\mathcal{U}^*$.*

2.3   Definition of an interpretation

In this section we give a definition capturing the features of parametricity making the rest of this paper work. We assume the reader familiar with GAT (Generalized Algebraic Theories [11]).

For T a GAT, we denote by Alg_T the category of models of T and $\mathcal{I}_T$ its initial model, so $\mathcal{I}_T$ is the initial object in Alg_T. Note that we will only consider finitary GATs.

Definition 15. *Let T be a finitary GAT. An interpretation for T is a GAT of the form:*

$$T, O, E, E' \tag{100}$$

where:

- *O is a set of unary operations defined recursively in $\mathcal{I}_T$ by equations E.*

- *E' is a set of unary equations proved inductively in $\mathcal{I}_T$.*

Remark 16. *Here a unary operation is an operation with one main input, and possibly secondary inputs inferred from it. This means that:*

$$(x : A) \rightarrow (y : B\ x) \rightarrow C \tag{101}$$

is considered unary, as x can be inferred from y. For example the operations (65) to (68) are considered unary. They take a context, a type, a term or a substitution as their main input.

Unary equations are defined similarly as equations depending on one main variable, with possibly secondary variables inferred from it.

Remark 17. *We clarify the expression 'recursively defined in $\mathcal{I}_T$'. It means that for any constructor c in T and any added unary operation $\_^*$ in T', we have an equation in T' of the form:*

$$(c(x_1, \ldots, x_n))^* = c^*(x_1, x_1^*, \ldots, x_n, x_n^*) \tag{102}$$

where c^* is a term in T.

Moreover it means that for any equation $s = t$ in T, we have $s^* = t^*$ in T, where s^* and t^* are computed using the recursive equations.

Equation 102 makes sense only when there is precisely one unary operation $\_^*$ by sort (as for parametricity), but the analogous general formula is clear.

Remark 18. *This definition can be reformulated more precisely using the theory of signatures for QIITS as in [19]. In this theory, given a signature:*

$$\Gamma \vdash \tag{103}$$

we have a signature for displayed algebras:

$$\Gamma \vdash \Gamma^D \tag{104}$$

and a signature for sections of such a displayed algebra:

$$\Gamma, \Gamma^D \vdash \Gamma^S \tag{105}$$

Then an interpretation is an extension of Γ of the form:

$$\Gamma, \Gamma^S[\mathrm{id}, t] \tag{106}$$

where we have:

$$\Gamma \vdash t : \Gamma^D \tag{107}$$

in the theory of signature.

Interpretations will be used in Section 4. In Section 3 we will use the weaker notion of extensions by operations and equations, without new sorts. Now we give toy examples of interpretations.

Example 19. *The theory of sets with an endofunction is an interpretation of the theory of sets. This just means the extension of:*

$$X : \mathrm{Set} \tag{108}$$

by:

$$s : X \to X \tag{109}$$

is an interpretation. We do not need any equation defining s because X is empty in the initial model.

Example 20. *The theory of groups is an interpretation of the theory of monoids. Indeed it is the extension of the theory of a monoid $(M, e, \cdot)$ where O is:*

$$\_^{-1} : M \to M \tag{110}$$

and E is:

$$e^{-1} = e \tag{111}$$

$$(m \cdot n)^{-1} = n^{-1} \cdot m^{-1} \tag{112}$$

with E' as follows:

$$m \cdot m^{-1} = e \tag{113}$$

$$m^{-1} \cdot m = e \tag{114}$$

12

Example 21. *The theory of reflexive graphs is an interpretation of the theory of graphs. Indeed this is the extension of the theory of graphs:*

$$V : \mathrm{Set} \tag{115}$$

$$E : V \to V \to \mathrm{Set} \tag{116}$$

by:

$$r : (v : V) \to E\ v\ v \tag{117}$$

Now we are ready to give our main interpretation.

Theorem 1. *Parametricity is an interpretation of type theory.*

Proof. We denote by TT (resp. pTT) the theory of models of type theory (resp. parametric such models). We see that operations $\_^*$ are unary for TT.

In order to prove that pTT is an interpretation of TT, we need to check that equations (69) to (99) from Definition 12 indeed define some operations $\_^*$ in $\mathcal{I}_{\mathrm{TT}}$ the initial model of type theory, as indicated in (65) to (68).

To check this we see the initial model of type theory as a QIIT [4]. Then we just need to check that for any equation $s = t$ in TT, the elements s^*, t^* defined recursively by equations (69) to (99) are equal in $\mathcal{I}_{\mathrm{TT}}$. This tedious but straightforward task is done in the Appendix. $\qquad\blacksquare$

Remark 22. *The reader uninterested in extra generality can replace any interpretation T' of T by the interpretation* pTT *of* TT *in the rest of this paper.*

We will show that for any interpretation T' of T, the forgetful functor:

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \tag{118}$$

has a right adjoint.

3 Adjoints to forgetful functors

In this section we assume an extension of finitary GATs denoted $T \subset T'$ where T' adds operations and equations, but no new sort to T. We want to show that the forgetful functor (118) has a right adjoint if and only if it commutes with finite colimits. We have in mind the special case where T' is an interpretation of T.

We will use the fact that finitary GATs correspond to finitary essentially algebraic theories as indicated in Section 6 of [11]. This means (among other things) that models for a finitary GAT form an lfp (locally finitely presented) category. The rich theory of such categories is presented in [3]. In this section we will use both GATs and essentially algebraic theories at will.

All the results presented in this section are well-known to experts in lfp categories.

3.1 Existence of a left adjoint

Lemma 23. *The forgetful functor:*

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \tag{119}$$

has a left adjoint.

Proof. This is mentioned at the end of Section 15 in [11], as a straightforward extension of this fact for algebraic theories [20]. □

Example 24. *For parametricity, the left adjoint is freely adding parametricity to a model of type theory. This construction is often degenerate. Indeed freely adding parametricity to a model contradicting parametricity gives an inconsistent model, that is a mathematical world where everything is true.*

3.2 The forgetful functor is finitary

Recall that a functor is called finitary if it commutes with filtered colimits, and conservative if it reflects isomorphisms.

Lemma 25. *The forgetful functor:*

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \tag{120}$$

is finitary and conservative.

Proof. Lemma 2.3.6 from [15] states that for a finitary essentially algebraic theory T the forgetful functor (here S is the set of sorts in T):

$$U_T : \mathrm{Alg}_T \to \mathrm{Set}^S \tag{121}$$

is finitary and conservative. Then we have a commuting triangle of functors:

$$U_T U = U_{T'} \tag{122}$$

and U is finitary and conservative by Lemma 26. □

Lemma 26. *Assume two functors F and G. If G and GF are finitary and conservative, then so is F.*

Proof. It is clear that F is conservative, because if $F(g)$ is an isomorphism, so is $GF(g)$, and then so is g because GF is conservative.

For $U : \mathcal{C} \to \mathcal{D}$ a functor and a diagram $i \mapsto c_i$ in $\mathcal{C}$ we denote by ψ_U the canonical map:

$$\psi_U : \mathrm{colim}_i U(c_i) \to U(\mathrm{colim}_i c_i) \tag{123}$$

By definition, U commutes with the colimit of $i \mapsto c_i$ if and only if ψ_U is an isomorphism.

For any diagram $i \mapsto c_i$, we have a commutative triangle:

$$G(\psi_F) \circ \psi_G = \psi_{GF} \tag{124}$$

If the diagram is filtered ψ_G and ψ_{GF} are isomorphisms, therefore so is $G(\psi_F)$. But G is conservative so ψ_F is an isomorphism and F is finitary. □

3.3 Sufficient condition for a right adjoint

The next lemma is well-known for functors between lfp categories.

Lemma 27. *The forgetful functor:*

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \tag{125}$$

has a right adjoint if and only if it commutes with small colimits.

Proof. Recall that the category of models for a finitary essentially algebraic theory is an lfp category (by Theorem 3.36 in [3]). Moreover (by Theorem 1.46 in [3]) any lfp category $\mathcal{C}$ is equivalent to:

$$\mathrm{Lex}(\mathcal{C}_{\mathrm{fin}}^{op}, \mathrm{Set}) \tag{126}$$

via the Yoneda embedding where:

- $\mathcal{C}_{\mathrm{fin}}$ is the category of finitely presented objects in $\mathcal{C}$.

- $\mathrm{Lex}(\mathcal{C}, \mathcal{D})$ is the category of functors from $\mathcal{C}$ to $\mathcal{D}$ commuting with finite limits.

We suppose given a functor between lfp categories:

$$U : \mathcal{C} \to \mathcal{D} \tag{127}$$

commuting with small colimits. We define:

$$R \;:\; \mathcal{D} \to \mathrm{Lex}(\mathcal{C}_{\mathrm{fin}}^{op}, \mathrm{Set}) \tag{128}$$

$$R(d) = \mathrm{Hom}_{\mathcal{D}}(U\_, d) \tag{129}$$

This is well-defined because U commutes with finite colimits. Now recall that in an lfp category any object c is a canonical filtered colimit of finitely presented objects (Proposition 1.22 in [3]), so we have:

$$c = \mathrm{colim}_i \, c_i \tag{130}$$

with c_i finitely presented. But by the definition of R we have:

$$\mathrm{Hom}_{\mathcal{C}}(c_i, R(d)) = \mathrm{Hom}_{\mathcal{D}}(U(c_i), d) \tag{131}$$

therefore:

$$\mathrm{Hom}_{\mathcal{C}}(c, R(d)) = \mathrm{Hom}_{\mathcal{C}}(\mathrm{colim}_i \, c_i, R(d)) \tag{132}$$
$$= \lim_i \mathrm{Hom}_{\mathcal{C}}(c_i, R(d)) \tag{133}$$
$$= \lim_i \mathrm{Hom}_{\mathcal{D}}(U(c_i), d) \tag{134}$$
$$= \mathrm{Hom}_{\mathcal{D}}(\mathrm{colim}_i \, U(c_i), d) \tag{135}$$
$$= \mathrm{Hom}_{\mathcal{D}}(U(c), d) \tag{136}$$

where we used the fact that U commutes with filtered colimits. So R is indeed a right adjoint to U. $\square$

Theorem 2. *The forgetful functor:*

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \tag{137}$$

has a right adjoint if and only if it commutes with finite colimits.

Proof. It is well-known that a functor commuting with finite and filtered colimits commutes with small colimits. This is because any small colimit is a coequalizer of coproducts, and a coproduct is a filtered colimit of finite coproducts.

But U commutes with filtered colimits by Lemma 25, so by Lemma 27 it has a right adjoint if and only if it commutes with finite colimits. $\square$

4 Constructing right adjoints from interpretations

In this section we assume given T' an interpretation of T. Mimicking parametricity, we denote by $\_^*$ any unary operation added in T'. We want to prove that the forgetful functor:

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \tag{138}$$

commutes with finite colimits, so that it has a right adjoint. This theorem is proved using the definition of colimits in Alg_T and $\mathrm{Alg}_{T'}$ as QIITs.

Notation 28. *In the rest of this section we write $\langle s_j \rangle$ for the sequence $(s_1, \ldots, s_n)$, where n can be inferred.*

Notation 29. *As already indicated in Remark 17, for c a constructor in T, and $\_^*$ a unary operation added in T', we denote by:*

$$c\langle x_j \rangle^* = c^* \langle x_j, x_j^* \rangle \tag{139}$$

with c^* some term in T the equation defining $\_^*$ recursively on c.

4.1 Commutation with the initial object

The next lemma implies the well-known fact that the initial model of type theory is parametric.

Lemma 30. *The forgetful functor:*

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \tag{140}$$

commutes with initial objects.

Proof. We consider the initial object $\mathcal{I}_T$ in Alg_T. By definition of an interpretation, we can define operations $\_^*$ on $\mathcal{I}_T$ which obey the unary equations added in T', so that $(\mathcal{I}_T, \_^*) : \mathrm{Alg}_{T'}$.

Now it is enough to prove that $(\mathcal{I}_T, \_^*)$ is initial in $\mathrm{Alg}_{T'}$, so that:

$$U(\mathcal{I}_{T'}) = U(\mathcal{I}_T, \_^*) = \mathcal{I}_T \tag{141}$$

To this end, we show that the unique morphism:

$$\psi : \mathrm{Hom}_{\mathrm{Alg}_T}(\mathcal{I}_T, U(\mathcal{C})) \tag{142}$$

for any $\mathcal{C} : \mathrm{Alg}_{T'}$ commutes with $\_^*$. So we prove by induction on $\mathcal{I}_T$ that $\psi(x^*) = \psi(x)^*$ for any x.

Indeed we have:

$$\psi(c\langle x_j\rangle^*) \overset{d}{=} \psi(c^*\langle x_j, x_j^*\rangle) \tag{143}$$
$$\overset{m}{=} c^*\langle \psi(x_j), \psi(x_j^*)\rangle \tag{144}$$
$$\overset{i}{=} c^*\langle \psi(x_j), \psi(x_j)^*\rangle \tag{145}$$
$$\overset{d}{=} (c\langle \psi(x_j)\rangle)^* \tag{146}$$
$$\overset{m}{=} \psi(c\langle x_j\rangle)^* \tag{147}$$

where $\overset{i}{=}$ indicates induction, $\overset{d}{=}$ definition of $\_^*$ and $\overset{m}{=}$ the fact that ψ is a morphism in Alg_T.

From this we can conclude that:

$$\psi : \mathrm{Hom}_{\mathrm{Alg}_{T'}}((\mathcal{I}_T, \_^*), \mathcal{C}) \tag{148}$$

and it is the unique such morphism by initiality of $\mathcal{I}_T$. $\square$

4.2   Commutation with pushouts

The next lemma uses in a crucial way the hypothesis that operations added in an interpretation are unary.

Lemma 31. *The forgetful functor:*

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \tag{149}$$

commutes with pushouts.

Proof. Given a span in $\mathrm{Alg}_{T'}$:

$$\mathcal{C}_1 \overset{f_1}{\leftarrow} \mathcal{D} \overset{f_2}{\to} \mathcal{C}_2 \tag{150}$$

our goal is to define operations $\_^*$ on the pushout:

$$\mathcal{C} = U(\mathcal{C}_1) \coprod_{U(\mathcal{D})} U(\mathcal{C}_2) \tag{151}$$

and prove that $\mathcal{C}$ equipped with these operations is a pushout in $\mathrm{Alg}_{T'}$.

The object $\mathcal{C}$ is generated by:

- The constructors in T (as for $\mathcal{I}_T$ in the previous lemma).

- For $\epsilon = 1, 2$ morphisms:

$$p_\epsilon : \operatorname{Hom}_{\operatorname{Alg}_T}(U(\mathcal{C}_\epsilon), \mathcal{C}) \tag{152}$$

  meaning that we have constructors p_ϵ for contexts, types, terms and substitutions, as well as equations for p_ϵ commuting with any constructor in T.

- For any x in $\mathcal{D}$ we have:

$$p_1(f_1(x)) = p_2(f_2(x)) \tag{153}$$

First we define $\_^*$ recursively on $\mathcal{C}$. For the constructors of T we proceed as for $\mathcal{I}_T$, for the new constructors we define:

$$p_\epsilon(x)^* = p_\epsilon(x^*) \tag{154}$$

This definition makes sense only for unary operations.

We need to check this preserves the equations. For equations in T this is checked in the Appendix, and we see for equations on p_ϵ that:

$$p_\epsilon(c\langle x_j \rangle)^* \overset{d}{=} p_\epsilon(c\langle x_j \rangle^*) \tag{155}$$
$$\overset{d}{=} p_\epsilon(c^* \langle x_j, x_j^* \rangle) \tag{156}$$
$$\overset{m}{=} c^* \langle p_\epsilon(x_j), p_\epsilon(x_j^*) \rangle \tag{157}$$
$$\overset{d}{=} c^* \langle p_\epsilon(x_j), p_\epsilon(x_j)^* \rangle \tag{158}$$
$$\overset{d}{=} c\langle p_\epsilon(x_j) \rangle^* \tag{159}$$

where $\overset{d}{=}$ indicates the definition of $\_^*$ and $\overset{m}{=}$ the fact that p_ϵ are morphisms in Alg_T. Moreover for x in $\mathcal{D}$:

$$p_1(f_1(x))^* \overset{d}{=} p_1(f_1(x)^*) \tag{160}$$
$$\overset{f}{=} p_1(f_1(x^*)) \tag{161}$$
$$= p_2(f_2(x^*)) \tag{162}$$
$$\overset{f}{=} p_2(f_2(x)^*) \tag{163}$$
$$\overset{d}{=} p_2(f_2(x))^* \tag{164}$$

where $\overset{d}{=}$ indicates the definition of $\_^*$ and $\overset{f}{=}$ comes from the fact that f_ϵ is a morphism in $\operatorname{Alg}_{T'}$. So we have defined the operations $\_^*$.

Next we check that any x in $\mathcal{C}$ obeys the unary equations added in T'. We proceed inductively on x. When x is constructed from T we use the hypothesis that equations are inductively proven in the initial model. When x is of the form $p_\epsilon(x')$ we use the fact that equations added in T' are true in $\mathcal{C}_\epsilon$ so they are true for x', together with the fact that p_ϵ is a morphism so it preserves equations.

18

Now we have $(\mathcal{C}, \_^*) : \mathrm{Alg}_{T'}$, we want to check that it is a pushout. To do this it is enough to check that any morphism:

$$\psi : \mathrm{Hom}_{\mathrm{Alg}_T}(U(\mathcal{C}_1) \coprod_{U(\mathcal{D})} U(\mathcal{C}_2), U(\mathcal{E})) \qquad (165)$$

defined from a commutative square in $\mathrm{Alg}_{T'}$:

$$
\begin{array}{ccc}
\mathcal{D} & \xrightarrow{f_1} & \mathcal{C}_1 \\
\downarrow{\scriptstyle f_2} & & \downarrow{\scriptstyle g_1} \\
\mathcal{C}_2 & \xrightarrow{g_2} & \mathcal{E}
\end{array}
$$

does commute with the operations $\_^*$ previously defined. By definition, ψ is such that:

$$\psi(p_\epsilon(x)) = g_\epsilon(x) \qquad (166)$$

We proceed inductively, using computations from the previous lemma, together with the following new case:

$$\psi(p_\epsilon(x)^*) = \psi(p_\epsilon(x^*)) \qquad (167)$$
$$= g_\epsilon(x^*) \qquad (168)$$
$$= g_\epsilon(x)^* \qquad (169)$$
$$= \psi(p_\epsilon(x))^* \qquad (170)$$

This concludes the proof. $\qquad \square$

4.3   Main theorem and applications

We are ready to give the most important result in this paper. Recall that we have assumed a forgetful functor:

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \qquad (171)$$

for T' an interpretation of T.

Theorem 3. *The forgetful functor:*

$$U : \mathrm{Alg}_{T'} \to \mathrm{Alg}_T \qquad (172)$$

has a right adjoint.

Proof. By Theorem 2 it is enough to show that U commutes with finite colimits. This is precisely the content of Lemmas 30 and 31. $\qquad \square$

Example 32. *We consider groups interpreting monoids. In this case the right adjoint is:*

$$C \; : \; \mathrm{Mon} \to \mathrm{Grp} \qquad (173)$$
$$C(M) = M^\times \qquad (174)$$

19

where $M^\times$ is the group of invertible elements in M. Indeed $\mathbb{Z}$ is the free group generated by $\{1\}$, so the underlying set of $C(M)$ is:

$$C(M) = \mathrm{Hom}_{\mathrm{Set}}(\{1\}, C(M)) \tag{175}$$

$$= \mathrm{Hom}_{\mathrm{Grp}}(\mathbb{Z}, C(M)) \tag{176}$$

$$= \mathrm{Hom}_{\mathrm{Mon}}(\mathbb{Z}, M) \tag{177}$$

$$= M^\times \tag{178}$$

The group structure is computed in the same way.

Example 33. *We consider the interpretation of graphs by reflexive graphs. Recall that reflexive graphs are given by the theory:*

$$V : \mathrm{Set} \tag{179}$$

$$E : V \to V \to \mathrm{Set} \tag{180}$$

$$r : (v : V) \to E\ v\ v \tag{181}$$

The right adjoint is:

$$C\ :\ \mathrm{Gph} \to \mathrm{rGph} \tag{182}$$

$$C(V, E) = (v : V) \times E\ v\ v,$$
$$(v, e)(v', e') \mapsto E\ v\ v',$$
$$(v, e) \mapsto e \tag{183}$$

To see this we consider $\mathcal{I}_c$ the free reflexive graph generated by $\{c\}$ a vertex. Then the set of vertices of $C(V, E)$ is:

$$C(V, E) = \mathrm{Hom}_{\mathrm{Set}}(\{c\}, C(V, E)) \tag{184}$$

$$= \mathrm{Hom}_{\mathrm{rGph}}(\mathcal{I}_c, C(V, E)) \tag{185}$$

$$= \mathrm{Hom}_{\mathrm{Gph}}(U(\mathcal{I}_c), (V, E)) \tag{186}$$

$$= (v : V) \times E\ v\ v \tag{187}$$

The rest of the structure is computed in the same way.

Example 34. *We consider the interpretation:*

$$X : \mathrm{Set} \tag{188}$$

$$s : X \to X \tag{189}$$

of the theory with $X : \mathrm{Set}$ alone. We denote its category of models by Set_s.

$$\mathrm{Set}_s = \{X : \mathrm{Set} \mid s : X \to X\} \tag{190}$$

Then the right adjoint is:

$$C\ :\ \mathrm{Set} \to \mathrm{Set}_s \tag{191}$$

$$C(X) = \mathbb{N} \to X,$$
$$f \mapsto (n \mapsto f(n + 1)) \tag{192}$$

To see this, note that $\mathbb{N}$ with the successor function is the free object in Set_s generated by $\{0\}$. Then the underlying set of $C(X)$ is:

$$C(X) = \mathrm{Hom}_{\mathrm{Set}}(\{0\}, C(X)) \tag{193}$$
$$= \mathrm{Hom}_{\mathrm{Set}_s}(\mathbb{N}, C(X)) \tag{194}$$
$$= \mathrm{Hom}_{\mathrm{Set}}(\mathbb{N}, X) \tag{195}$$

The function $f \mapsto (n \mapsto f(n+1))$ is computed in the same way.

These three examples should be contrasted with each other:

- In the first example, being invertible is a property of an element in a monoid (because there is at most one inverse). Then the right adjoint just needs to send a monoid to its group of invertible elements. This can be generalized to any unary property inductively provable, with the right adjoint sending an object to its subobject of elements obeying this property.

- In the second example, having an edge from v to v is really a structure on a vertex v (because there can be many such edges). So in this case we need to consider vertices together with a chosen edge in order to construct the right adjoint.

- The third example is the most interesting. Here having an image by s is clearly a structure, but to build the right adjoint it is not enough to require that any element comes with its image by s. Indeed this image should itself have an image, and so on. An iteration is taking place. This can be generalized to the interpretation of any theory T by the theory of T-algebras with an endomorphism.

Now we study our main example.

Example 35. *Consider the forgetful functor:*

$$U : \mathrm{Alg}_{\mathrm{pTT}} \to \mathrm{Alg}_{\mathrm{TT}} \tag{196}$$

from parametric models to models of type theory. By Theorems 1 and 3 it has a right adjoint C. Now we study $C(\mathcal{D})$ for $\mathcal{D}$ a model of type theory. We denote by $\mathcal{I}_X$ the free parametric model generated by an element $X : \mathrm{Ctx}$. Then:

$$\mathrm{Ctx}_{C(\mathcal{D})} = \mathrm{Hom}_{\mathrm{Set}}(\{X\}, \mathrm{Ctx}_{C(\mathcal{D})}) \tag{197}$$
$$= \mathrm{Hom}_{\mathrm{Alg}_{\mathrm{pTT}}}(\mathcal{I}_X, C(\mathcal{D})) \tag{198}$$
$$= \mathrm{Hom}_{\mathrm{Alg}_{\mathrm{TT}}}(U(\mathcal{I}_X), \mathcal{D}) \tag{199}$$

but reasoning as in Lemmas 30 and 31, we can prove that $U(\mathcal{I}_X)$ is isomorphic

to the free (non-parametric) model of type theory with:

$$X : \text{Ctx} \tag{200}$$

$$X^* : \text{Ty } X \tag{201}$$

$$X^{**} : \text{Ty } (x : X, X^*(x), X^*(x)) \tag{202}$$

$$\vdots$$

with the usual notation for contexts in a CwF. So giving a context in $C(\mathcal{D})$ is equivalent to giving:

$$\Gamma : \text{Ctx}_{\mathcal{D}} \tag{203}$$

$$\Gamma^* : \text{Ty}_{\mathcal{D}} \, \Gamma \tag{204}$$

$$\Gamma^{**} : \text{Ty}_{\mathcal{D}} \, (x : \Gamma, \Gamma^*(x), \Gamma^*(x)) \tag{205}$$

$$\vdots$$

We can get similar formulas for types, terms, and so on.

Remark 36. *The right adjoint C does not suffer from the same defect as the left adjoint in Example 24. We assume an empty type $\bot$ and we say that a model is inconsistent if its $\bot$ is inhabited. If $C(\mathcal{D})$ is inconsistent then so is $U(C(\mathcal{D}))$, and the counit:*

$$\epsilon : \text{Hom}_{\text{Alg}_{\text{TT}}}(U(C(\mathcal{D})), \mathcal{D}) \tag{206}$$

implies that $\mathcal{D}$ is inconsistent.

Remark 37. *Binary parametricity can be treated by our method. By following the last example, we see that a context in $C(\mathcal{D})$ would consist of:*

$$\Gamma : \text{Ctx}_{\mathcal{D}} \tag{207}$$

$$\Gamma^* : \text{Ty}_{\mathcal{D}} \, (\Gamma, \Gamma) \tag{208}$$

$$\begin{aligned} \Gamma^{**} : \text{Ty}_{\mathcal{D}} \, (&x_{00}, x_{01} : \Gamma, \Gamma^*(x_{00}, x_{01}), \\ &x_{10}, x_{11} : \Gamma, \Gamma^*(x_{10}, x_{11}), \\ &\Gamma^*(x_{00}, x_{10}), \Gamma^*(x_{01}, x_{11})) \end{aligned} \tag{209}$$

$$\vdots$$

So we are constructing the semi-cubical model in $\mathcal{D}$.

Remark 38. *It should be noted that our method does not immediately give an explicit definition for the three dots in Remarks 35 and 37. This lack of concreteness is compensated by some extra generality.*

5   Conclusion

In this paper we defined a procedure from interpretations of type theory to structures on types, outputting semi-cubical structure when given external parametricity. In order to do this we defined interpretations of any theory, and built a right adjoint from any such interpretation.

Our next work will be to apply this method to other interpretations of type theory. We would like to study forcing interpretations giving us some variant of presheaf models, and also the *univalent parametricity* from [25], hopefully allowing us to build definitionally univalent models (meaning models where $A =_\mathcal{U} B$ is definitionally equal to $A \simeq B$) from univalent models.

We would also like to make sense of the table already given in introduction:

Interpretation	Structure
External parametricity	Semi-cubical types
Internal parametricity	Cubical types
External univalence	Kan semi-cubical types
Internal univalence	Kan cubical types

This means that we need to find suitable interpretations. From a practical point of view this would give an efficient way to build univalent models of type theory, and might help to design variants of cubical type theory. From a conceptual point of view this could explain how the notion of Kan cubical structure can be deduced from the notion of equivalence.

We also believe this work shed some light on forgetful functors having both a left and a right adjoint. There is a large literature on forgetful functors having a right adjoint and coalgebras (see for example [2]), but we do not know any reference on such functors having a left adjoint as well. We believe these could be called unary functors by analogy with finitary functors. More precisely, we guess there is some kind of converse to Theorem 3, stating that the forgetful functor of an extension of theories has a right adjoint if and only if the extension obeys a condition weaker than being an interpretation, and stronger than having unary operations.

Acknowledgements

References

[1] *Homotopy type theory: univalent foundations of mathematics*. 2013.

[2] Jirí Adámek and Hans Porst. On varieties and covarieties in a category. *Mathematical Structures in Computer Science*, 13(2):201, 2003.

[3] Jiří Adámek and Jiří Rosický. *Locally presentable and accessible categories*. Cambridge University Press, 1994.

[4] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. *ACM SIGPLAN Notices*, 51(1):18–29, 2016.

[5] Thorsten Altenkirch and Ambrus Kaposi. Towards a cubical type theory without an interval. In *21st International Conference on Types for Proofs and Programs (TYPES 2015)*. Schloss Dagstuhl-Leibniz Zentrum für Informatik, 2018.

[6] Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 503–515, 2014.

[7] Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. A presheaf model of parametric type theory. *Electronic Notes in Theoretical Computer Science*, 319:67–82, 2015.

[8] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 345–356, 2010.

[9] Jean-Philippe Bernardy and Marc Lasson. Realizability and parametricity in pure type systems. In *International Conference on Foundations of Software Science and Computational Structures*, pages 108–122. Springer, 2011.

[10] Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128. Schloss Dagstuhl-Leibniz Zentrum für Informatik, 2014.

[11] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of pure and applied logic*, 32:209–243, 1986.

[12] Evan Cavallo and Robert Harper. Internal parametricity for cubical type theory. In *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[13] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. In *21st International Conference on Types for Proofs and Programs*, number 69, page 262. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015.

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

[15] Christian Dzierzon. *Essentially Algebraic Descriptions of Locally Presentable Categories*. PhD thesis, Staats und Universitätsbibliothek [Host], 2005.

[16] Neil Ghani, Fredrik Nordvall Forsberg, and Federico Orsanigo. Proof-relevant parametricity. In *A List of Successes That Can Change the World*, pages 109–131. Springer, 2016.

[17] Patricia Johann and Kristina Sojakova. Cubical categories for higher-dimensional parametricity. *arXiv preprint arXiv:1701.06244*, 2017.

[18] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–24, 2019.

[19] András Kovács and Ambrus Kaposi. Signatures and induction principles for higher inductive-inductive types. *Logical Methods in Computer Science*, 16, 2020.

[20] F William Lawvere. *Functorial Semantics of Algebraic Theories: And, Some Algebraic Problems in the Context of Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963.

[21] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*. Bibliopolis, Naples, 1984.

[22] John Reynolds. Types, abstraction and parametric polymorphism. In *Proceedings of the IFIP 9th World Computer Congress*, pages 513–523, 1983.

[23] Michael Shulman. All $(\infty, 1)$-toposes have strict univalent universes. *arXiv preprint arXiv:1904.07004*, 2019.

[24] Kristina Sojakova and Patricia Johann. A general framework for relational parametricity. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 869–878, 2018.

[25] Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. Equivalences for free: univalent parametricity for effective transport. *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–29, 2018.

[26] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical Agda: a dependently typed programming language with univalence and higher inductive types. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–29, 2019.

[27] Philip Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, pages 347–359, 1989.