Submitted to MFPS 2024

Parametricity via Cohesion

C.B. Aberlé

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

Parametricity is a key metatheoretic property of type systems, which implies strong uniformity properties of the structure of types within systems possessing it. In recent years, various systems of dependent type theory have emerged with the aim of expressing such parametric reasoning in their internal logic. More recently still, such internal parametricity has been applied to solving various problems arising from the complexity of higher-dimensional *coherence conditions* in type theory. This paper presents a first step toward the unification, simplification, and extension of these methods for internalizing parametricity. Specifically, I argue that there is an essentially modal aspect of parametricity, intimately connected with the category-theoretic concept of *cohesion.* On this basis, I describe a general categorical semantics for modal parametricity, develop a corresponding framework of axioms (with computational interpretations) in dependent type theory, and demonstrate the practical utility of these axioms in managing the complexity of higher-dimension coherence by using them to painlessly derive induction principles for higher inductive types. In closing, I sketch the outlines of a more general *synthetic* theory of parametricity, with applications in domains ranging from homotopy type theory to the analysis of program modules.

Keywords: parametricity, cohesion, type theory, category theory, homotopy type theory, constructive mathematics, agda.

1 The Past, Present & Future of Internal Parametricity

Reynolds [Rey83] began his seminal introduction of the concept of *parametricity* with the declaration that “type structure is a syntactic discipline for enforcing levels of abstraction.” In the ensuing decades, this idea of Reynolds' has been overwhelmingly vindicated by the success of type systems in achieving modularity and abstraction in domains ranging from programming to interactive theorem proving. Yet much remains to be done in exploring the theoretical basis for and ramifications of this idea, and Reynolds' particular solution to its formalization – the concept of *relational parametricity.*

Reynolds' original analysis of parametricity targeted the polymorphic λ-calculus (aka System F). Intuitively, polymorphic programs in System F cannot inspect the types over which they are defined and so must behave essentially the same for all types at which they are instantiated. To make this intuitive idea precise, Reynolds posed an ingenious solution in terms of logical relations, whereby every System F type is equipped with a suitable binary relation on its inhabitants, such that all definable terms in System F must preserve the relations defined on their component types. On this basis, Reynolds was able to establish many significant properties of the abstraction afforded by System F's type structure, e.g. all closed terms of type $\forall X.X \to X$ are extensionally equivalent to the identity function.

[1] Email: `caberle@andrew.cmu.edu`

Reynolds' original analysis of parametricity for System F remained wholly *external* to System F itself. However, with the advent of systems of dependent type theory capable of conducting such analysis of proofs and programs *internally,* the possibility arises of finding an appropriate set of axioms for dependent type theory that would allow it to prove such parametricity theorems of its own type structure. Such internalization could moreover have far-reaching benefits for the overall theory of parametricity. Classical accounts of parametricity for various type systems have generally been "analytic" in nature – defined in terms of the particular syntactic constructs and models of the systems they target. On the other hand, axioms for parametricity internal to dependent type theory could provide a "synthetic" basis for parametricity whereby any system satisfying these axioms would necessarily enjoy parametricity and its consequences.

The need for such an axiomatic specification of parametricity is made all the more apparent by recent developments in *Homotopy Type Theory* (HoTT) and related fields, wherein type theories with internal parametricity have been developed and applied by various authors to the solution of several open problems in these fields [CH20], [KS24]. Each such type theory has posed its own solution to the problem of internalizing parametricity, and although some commonalities exist between them, there is as yet no one framework that subsumes them all. In particular, the majority of these type theories have targeted only specific semantic models (usually some appropriately-chosen presheaf categories) rather than an axiomatically-defined class of models, and in this sense the analysis of parametricity offered by these type theories remains analytic, rather than synthetic. This limits any insight into whether and how these approaches to internal parametricity may be related to one another, generalized, or simplified.

As a first step toward the unification, simplification, and generalization of the various approaches to parametricity, I propose an analysis of parametricity in terms of the category-theoretic concept of *cohesion* (and, more specifically, Lawvere's concept of *sufficient cohesion*). Cohesion here refers to a particular relation between categories whereby a category is equipped with an adjoint string of modalities that together make its objects behave like *spaces,* whose points are bound together by an abstract property of *cohesion,* that constrains what maps may be defined between these spaces. Such constraint of the class of definable maps between objects is reminiscent of intuitive idea behind Reynolds' original formulation of parametricity, particularly if one interprets the cohesion that binds types together as the structure of relations that inhere between them. Moreover, by inspection of the categorical models of extant type theories with internal parametricity, along with classical models of parametricity, one finds that many of these (if not quite all) exhibit cohesion, in this sense. The main contribution of this paper is thus twofold: 1) to show that this basic setup of cohesion is essentially all that is needed to recover classical parametricity results internally in dependent type theory, and 2) to show how this axiomatic framework can be applied in solving problems arising from the complexity of *coherence conditions* in HoTT – specifically, the problem of deriving induction principles for higher inductive types from their recursors. As a further illustration and verification of this idea, this paper is also a literate Agda document wherein the axioms for and theorems resulting from such parametricity via cohesion have been fully formalized and checked for validity. [2]

```
{-# OPTIONS --rewriting --cohesion --flat-split --without-K #-}
module parametricity-via-cohesion where

open import Agda.Primitive
open import Data.Empty
open import Agda.Builtin.Unit
open import Agda.Builtin.Bool
open import Agda.Builtin.Sigma
open import Agda.Builtin.Equality
open import Agda.Builtin.Equality.Rewrite
```

[2] The full source of this file can be found at https://github.com/cbaberle/Parametricity-via-Cohesion.

2 Cohesion & Parametricity

The notion of *cohesion* as an abstract characterization of when one category (specifically a topos) behaves like a category of spaces defined over the objects of another, is due primarily to Lawvere [Law07,Law05]. The central concept of axiomatic cohesion is an arrangement of four adjoint functors as in the following diagram:

$$\begin{tikzcd} \mathcal{E} \arrow[d, bend right=60] \arrow[d, bend left=20] \arrow[d, phantom, "\Pi \dashv \Delta \dashv \Gamma \dashv \nabla"] \\ \mathcal{S} \arrow[u, bend left=20] \arrow[u, bend right=60] \end{tikzcd}$$

where $\mathcal{E}, \mathcal{S}$ are both topoi, Δ, ∇ are both fully faithful, and Π preserves finite products. Given such an arrangement, we think of the objects of $\mathcal{E}$ as *spaces* and those of $\mathcal{S}$ as *sets* (even if $\mathcal{S}$ is not the category of sets), where Γ is the functor that sends a space to its set of points, Δ sends a set to the corresponding *discrete* space, ∇ sends a set to the corresponding *codiscrete space*, and Π sends a space to its set of connected components. These in turn induce a string of adjoint modalities on $\mathcal{E}$:

$$\int \dashv \flat \dashv \sharp$$

where $\int = \Delta \circ \Pi$ and $\sharp = \nabla \circ \Gamma$ are idempotent monads, and $\flat = \Delta \circ \Gamma$ is an idempotent comonad.

A concrete example of cohesion comes from the category of reflexive graphs **RGph**, which is cohesive over the category of sets **Set** [Law05]. Here, Γ is the functor that sends a reflexive graph to its set of vertices, Δ sends a set V to the “discrete” reflexive graph on V whose only edges are self-loops, ∇ sends V to the “codiscrete” (i.e. complete) graph where there is a unique edge between any pair of vertices in V, and Π sends a reflexive graph to its set of (weakly) connected components. It is worth noting, at this point, that many classical models of parametricity (e.g. [Atk12]) are based upon semantic interpretation of type structure in terms of reflexive graphs. This, I wish to argue, is no accident, and the key property of reflexive graphs underlying such interpretations is precisely their cohesive structure. More generally, for any base topos $\mathcal{S}$, we may construct its corresponding topos $\mathbf{RGph}(\mathcal{S})$ of *internal reflexive graphs,* which will similarly be cohesive over $\mathcal{S}$. We can therefore in fact use the language of such internal reflexive graphs to derive parametricity results for *any* topos.

In fact, this same setup of cohesion is interpretable, *mutatis mutandis,* in the case where $\mathcal{E}, \mathcal{S}$ are not (1-)topoi, but rather ∞-topoi, i.e. models of homotopy type theory [Shu18]. This allows us to use the language of homotopy type theory – suitably extended with constructs for the above-described modalities (the $\flat$ modality in particular, which, for technical reasons, cannot be axiomatized directly in ordinary HoTT) – to work *synthetically* with the structure of such a cohesive ∞-topos. For present purposes, we accomplish this by working in Agda with the `--cohesion` flag enabled, along with `--without-K`, which ensures compatibility with the treatment of propositional equality in HoTT. [3]

I therefore begin by recalling some standard definitions from HoTT [Uni13], which shall be essential in defining much of the structure to follow. Essentially all of these definitions have to do with the identity type former `_≡_` and its associated constructor `refl : ∀ {ℓ} {A : Set ℓ} {a : A} → a ≡ a`, as defined in the module `Agda.Builtin.Equality`:

```
module hott where
```

First of all, we have the induction principle for the identity type, aka the J rule:

```
    J : ∀ {ℓ κ} {A : Set ℓ} {a : A}
        → (B : (b : A) → a ≡ b → Set κ)
```

[3] I also enable the `--rewriting` flag to allow specifying certain identities as rewrite rules in order to provide them with computational significance, and the `--flat-split` flag to allow pattern-matching over cohesive variables, which although not strictly necessary for the account of parametricity in what follows, is needed for some examples involving e.g. discreteness of the Booleans.

```
      → {b : A} → (p : a ≡ b) → B a refl → B b p
J B refl b = b
```

We then obtain the operation of *transport* as the recursor for the identity type:

```
transp : ∀ {ℓ κ} {A : Set ℓ} {a b : A}
         → (B : A → Set κ) → a ≡ b → B a → B b
transp B p b = J (λ a _ → B a) p b
```

Additionally, both `J` and `transp` are symmetric, and so can be applied "in the opposite direction":

```
J⁻¹ : ∀ {ℓ κ} {A : Set ℓ} {a : A}
      → (B : (b : A) → a ≡ b → Set κ)
      → {b : A} → (p : a ≡ b) → B b p → B a refl
J⁻¹ B refl b = b

transp⁻¹ : ∀ {ℓ κ} {A : Set ℓ} {a b : A}
           → (B : A → Set κ) → a ≡ b → B b → B a
transp⁻¹ B p b = J⁻¹ (λ a _ → B a) p b
```

Moreover, since all functions must preserve relations of identity, we may apply a function to both sides of an identification as follows:

```
ap : ∀ {ℓ κ} {A : Set ℓ} {B : Set κ} {a b : A}
     → (f : A → B) → a ≡ b → f a ≡ f b
ap f refl = refl
```

The notion of *contractibility* expresses the idea that a type is essentially uniquely inhabited.

```
isContr : ∀ {ℓ} (A : Set ℓ) → Set ℓ
isContr A = Σ A (λ a → (b : A) → a ≡ b)
```

Similarly, the notion of *equivalence* expresses the idea that a function between types has an essentially unique inverse.[4]

```
isEquiv : ∀ {ℓ κ} {A : Set ℓ} {B : Set κ}
          → (A → B) → Set (ℓ ⊔ κ)
isEquiv {A = A} {B = B} f =
    (b : B) → isContr (Σ A (λ a → f a ≡ b))

mkInv : ∀ {ℓ κ} {A : Set ℓ} {B : Set κ}
        → (f : A → B) → isEquiv f → B → A
mkInv f e b = fst (fst (e b))
```

```
open hott
```

The reader familiar with HoTT may note that, so far, I have not included anything relating to the *univalence axiom* – arguably the characteristic axiom of HoTT. In fact, this is by design, as a goal of the current formalization is to assume only axioms that can be given straightforward computational interpretations that preserve the property that every closed term of the ambient type theory evaluates to a *canonical normal form* (canonicity), so that these axioms give a *constructive* and *computationally sound* interpretation of parametricity. While the univalence axiom can be given a computational interpretation compatible with canonicity, as in Cubical Type Theory [CHS22], doing so is decidedly not a straightforward matter. Moreover, it turns out that the univalence axiom is largely unneeded in what follows, save for demonstrating admissibility of some additional axioms which permit more direct computational interpretations that are (conjecturally) compatible with canonicity. I thus shall have need for univalence only as a metatheoretic

[4] Those familiar with HoTT may note that I use the *contractible fibres* definition of equivalence, as this shall be the most convenient to work with, for present purposes.

assumption. In this setting, univalence allows us to convert equivalences between types into identifications of those types, which may then be transported over accordingly.

Having defined some essential structures of the language of HoTT, we may now proceed to similarly define some essential structures of the language of HoTT with *cohesive modalities.*

```
module cohesion where
```

Of principal importance here is the ♭ modality, which (intuitively) takes a type A to its corresponding *discretization* $\flat A$, wherein all cohesion between points as been eliminated. However, in order for this operation to be well-behaved, A must not depend upon any variables whose types are not themselves *discrete*, in this sense. To solve this problem, the `--cohesion` flag introduces a new form of variable binding `@♭ x : X`, which judgmentally asserts that `x` is an element of the discretization of `X`, and enforces the restriction that `X` may only depend upon other variables bound with `@♭`. In this case, we say that `x` is a *crisp* element of `X`.

This notion in hand, we may define the ♭ modality as an operation on *crisp* types:

```
    data ♭ {@♭ ℓ : Level} (@♭ A : Set ℓ) : Set ℓ where
        con : (@♭ x : A) → ♭ A
```

This modality is then equipped with the following *counit* map ϵ:[5]

```
    ϵ : {@♭ ℓ : Level} {@♭ A : Set ℓ} → ♭ A → A
    ϵ (con x) = x
```

A crisp type is then *discrete* precisely when this map is an equivalence:

```
    isDiscrete : ∀ {@♭ ℓ : Level} → (@♭ A : Set ℓ) → Set ℓ
    isDiscrete {ℓ = ℓ} A = isEquiv (ϵ {ℓ} {A})
```

```
open cohesion
```

Beyond such notions of discreteness, etc., what more is required for the sake of parametricity is a way of detecting when the elements of a given type are related, or somehow bound together, by the cohesive structure of that type.

For this purpose, it is useful to take a geometric perspective upon cohesion, and correspondingly, parametricity. What we are after is essentially the *shape* of an abstract relation between points, and an object I in our cohesive topos $\mathcal{E}$ (correspondingly, a type in our type theory) which *classifies* this shape in other objects (types), in that maps $I \to A$ correspond to such abstract relations between points in A. In this case, the *shape* of an abstract relation may be considered as a *path*, i.e. two distinct points which are somehow *connected.* By way of concrete example, in the topos of reflexive graphs **RGph**, the role of a classifier for this shape is played by the "walking edge" graph $\bullet \to \bullet$, consisting of two points and a single non-identity (directed) edge. More generally, using the language of cohesion, we can capture this notion of an abstract line segment in the following axiomatic characterization of I:

I is an object of $\mathcal{E}$ that is *strictly bipointed* and *weakly connected.*

Unpacking the terms used in this characterization, we have the following:

- *Strictly bipointed* means that I is equipped with two elements $i_0, i_1 : I$, such that the proposition $(i_0 \equiv i_1) \to \bot$ (i.e. $i_0 \neq i_1$) holds in the internal language of $\mathcal{E}$.
- *Weakly connected* means that the unit map $\eta : I \to \int I$ is essentially constant, in that it factors through a contractible object/type. Intuitively, this says that the image of I in $\int I$ essentially consists of a single connected component.

Note that the above-given example of the walking edge graph straightforwardly satisfies both of these requirements, as it consists of two distinct vertices belonging to a single (weakly) connected component. I also note in passing that, if the assumption of weak connectedness is strengthened to *strong connectedness* –

[5] arising from the fact that, semantically, ♭ is an idempotent comonad on crisp types.

i.e. the object/type $\int I$ is itself contractible – then the existence of such an object I as above is equivalent to Lawvere's axiom of *sufficient cohesion* [Law07]. We might therefore refer to the conjunction of the above conditions as an axiom of *weak sufficient cohesion* for the ambient ∞-topos $\mathcal{E}$.

We can begin to formalize such *weak sufficient cohesion* in Agda by postulating a type I with two given elements i_0, i_1:

```
postulate
    I : Set₀
    i0 i1 : I
```

We could also, in principle, directly postulate the strict bipointedness of I, as an axiom having the form $i0 \equiv i1 \to \bot$. However, this is in fact unnecessary, as this axiom will instead follow from an equivalent formulation introduced in a following subsection.

On the other hand, we do not yet have the capability to postulate an axiom of weak connectedness as written above, since we have not yet formalized the $\int$ modality. We could do so, but again, it is in fact better for present purposes to rephrase this axiom to an equivalent form involving only the $\flat$ modality, which can be done as follows:

> A type A is weakly connected if and only if, for every discrete type B, any function $A \to B$ is essentially constant, in the sense of factoring through a contractible type.

To see that this equivalence holds: in one direction, assume that A is weakly connected. Then for any map $f : A \to B$, by the adjunction $\int \dashv \flat$ and discreteness of B, there exist maps $f_\flat : A \to \flat B$ and $f^\int : \int A \to B$, such that the following diagram commutes:

$$\begin{tikzcd}
A \arrow[r, "\eta"] \arrow[d, "f_\flat"'] \arrow[dr, "f"] & \int A \arrow[d, "f^\int"] \\
\flat B \arrow[r, "\epsilon"] & B
\end{tikzcd}$$

Then since by assumption η factors through a contractible type, so does f.

In the other direction, assume that every map $f : A \to B$ is essentially constant, for every discrete type B. Then in particular, the map $\eta : A \to \int A$ is essentially constant, since $\int A$ is discrete (as it lies in the image of the discretization functor Δ).

Hence the property of I being weakly connected can be expressed purely in terms of its relation to the discrete types. Specifically, if we think of maps $I \to A$ as abstract *relations* or *paths* between elements of A, then weak connectedness of I equivalently says that all paths between points of discrete types are constant.

In order to express this property in Agda, it shall therefore be prudent first to introduce some machinery for ergonomically handling paths, analogous to the definition of path types in Cubical Type Theory (as in e.g. [CCHM15], [ABC+21]).

2.1 Path Types

In principle, given $a, b : A$, we could define the type of paths from a to b in A as the type $\Sigma f : I \to A.(f\ i_0 = a) \times (f\ i_1 = b)$. However, experience with such a naïve formalization shows that it incurs a high cost of laborious transportations along identities that should be easy enough to infer automatically. Hence I instead follow the approach taken by Cubical Type Theory and related systems, and give an explicit axiomatization for *path types*, with corresponding rewrite rules to apply the associated equalities automatically:

```
postulate
    Path : ∀ {ℓ} (A : I → Set ℓ) (a0 : A i0) (a1 : A i1) → Set ℓ
```

The introduction rule for path types corresponds to *function abstraction.*

```
    pabs : ∀ {ℓ} {A : I → Set ℓ}
           → (f : (i : I) → A i) → Path A (f i0) (f i1)
```

and likewise, the elimination rule corresponds to *function application.*

```
    papp : ∀ {ℓ} {A : I → Set ℓ} {a0 : A i0} {a1 : A i1}
           → Path A a0 a1 → (i : I) → A i
```

We may then postulate the usual β-law as an identity for this type, along with special identities for application to i_0 and i_1. All of these are made into rewrite rules, allowing Agda to apply them automatically, and thus obviating the need for excessive use of transport. [6]

```
    pβ : ∀ {ℓ} {A : I → Set ℓ} (f : (i : I) → A i)
         → (i : I) → papp (pabs f) i ≡ f i
    {-# REWRITE pβ #-}

    papp0 : ∀ {ℓ} {A : I → Set ℓ} {a0 : A i0} {a1 : A i1}
            → (p : Path A a0 a1) → papp p i0 ≡ a0
    {-# REWRITE papp0 #-}

    papp1 : ∀ {ℓ} {A : I → Set ℓ} {a0 : A i0} {a1 : A i1}
            → (p : Path A a0 a1) → papp p i1 ≡ a1
    {-# REWRITE papp1 #-}
```

With this formalization of path types in hand, we can straightforwardly formalize the equivalent formulation of weak connectedness of I given above. For this purpose, we first define the map `idToPath` that takes an identification $a = b$ to a path from a to b:

```
idToPath : ∀ {ℓ} {A : Set ℓ} {a b : A}
           → a ≡ b → Path (λ _ → A) a b
idToPath {a = a} refl = pabs (λ _ → a)
```

A type A is *path-discrete* if for all $a, b : A$, the map `idToPath` is an equivalence.

```
isPathDiscrete : ∀ {ℓ} (A : Set ℓ) → Set ℓ
isPathDiscrete {ℓ = ℓ} A =
    {a b : A} → isEquiv (idToPath {ℓ} {A} {a} {b})
```

We then postulate the following axioms:

```
postulate
    pathConst1 : ∀ {@♭ ℓ : Level} {@♭ A : Set ℓ} {a b : A}
                 → isDiscrete A → (e : Path (λ _ → A) a b)
                 → Σ (a ≡ b) (λ p → idToPath p ≡ e)
    pathConst2 : ∀ {@♭ ℓ : Level} {@♭ A : Set ℓ} {a b : A}
                 → (dA : isDiscrete A) → (e : Path (λ _ → A) a b)
                 → (q : a ≡ b) → (r : idToPath q ≡ e)
                 → pathConst1 dA e ≡ (q , r)
```

which together imply that, if A is discrete, then it is *path-discrete.*

```
isDisc→isPDisc : ∀ {@♭ ℓ : Level} {@♭ A : Set ℓ}
                 → isDiscrete A → isPathDiscrete A
isDisc→isPDisc dA e =
    (pathConst1 dA e , λ (p , q) → pathConst2 dA e p q)
```

As it stands, we have not yet given a procedure for evaluating the axioms `pathConst1` and `pathConst2` when they are applied to canonical forms, which means that computation on these terms will generally get

[6] We could additionally postulate an η-law for path types, analogous to the usual η-law for functions types; however, this is unnecessary for what follows, so I omit this assumption.

stuck and thus violate canonicity. Toward rectifying this, I prove a key identity regarding these axioms, add a further postulate asserting that this identity is equal to `refl`, and convert both of these to rewrite rules:

```
rwPathConst1 : ∀ {@♭ ℓ : Level} {@♭ A : Set ℓ}
               → {a : A} → (dA : isDiscrete A)
               → pathConst1 dA (pabs (λ _ → a)) ≡ (refl , refl)
rwPathConst1 {a = a} dA = pathConst2 dA (pabs (λ _ → a)) refl refl
{-# REWRITE rwPathConst1 #-}

postulate
    rwPathConst2 : ∀ {@♭ ℓ : Level} {@♭ A : Set ℓ}
                   → {a : A} → (dA : isDiscrete A)
                   → pathConst2 dA (pabs (λ _ → a)) refl refl ≡ refl
    {-# REWRITE rwPathConst2 #-}
```

Although a full proof of canonicity is beyond the scope of this paper, I conjecture that adding these rules suffices to preserve canonicity. [7] I shall check a few specific cases of this conjecture later in the paper.

So much for the (weak) connectedness of I. Let us turn our attention now to the other property previously stipulated of I, namely *strict bipointedness*. As mentioned previously, we could simply postulate this stipulation directly as an axiom – however, for the purpose of proving parametricity theorems, a more prudent strategy is to instead formalize a class of I-indexed type families, whose computational behavior follows from this assumption (and which, in turn, implies it). Because these type families essentially correspond to the *graphs* of predicates and relations on arbitrary types, I refer to them as *graph types*.

2.2 Graph Types

For present purposes, we need only concern ourselves with the simplest class of graph types: *unary graph types*, which, as the name would imply, correspond to graphs of unary predicates. Given a type A, a type family $B : A \to \mathsf{Type}$, and an element $i : I$, the graph type $\mathsf{Gph}^1\ i\ A\ B$ is defined to be equal to A when i is i_0, and equivalent to $\Sigma x : A.Bx$ when i is i_1. Intuitively, an element of $\mathsf{Gph}^1\ i\ A\ B$ is a dependent pair whose second element only exists when i is equal to i_1. We may formalize this in Agda as follows, by postulating a rewrite rule that evaluates $\mathsf{Gph}^1\ i_0\ A\ B$ to A:

```
postulate
    Gph1 : ∀ {ℓ} (i : I) (A : Set ℓ) (B : A → Set ℓ) → Set ℓ

    g1rw0 : ∀ {ℓ} (A : Set ℓ) (B : A → Set ℓ) → Gph1 i0 A B ≡ A
    {-# REWRITE g1rw0 #-}
```

We then have the following introduction rule for elements of $\mathsf{Gph}^1\ i\ A\ B$, which are pairs where the second element of the pair only exists under the assumption that $i = i_1$. When $i = i_0$ instead, the pair collapses to its first element:

```
    g1pair : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ} (i : I)
             → (a : A) → (b : (i ≡ i1) → B a) → Gph1 i A B

    g1pair0 : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ}
              → (a : A) → (b : (i0 ≡ i1) → B a)
              → g1pair {B = B} i0 a b ≡ a
    {-# REWRITE g1pair0 #-}
```

The first projection from such a pair may be taken no matter what i is, and reduces to the identity function when i is i_0.

```
    g1fst : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ} (i : I)
            → (g : Gph1 i A B) → A
```

[7] perhaps not for Agda specifically, but for a more idealized system of type theory, such as Martin-Löf Type Theory

```
g1β1 : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ} (i : I)
          → (a : A) → (b : (i ≡ i1) → B a)
          → g1fst i (g1pair {B = B} i a b) ≡ a
{-# REWRITE g1β1 #-}

g1fst0 : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ}
         → (g : Gph1 i0 A B) → g1fst {B = B} i0 g ≡ g
{-# REWRITE g1fst0 #-}
```

The second projection, meanwhile, may only be taken when i is equal to i_1:

```
g1snd : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ}
        → (g : Gph1 i1 A B) → B (g1fst i1 g)

g1β2 : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ}
       → (a : A) → (b : (i1 ≡ i1) → B a)
       → g1snd (g1pair {B = B} i1 a b) ≡ b refl
{-# REWRITE g1β2 #-}
```

It is straightforward to see that the inclusion of graph types makes strict bipointedness of I provable, as follows:

```
strBpt : i0 ≡ i1 → ⊥
strBpt p = g1snd (transp (λ i → Gph1 i ⊤ (λ _ → ⊥)) p tt)
```

In fact, the converse holds under the assumption of univalence. Specifically, in the presence of univalence and the assumption of strict bipointedness for I, the type $\mathsf{Gph}^1\ i\ A\ B$ may be regarded as a computationally convenient shorthand for the type $\Sigma x : A.(i = i1) \to Bx$, in much the same way as the type $\mathsf{Path}\ A\ a_0\ a_1$ serves as shorthand for the type $\Sigma f : (\Pi i : I.Ai).(f\ i_0 = a_0) \times (f\ i_1 = a_1)$. This is due to the following equivalence:

$$\begin{aligned}
&\Sigma x : A.(i_0 = i_1) \to Bx\\
\simeq\ &\Sigma x : A.\bot \to Bx\\
\simeq\ &\Sigma x : A.\top\\
\simeq\ &A
\end{aligned}$$

which is given by the map $\mathsf{fst} : (\Sigma x : A.(i_0 = i_1) \to Bx) \to A$ and which, under univalence, becomes an identity between its domain and codomain, thereby justifying the use of this and associated identities as axioms for graph types. I conjecture that moreover, taking these identities as rewrite rules is fully compatible with canonicity.[8]

A few useful theorems concerning identities between elements of graph types are as follows, the latter of which is also made into a rewrite rule:

```
apg1pair : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ}
           → {a b : A} {aB : B a} {bB : B b}
           → (p : a ≡ b) → aB ≡ transp⁻¹ B p bB
           → (i : I) → g1pair i a (λ _ → aB) ≡ g1pair i b (λ _ → bB)
apg1pair refl refl i = refl

apg1pair0 : ∀ {ℓ} {A : Set ℓ} {B : A → Set ℓ}
            → {a b : A} {aB : B a} {bB : B b}
```

[8] subject to the same caveat as before that this means canonicity for an idealized system of MLTT, and not Agda specifically

```
           → (p : a ≡ b) → (q : aB ≡ transp⁻¹ B p bB)
           → apg1pair p q i0 ≡ p
apg1pair0 refl refl = refl
{-# REWRITE apg1pair0 #-}
```

In principle, we could continue in this manner, and define graph types for relations of arities greater than 1 as well. However, unary graph types are sufficient for what follows. Having thus given appropriate axioms (and corresponding computation rules) to capture the desiderata that I be strictly bipointed and weakly connected, we are now in a position to prove some classical parametricity theorems using this structure.

2.3 Parametricity via Sufficient Cohesion

I begin this section with an old chestnut of parametricity theorems – a proof that any *polymorphic function* of type `(X : Set) → X → X` must act as the identity function on any *path-discrete* type `X`.

```
PolyId : (ℓ : Level) → Set (lsuc ℓ)
PolyId ℓ = (X : Set ℓ) → X → X
```

To prove the desired parametricity theorem for the type `PolyId` as above, we first prove a *substitution lemma* for this type, which can be stated as follows:

> For any function `α : PolyId` and any path-discrete type `A` with `a : A` and a type family `B : A → Set`, there is a function `B a → B (α A a)`.

The proof of this lemma is contained in the following submodule:

```
module paramId {ℓ} (A : Set ℓ) (pdA : isPathDiscrete A) (B : A → Set ℓ)
               (a : A) (b : B a) (α : PolyId ℓ) where
```

The key step in the proof of this lemma is to construct a *dependent path* over the type family `Gph1 i A B : I → Set` as follows:

```
    lemma0 : (i : I) → Gph1 i A B
    lemma0 i = α (Gph1 i A B) (g1pair i a (λ _ → b))
```

Then taking the second projection of `lemma0` evaluated at `i1` yields an element of `B` evaluated at the first projection of `lemma0` evaluated at `i1`:

```
    lemma1 : B (g1fst i1 (lemma0 i1))
    lemma1 = g1snd (lemma0 i1)
```

And we can use lemma0 to construct a path from `α A a` to `g1fst i1 (lemma0 i1)` as follows:

```
    lemma2 : Path (λ _ → A) (α A a) (g1fst i1 (lemma0 i1))
    lemma2 = pabs (λ i → g1fst i (lemma0 i))
```

And then since `A` is path-discrete, this path becomes an identity along which we can transport `lemma1`:

```
    substLemma : B (α A a)
    substLemma = transp⁻¹ B (mkInv idToPath pdA lemma2) lemma1
```

From this substitution lemma, it straightforwardly follows that any element of `PolyId` must be extensionally equivalent to the identity function when evaluated at a path-discrete type.

```
polyId : ∀ {ℓ} (A : Set ℓ) (pdA : isPathDiscrete A) (a : A)
         → (α : PolyId ℓ) → α A a ≡ a
polyId A pdA a α =
    paramId.substLemma A pdA (λ b → b ≡ a) a refl α
```

So far, we have proved only that the restrictions of elements of `PolyId` to path-discrete types are equivalent to that of the polymorphic identity function on path-discrete types. The theorem would then after all be trivial if it turned out that the only path-discrete types were (e.g.) those containing at most one element (i.e. the *mere propositions,* in the terminology of HoTT). To show that this is not the case, we make use of our assumption of (weak) connectedness for I, which we have already seen implies that every discrete type is path-discrete. One may think of the discrete types as forming their own distinct system of type theory which is then *embedded* in the broader system of cohesive types we have been working in, via the ♭ modality. We can then use the cohesive structure of the latter system to analyze properties of the embedded theory. Moreover, the subuniverse of discrete types contains many specific nontrivial types to which these theorems may be applied, e.g. – as in the following example – the type of Booleans `Bool`.

```
module BoolDiscrete where
```

Showing that `Bool` is discrete is a simple matter of pattern matching.

```
    boolIsDisc : isDiscrete Bool
    boolIsDisc false = (con false , refl) , λ { (con false , refl) → refl }
    boolIsDisc true  = (con true  , refl) , λ { (con true  , refl) → refl }
```

It follows that `Bool` is also path-discrete and so the above parametricity theorem may be applied to it.

```
    boolIsPDisc : isPathDiscrete Bool
    boolIsPDisc = isDisc→isPDisc boolIsDisc

    polyIdBool : (α : PolyId lzero) → (b : Bool) → α Bool b ≡ b
    polyIdBool α b = polyId Bool boolIsPDisc b α
```

This provides an opportunity to check some simple cases of the canonicity conjecture for the rewrite rules we have so far postulated. Specifically, we may check that the proof of `polyId` yields a canonical form (namely `refl`) when applied to canonical forms involving Booleans:

```
    shouldBeRefl1 : true ≡ true
    shouldBeRefl1 = polyIdBool (λ X → λ x → x) true
```

Running Agda's normalization procedure on this term shows that it does indeed evaluate to `refl`.

2.4 Parametricity & (Higher) Inductive Types

The foregoing proof of parametricity for the type of the polymorphic identity function remains ultimately a toy example. To demonstrate the true power of this approach to parametricity, I turn now to some more intricate examples of its use in proving universal properties for simplified representations of inductive types and higher inductive types.

In general, it is easy to write down what should be the recursion principle for an inductive type generated by a set of constructors, but harder (though feasible) to write down the corresponding induction principle. When one begins to consider more complex generalizations of inductive types, such as higher inductive types, inductive-inductive types, etc., these difficulties begin to multiply.

What would be ideal would be a way of deriving the induction principle for an inductive type from its recursor, hence requiring only the latter to be specified as part of the primitive data of the inductive type. However, in most systems of ordinary dependent type theory this is generally not possible [Geu01]. In HoTT, there is one way around this issue, due to Awodey, Frey & Speight [AFS18], whereby additional *naturality* constraints are imposed upon the would-be inductive type, that serve to make the corresponding induction principle derivable from the recursor. However, when one goes on to apply this technique to *higher inductive types,* which may specify constructs not only for elements of a type, but also for instances of its (higher) identity types, the complexity of these naturality conditions renders them impractical to work with. The ballooning complexity of these conditions is an instance of the infamous *coherence problem* in HoTT, whereby the complexity of coherence conditions for higher-categorical structures seemingly escapes any simple inductive definition.

As an alternative, let us consider ways of using parametricity to derive induction principles from recursion principles for inductive and higher inductive types, starting with the prototypical inductive type – the natural numbers $\mathbb{N}$.

First, we define the type of the recursor for $\mathbb{N}$.

```
Recℕ : Setω
Recℕ = ∀ {ℓ} (A : Set ℓ) → A → (A → A) → A
```

We may then postulate the usual constructors and identities for ℕ.[9]

```
postulate
    ℕ : Set₀
    zero : ℕ
    succ : ℕ → ℕ
    recℕ : ℕ → Recℕ
    zeroβ : ∀ {ℓ} (A : Set ℓ) (a : A) (f : A → A)
            → recℕ zero A a f ≡ a
    {-# REWRITE zeroβ #-}
    succβ : ∀ {ℓ} (n : ℕ) (A : Set ℓ) (a : A) (f : A → A)
            → recℕ (succ n) A a f ≡ f (recℕ n A a f)
    {-# REWRITE succβ #-}
    ℕη : (n : ℕ) → recℕ n ℕ zero succ ≡ n
    {-# REWRITE ℕη #-}
```

From here, we may proceed as in the proof of parametricity for `PolyId`, by proving an analogous substitution lemma for `Recℕ`, following essentially the same steps:

```
module paramℕ {ℓ} (α : Recℕ) (A : Set ℓ) (pdA : isPathDiscrete A)
                  (B : A → Set ℓ) (a : A) (b : B a)
                  (f : A → A) (ff : (x : A) → B x → B (f x)) where

    lemma0 : (i : I) → Gph1 i A B
    lemma0 i = α (Gph1 i A B)
                 (g1pair i a (λ _ → b))
                 (λ g → let g' j q =
                                transp (λ k → Gph1 k A B) q g in
                     g1pair i (f (g1fst i g))
                              (λ p →
                                J⁻¹ (λ j q →
                                       B (f (g1fst j (g' j q))))
                                    p
                                    (ff (g1fst i1 (g' i1 p))
                                        (g1snd (g' i1 p)))))

    lemma1 : B (g1fst i1 (lemma0 i1))
    lemma1 = g1snd (lemma0 i1)

    lemma2 : Path (λ _ → A) (α A a f) (g1fst i1 (lemma0 i1))
    lemma2 = pabs (λ i → g1fst i (lemma0 i))

    substLemma : B (α A a f)
    substLemma = transp⁻¹ B (mkInv idToPath pdA lemma2) lemma1
```

In order to apply this lemma to ℕ itself, we must further postulate that ℕ is path-discrete.[10]

```
postulate
```

[9] Note that among the identities postulated as rewrite rules for ℕ is its η-law, i.e. $\mathsf{rec}\mathbb{N}\ n\ \mathbb{N}\ \mathsf{zero}\ \mathsf{succ} = n$ for all $n : \mathbb{N}$. This identity will be important for the following derivation of the induction principle for ℕ, and postulating it as a rewrite rule, rather than an axiom, is thus necessary to preserve canonicity.

[10] One might think that we could show by induction that ℕ is discrete, hence path-discrete by weak connectedness of I; however, since we have not yet proven induction for the above-given encoding of ℕ internally, we must instead take path-discreteness of ℕ as an additional axiom, from which induction on ℕ will follow.

```
    pdℕ1 : ∀ {m n : ℕ} (e : Path (λ _ → ℕ) m n)
         → Σ (m ≡ n) (λ p → idToPath p ≡ e)
    pdℕ2 : ∀ {m n : ℕ} (e : Path (λ _ → ℕ) m n)
         → (q : m ≡ n) (r : idToPath q ≡ e)
         → pdℕ1 e ≡ (q , r)

pdℕ : isPathDiscrete ℕ
pdℕ e = (pdℕ1 e , λ (q , r) → pdℕ2 e q r)

rwPDℕ1 : (n : ℕ) → pdℕ1 (pabs (λ _ → n)) ≡ (refl , refl)
rwPDℕ1 n = pdℕ2 (pabs (λ _ → n)) refl refl
{-# REWRITE rwPDℕ1 #-}

postulate
    rwPDℕ2 : (n : ℕ) → pdℕ2 (pabs (λ _ → n)) refl refl ≡ refl
    {-# REWRITE rwPDℕ2 #-}
```

Induction for ℕ then follows as a straightforward consequence of the substitution lemma. [11]

```
indℕ : (P : ℕ → Set)
     → P zero
     → ((n : ℕ) → P n → P (succ n))
     → (n : ℕ) → P n
indℕ P pz ps n =
    paramℕ.substLemma (recℕ n) ℕ pdℕ P zero pz succ ps
```

As in the case of the parametricity theorem for `PolyId`, we may test that the derived induction principle for ℕ evaluates canonical forms to canonical forms. The following example tests this for the usual inductive proof that `zero` is an identity element for addition on the right (when addition is defined by induction on the left argument).

```
module ℕexample where
    _plus_ : ℕ → ℕ → ℕ
    m plus n = recℕ m ℕ n succ

    zeroIdR : (n : ℕ) → n plus zero ≡ n
    zeroIdR n =
        indℕ (λ m → m plus zero ≡ m) refl (λ m p → ap succ p) n

    shouldBeRefl2 : succ (succ zero) ≡ succ (succ zero)
    shouldBeRefl2 = zeroIdR (succ (succ zero))
```

Running Agda's normalization procedure on `shouldBeRefl2` again confirms that it evaluates to `refl`.

Moving on from inductive types to *higher* inductive types, we may now consider deriving the induction principle for the circle S^1, defined to be the type freely generated by a single basepoint $\mathsf{base} : S^1$, with a nontrivial identification $\mathsf{loop} : \mathsf{base} = \mathsf{base}$. The recursor for S^1 thus has the following type:

```
RecS¹ : Setω
RecS¹ = ∀ {ℓ} (A : Set ℓ) → (a : A) → a ≡ a → A
```

Then, just as before, we may postulate the corresponding constructors and $\beta\eta$-laws for S^1.

```
postulate
    S¹ : Set₀
    base : S¹
```

[11] Note that our use of the η-law for ℕ as a rewrite rule is critical for the last step of this proof, since otherwise we would obtain not a proof of `P n`, but rather `P (recℕ n ℕ succ zero)`.

```
    loop : base ≡ base
    recS¹ : S¹ → RecS¹
    baseβ : ∀ {ℓ} (A : Set ℓ) (a : A) (l : a ≡ a)
            → recS¹ base A a l ≡ a
    {-# REWRITE baseβ #-}
    loopβ : ∀ {ℓ} (A : Set ℓ) (a : A) (l : a ≡ a)
            → ap (λ s → recS¹ s A a l) loop ≡ l
    {-# REWRITE loopβ #-}
    S¹η : (s : S¹) → recS¹ s S¹ base loop ≡ s
    {-# REWRITE S¹η #-}
```

The proof of induction for S^1 is then in essentials the same as the one given above for ℕ. We begin by proving a substitution lemma for `RecS¹`, following exactly the same steps as in the proof of the corresponding lemma for `Recℕ`.

```
module paramS¹ {ℓ} (A : Set ℓ) (pdA : isPathDiscrete A) (B : A → Set ℓ)
                   (a : A) (b : B a) (l : a ≡ a)
                   (lB : b ≡ transp⁻¹ B l b) (α : RecS¹) where

    lemma0 : (i : I) → Gph1 i A B
    lemma0 i = α (Gph1 i A B) (g1pair i a (λ _ → b)) (apg1pair l lB i)

    lemma1 : B (g1fst i1 (lemma0 i1))
    lemma1 = g1snd (lemma0 i1)

    lemma2 : Path (λ _ → A) (α A a l) (g1fst i1 (lemma0 i1))
    lemma2 = pabs (λ i → g1fst i (lemma0 i))

    substLemma : B (α A a l)
    substLemma = transp⁻¹ B (mkInv idToPath pdA lemma2) lemma1
```

We then postulate that S^1 is path-discrete, as before, in order to apply this lemma to S^1 itself.

```
postulate
    pdS¹1 : ∀ {s t : S¹} (e : Path (λ _ → S¹) s t)
            → Σ (s ≡ t) (λ p → idToPath p ≡ e)
    pdS¹2 : ∀ {s t : S¹} (e : Path (λ _ → S¹) s t)
            → (q : s ≡ t) (r : idToPath q ≡ e)
            → pdS¹1 e ≡ (q , r)

pdS¹ : isPathDiscrete S¹
pdS¹ e = (pdS¹1 e , λ (q , r) → pdS¹2 e q r)

rwPDS¹1 : (s : S¹) → pdS¹1 (pabs (λ _ → s)) ≡ (refl , refl)
rwPDS¹1 s = pdS¹2 (pabs (λ _ → s)) refl refl
{-# REWRITE rwPDS¹1 #-}

postulate
    rwPDS¹2 : (s : S¹) → pdS¹2 (pabs (λ _ → s)) refl refl ≡ refl
    {-# REWRITE rwPDS¹2 #-}
```

Then the desired induction principle for S^1 follows straightforwardly.

```
indS¹ : (P : S¹ → Set)
      → (pb : P base)
      → pb ≡ transp⁻¹ P loop pb
      → (s : S¹) → P s
```

```
indS¹ P pb pl s =
    paramS¹.substLemma S¹ pdS¹ P base pb loop pl (recS¹ s)
```

Although it is not in general possible to verify that this same construction is capable of deriving induction principles for *all* higher inductive types – essentially because there is as yet no well-established definition of what higher inductive types are *in general* – there appears to be no difficulty in extending this method of proof to all known classes of higher inductive types. Moreover, that the proof of induction for S^1 is essentially no more complex than that for $\mathbb{N}$ suggests that this method is capable of taming the complexity of coherences for such higher inductive types, and in this sense provides a solution to this instance of the coherence problem.

3 Toward a Synthetic Theory of Parametricity

The theory so-far developed gives the rudiments of a synthetic framework for working in the internal language of a (weakly) *sufficiently cohesive* ∞*-topos*. This framework in turn proves capable of deriving significant parametricity results internally, with immediate applications in e.g. resolving coherence problems having to do with higher inductive types. It remains to be seen what further applications can be developed for this theory and its particular approach to parametricity. From this perspective, it is profitable to survey what other approaches there are to internalizing parametricity theorems in dependent type theory, and how they might be related to the one given in this paper.

3.1 Cohesion & Gluing

In recent years, there has been some related work toward a synthetic theory of parametricity in terms of the topos-theoretic construction of *Artin Gluing.* This approach, outlined initially by Sterling in his thesis [Ste22] and subsequently spearheaded by Sterling and his collaborators as part of the more general programme of *Synthetic Tait Computability* (STC), works in the internal language of a topos equipped with two (mere) propositions L and R, that are *mutually exclusive* in that $L \wedge R \to \bot$. The central idea of this approach is to synthetically reconstruct the usual account of parametricity in terms of logical relations by careful use of the *open* and *closed* modalities induced by these propositions, which play a similar role in STC to the *graph types* introduced above. Using this approach, one can e.g. prove representation independence theorems for module signatures as in recent work by Sterling & Harper [SH21].

Clearly, there is some affinity between the approach to parametricity in terms of gluing and that in terms of cohesion presented above, not least of which having to do with the fact that they both offer characteristically *modal* perspectives on parametricity. Yet there is a further sense in which these two approaches are related, which is that every sufficiently cohesive topos contains a model of Sterling's setup of STC for parametricity, as follows: given a sufficiently cohesive topos $\mathcal{E}$ over some base topos $\mathcal{S}$, for any strictly bipointed object $I \in \mathcal{E}$ with distinguished points $i_0, i_1 : I$, the *slice topos* $\mathcal{E}/I$, whose internal language corresponds to that of $\mathcal{E}$ extended with an arbitrary element $i : I$, is thereby equipped with two mutually exclusive propositions, namely $i = i_0$ and $i = i_1$. Hence all of the parametricity theorems available in STC can be recovered in the above-given framework (in particular, instances of closed modalities can be encoded as higher inductive types, which, as we have just seen, are easily added to the above framework).

On the other hand, there is no analogue in STC of the axiom of (weak) connectedness for the interval, and its consequent parametricity theorems, most notably the above-mentioned derivability of induction principles for higher inductive types. This suggests that in these latter results, the structure of *sufficient cohesion* and its relation to parametricity plays an essential role. Nonetheless, it remains interesting to consider how parametricity via cohesion and parametricity via gluing may yet prove to be related, and one may hope for a fruitful cross-pollination between these two theories.

3.2 Cohesion & Coherence

As we have seen, there appears to be an intimate link between *parametricity*, *cohesion*, and *coherence*, demonstrated (e.g.) by the above-given proof of induction for S^1. The existence of such a link is further supported by other recent developments in HoTT and related fields. E.g. Cavallo & Harper [CH20] have used their system of internal parametricity for Cubical Type Theory to derive coherence theorems for the smash product. More recently still, Kolomatskaia & Shulman [KS24] have introduced their system of *Displayed* Type Theory, which utilizes yet another form of internal parametricity to solve the problem of representing *semi-simplicial types* in HoTT.

In outlining the above framework for parametricity in terms of cohesion, I hope to have taken a first step toward the unification of these various systems that use parametricity to tackle instances of the coherence problem. Cavallo & Harper's system is readily assimilated to this framework, as are other related systems that take a *cubical* approach to internal parametricity, such as that of Nuyts, Devriese & Vezzossi [NVD17], since these all take their semantics in various topoi of bicubical sets, which all are sufficiently cohesive over corresponding topoi of cubical sets. On the other hand, Kolomatskaia & Shulman's system cannot be assimilated in quite the same way, since their system takes its semantics in the ∞-topos of augmented semisimplicial spaces, which is *not* cohesive over spaces. It thus remains to be seen if the above framework can be generalized so as to be inclusive of this example as well. Alternatively, one might seek to generalize or modify Kolomatskaia & Shulman's system so as to be interpretable in an ∞-topos that *is* cohesive over spaces, e.g. the ∞-topos of simplicial spaces. If this latter proves feasible, then this in turn would reveal yet further connections between parametricity via cohesion and another prominent attempt at a solution to the coherence problem, namely Riehl & Shulman's *Simplicial Type Theory* [RS17], which takes its semantics in simplicial spaces.

It appears, in all these cases, that parametricity is *the* tool for the job of taming the complexity of higher coherences in HoTT and elsewhere. In this sense, Reynolds was right in thinking that parametricity captures a fundamental property of abstraction, for, as any type theorist worth their salt knows, abstraction is ultimately the best tool we have for managing complexity.

3.3 Acknowledgements

The origins of this paper trace to research I did as an undergraduate at Merton College, Oxford, in the Summer of 2020, as part of the Merton College Summer Projects Scheme. Naturally, this work was conducted at a time of considerable stress for myself and the world at large, and I am massively indebted to the academic support staff at Merton, who were immensely helpful in keeping me afloat at that time. I am particularly grateful to Katy Fifield, Jemma Underdown, and Jane Gover for the support they provided to me in the course of my undergraduate studies. More recently, I am thankful to Frank Pfenning and Steve Awodey for their encouragement in continuing to pursue this line of research, and to Tesla Zhang for many helpful tips on how to make the best use of Agda for this paper.

References

[ABC$^+$21] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Robert Harper, Kuen-Bang Hou (Favonia), and Daniel R. Licata. Syntax and models of cartesian cubical type theory. *Mathematical Structures in Computer Science*, 31(4):424–468, 2021.

[AFS18] Steve Awodey, Jonas Frey, and Sam Speight. Impredicative encodings of (higher) inductive types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18. ACM, July 2018.

[Atk12] Robert Atkey. Relational parametricity for higher kinds. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46–61, 2012.

[CCHM15] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: A constructive interpretation of the univalence axiom. *FLAP*, 4:3127–3170, 2015.

[CH20] Evan Cavallo and Robert Harper. Internal parametricity for cubical type theory. In *Annual Conference for Computer Science Logic*, 2020.

[CHS22] Thierry Coquand, Simon Huber, and Christian Sattler. Canonicity and homotopy canonicity for cubical type theory. *Logical Methods in Computer Science*, Volume 18, Issue 1, February 2022.

[Geu01] Herman Geuvers. Induction is not derivable in second order dependent type theory. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications*, pages 166–181, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[KS24] Astra Kolomatskaia and Michael Shulman. Displayed type theory and semi-simplicial types, 2024.

[Law05] F. William Lawvere. Categories of spaces may not be generalized spaces as exemplified by directed graphs. *Reprints in Theory and Applications of Categories*, (9):1–7, 2005.

[Law07] F. William Lawvere. Axiomatic cohesion. *Theory and Applications of Categories*, 19(3):41–49, 2007.

[NVD17] Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. Parametric quantifiers for dependent type theory. *Proc. ACM Program. Lang.*, 1(ICFP), August 2017.

[Rey83] John C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, 1983.

[RS17] Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *Higher Structures*, 1(1):116–193, 2017.

[SH21] Jonathan Sterling and Robert Harper. Logical relations as types: Proof-relevant parametricity for program modules. *Journal of the ACM*, 68(6):1–47, October 2021.

[Shu18] Michael Shulman. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *Mathematical Structures in Computer Science*, 28(6):856–941, 2018.

[Ste22] Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory.* PhD thesis, Carnegie Mellon University, 2022.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.