Modal Simplicial Type Theory in Agda

by

Samuel Toth

Supervised by Ulrik Buchholtz

Submitted in partial fulfilment of the conditions for the award of the degree of
BSc in Computer Science

I hereby declare that this dissertation is all my own work,
except as indicated in the text

Signature: ST

School of Computer Science
University of Nottingham

April 15, 2026

Abstract

We present a formalised library of simplicial homotopy type theory in the proof assistant Agda. This builds on the foundations developed in the original work on simplicial type theory by Riehl and Shulman [RS17]; however, unlike existing formalisations [KRW23], we follow the approach of Gratzer, Weinberger, and Buchholtz [GWB26a; GWB25] working pervasively only up to equivalence, as well as incorporating modal type theory in order to construct a non-trivial example of a synthetic category. To make this possible, a pair of prototype extensions to the Agda programming language have been developed. The latter extension, implementing that sharp modality of Shulman's Cohesive Type Theory [Shu18], has now been merged into the main branch of Agda. We discuss the key results in the library, in particular the formalisation of a directed univalent universe of Spaces [GWB26a].

Acknowledgements

I would like to thank Ulrik Buchholtz for his guidance and supervision throughout both the project and my undergraduate degree. I would also like to thank Fredrik Bakke for helpful conversions and guidance on both mathematical issues and on the development and maintenance of Agda libraries. I would also like to thank him for his contributions to the Agda-synthetic-categories library. Finally I would like to thank Mark Williams for his generous support and encouragement throughout my entire time at Nottingham, and also for his contributions to the library.

Errata

1. April 28th 2026: Fredrik Bakke informed me that the included proof of the fact that tiny objects are closed under retracts was incorrect. I have since removed this erroneous proof.

Contents

Abstract i

Acknowledgements ii

Errata iii

Contents iv

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Related Work . . . 3
- 1.3 Project Aims and Objectives . . . 4
- 1.4 Project Management . . . 5
- 1.5 Documentation . . . 5
- 1.6 Summary of Contributions . . . 6

2 Modal Type Theory 8
- 2.1 MTT . . . 9
- 2.2 Agda flat . . . 10
- 2.3 Cohesive type theory . . . 11

3 Simplicial Homotopy Type Theory 14
- 3.1 Formalising Homotopy Type Theory . . . 14
  - 3.1.1 Equivalances . . . 14
  - 3.1.2 Identity systems and encode-decode . . . 16
  - 3.1.3 Cartesian squares, cubes and orthogonality . . . 17
- 3.2 Algebra, simplical shapes and duality . . . 18
- 3.3 Simplicial Foundations . . . 19
  - 3.3.1 Comma and Cocomma Types . . . 20
  - 3.3.2 (Pre)categories . . . 21
  - 3.3.3 Fibrations . . . 22
- 3.4 Category Theory . . . 25
  - 3.4.1 Yoneda . . . 25
  - 3.4.2 Universal constructions . . . 25

3.5 Constructing $\mathcal{S}$ . . . 27
3.5.1 Dependent right adjoints . . . 29
3.5.2 Amazing covariance . . . 31
3.5.3 Directed univalence . . . 32

4 Conclusion **35**
4.1 Technical Evaluation . . . 35
4.1.1 Formalisation . . . 35
4.1.2 Modal extensions to Agda . . . 36
4.2 LSEPI . . . 37
4.3 Future outlook . . . 37

Bibliography **38**

Chapter 1

Introduction

This report will present a new Agda library and online learning resource focused on the development of simplicial homotopy type theory, an extension to homotopy type theory which enables the development of synthetic higher categories. We will discuss both the software and tooling that have been developed as part of this project as well as the mathematical ideas behind simplicial type theory. Type theory itself is a formal language in which mathematical statements can be encoded, and a proof assistant (such as Agda) is a piece of software that checks the validity of these statements; a process which often involves a translation from a higher level language written by the user, to a lower level language that can more easily be checked. Despite proof assistants performing this translation, when formalising a traditional written proof, one often has to provide arguments with a higher degree of detail and precision. On one hand, this can make the formalisation process slow and challenging, but on the other hand, it can force the development of higher-level abstractions and proof techniques that can in turn clarify the original informal mathematics. In this report we will both present the successes and challenges in formalising simplicial type theory in Agda, as well as present some novel mathematics and proofs that have been necessitated or clarified by the process of formalisation. We will first give a short introduction into the background and existing work in this area, together with laying out the aims and objective of this project. We will then discuss modal type theory, and the extensions of Agda that have been prototyped in order to use this theory. Finally, we discuss our formalisation of simplicial type theory, culminating with the novel formalisation of the directed univalent universe of spaces.

1.1 Background

This project naturally builds upon a number of different fields and ideas within mathematics and theoretical computer science. In this section we give a brief discussion and introduction of the most important related and background topics.

Homotopy Type Theory Intentional Martin Löf type theory (MLTT) is a formal system for developing mathematics. Historically, it was an open problem as to whether a certain technical axiom, axiom-K or uniqueness of identity proofs, was provable in MLTT with just the rules of identity types. An answer in the negative was found by Hofmann and Streicher [HS94], where a counter-model was given in which types are interpreted as groupoids and identity types are interpreted as paths in the groupoid, which are not necessarily unique as would be implied by axiom-K. This opened up possibilities of new interpretations of type theory. Around a similar time, fields medallist Vladimir Vovodsky was looking towards type theory from the point of view of algebraic topology and homotopy theory, where ∞-groupoids are a fundamental tool. Vovodsky, together with the group formed as part of a special year of study at the Institute of Advanced Study, made several key observations, including:

- Extending the groupoid model of HoTT to ∞-groupoids
- A correct generalisation of isomorphism for arbitrary (non-truncated) types - referred to as *equivalence*
- The importance and utility of the *univalence axiom* - which characterises the paths in universes of types
- The definition of important geometric objects such as the spheres via higher inductive types

This work culminated in the 'HoTT book' [Uni13] which is still a very useful reference and introduction to the field. Since then, Homotopy Type Theory, or the program of Univalent Foundations, has flourished into a field at the intersection of mathematics and theoretical computer science, giving a foundation where homotopical thinking is natural. At its essence, homotopical thinking is about caring not just whether some objects are equal but keeping track of the ways in which they are equal.

Synthetic mathematics Synthetic mathematics refers to the idea of working in a non-standard foundation where the objects of mathematical study exist primitively. This is in contrast to traditional or analytic mathematics where you build up objects of interest (e.g. groups, vector spaces, the real numbers) from more primitive objects such as sets or types. The practice of doing synthetic mathematics goes back surprisingly far – arguably originating with Euclid's axiomatic approach to geometry. More recently, the synthetic approach has been investigated in a variety of different fields, often (but not always) through the lens of type theory. These include synthetic differential geometry [Koc06], synthetic homotopy theory [Uni13], synthetic algebraic geometry [CCH24] and, especially relevant to this topic, synthetic higher category theory. The advantage to studying mathematics synthetically becomes particularly highlighted when the objects themselves are difficult to work with or even to define via analytic methods – which is certainly the case for higher categories.

Higher Category Theory ∞-categories are categories where there is not just a set of morphisms between each pair of objects but a *space* of morphisms. Now the equations of the category must hold only up to higher coherences, which themselves only hold up to higher coherence, ad. infinitum. It is currently an important open problem in HoTT as to if it is possible to even define an ∞-category; this problem belongs to a class of similar problems known as coherence problems [Buc19, §3.3]. In classical foundations

there are many approaches to the study of ∞-categories, these include a variety of concrete models, such as Joyal's quasicategories [Joy08], the ∞-cosmological approach of Riehl and Verity [RV22] as well as the study of ∞-categories internal to ∞-topoi pioneered by Martini and Wolf [Mar22] [MW24].

Joyal's quasicategories are one particular choice model for ∞-categories, and their theory has been very well studied, for example in Lurie's work on higher toposes [Lur08]. A quasicategory is a simplicial set which satisfies a certain horn filling condition: For $0 < i < j$, any horn $\Lambda^i_j \to X$ must extend to a simplex $\Delta^j \to X$. To understand this requirement, take i to be 1 and j to be 2, then a horn in X is just a pair of composable morphisms and a 2-simplex is a morphism together with the data witnessing it as the composite. An alternative definition, introduced by Rezk [Rez00], starts with simplicial *spaces* instead of simplicial sets, and those simplicial spaces which form ∞-categories are known as complete Segal spaces. Compared to quasicategories, we can use a comparatively simple definition that takes advantage of the inbuilt *weakness* of spaces compared to sets. We may ask for the same horn filling condition, but instead of just asking for some extension, we ask for a contractible space of extensions. Equivalently, we may consider the $i = 1, j = 2$ case as above so that a Segal space is a simplicial space X such that the induced map $X_2 \to X_1 \times_{X_0} X_1$ is a weak equivalence. In this definition there are now two competing notions of invertible higher structure, in particular, for a simplicial space X, we have the space of objects X_0, as well as the space of isomorphisms $X_{\simeq}$ given by the simplicial structure. In order to obtain a well-behaved theory, we need some way of relating these structures, and hence we say a Segal space is complete if, in addition to the Segal condition above, the two notions of invertible structure are identified. This kind of completeness condition has also been studied inside univalent foundations for 1-categories where you may have a groupoid of objects equipped with a category structure. Such a category is called univalent, or complete, if the paths in the groupoid and the isomorphisms in the category coincide. One advantage of complete Segal spaces is that they can be obtained as a localisation of a higher presheaf topos and hence we can use Homotopy Type Theory as an internal language. This is achieved by postulating a generic bounded distributive lattice which we call the interval together with some additional axioms governing its behaviour so that it models the representable object Δ^1. We can then define a proposition on types that carves out those simplicial spaces which are (complete) Segal spaces and study their category theoretic properties.

1.2 Related Work

Apart from the stream of work on simplicial type theory, there have been many approaches to describing a type theory where types have an inbuilt, directed notion of morphism. Typically, this is achieved via a significant structural extension to type theory itself, introducing new directed forms of judgement, variance annotations or dual context systems modelling groupoidal contexts. Examples of these kinds of types theories are the directed type of theory of North [Nor18] and the later related work of Neumann and Altenkirch [NA25]. These works try to create theories in which *every* type is a category; this is in contrast to simplicial type theory where we have to prove

that the types we construct are categories. Whilst there is an obvious convenience benefit to the former approach, it is this goal that leads to the complication of the theory, for example the category of categories is not locally cartesian closed, and hence in a naive model with types as categories, we would not be able to form Π-types.

The construction of spaces in section 3.5, which is due to Gratzer, Weinberger, and Buchholtz [GWB26a], follows closely the earlier work of Licata, Orton, Pitts, and Spitters [LOPS18] where they use a tiny type to build internal universes. Weaver and Licata [WL20] also use a similar technique to construct internal universes in a bicubical setting; future work in this direction may give us a computational account of simplicial type theory.

Finally, we remark that synthetic approach to higher categories is being explored in areas outside of type theory. Cisinski, Cnossen, Nguyen, and Walde [CCNW26] in their work-in-progress book, describe a naive axiomatic approach to the theory of higher categories. Currently, simplicial type theory can account for almost all of the axioms assumed in that work; however, the details of a few axioms, for example showing that cocartesian fibrations are exponentiable, have yet to be published. In addition to the above work, Haugseng [Hau25] has a similar book project, again making use of an axiomatic approach to develop the theory of higher categories without having to appeal to any particular model.

1.3 Project Aims and Objectives

There are two key aims of this project. Firstly, to formalise parts of the existing literature on Simplicial HoTT (SHoTT) and demonstrate that this formalisation is reasonable and achievable in a setting without a strict interval type, native extension types or any custom computational rules (rewrite rules) on the interval lattice. Secondly, we explore modal type theory, implementing prototypes of a certain subset of MTT as well as extending the cohesive modal capabilities of the Agda proof assistant.

Formalising synthetic category theory Whilst there exists at least one formalisation of SHoTT, it takes place in the experimental proof assistant Rzk [KRW23]. This proof assistant extends type theory with an interval pretype – similarly to cubical type theory – and includes certain strictifications such as extension types. In contrast, this formalisation will take place in Agda, a mature and fully featured proof assistant, and will work with a foundation based on *book HoTT* extended only with axioms and a handful of modalities. This follows more closely the work by Gratzer, Weinberger and Buchholtz in that we will work pervasively only up to equivalence. Furthermore, in the Rzk proof assistant, there are no modal features, and this limits the scope of formalisation.

Extending the Modal Features of Agda The second aim involves extending Agda itself with more modal capabilities. Andreas Vezossi developed an extension to Agda

called Agda-flat that extends the type theory with an idempotent comonadic endo-modality, denoted ♭. This was implemented in order to formalise the construction of internal universes in type theory [LOPS18] and is based off the work on crisp type theory by Shulman [Shu18]. Initially, we explored adding the ♯ and *op* modalities as detailed in [GWB25], but ultimately we ended up with a simpler extension featuring just the ♯ modality. This, in combination with the existing ♭ modality, will give us the ability to talk about a large amount of category theory such as the universal property of the category of spaces and (full) subcategories.

1.4 Project Management

The main computing artefact is an open-source Agda library hosted on GitHub.[1] The project management features of GitHub are used to organise the development, these include: GitHub issues to track future features and existing bugs; organising changes into feature branches which become pull requests; and the use of the pull request review feature to make sure the changes are as consistent and as bug free as possible before being merged. Whilst the Agda code itself is checked for correctness as well as simple spell-checking and syntactic checks, there is also a significant amount of written commentary attached to the code which needs manual proof reading. We are also developing a code style and naming guide, which is important to maximise consistency as the code base grows and as we attracted external contributors. In addition to the pull requests, we also keep an updated changelog on the main website which includes details on new modules, features and future plans, as well as technical challenges or other interesting or unexpected discoveries in the formalisation process.

In order to increase the efficiency of the development process, continuous integration was also configured in GitHub, making use of Nix for reproducible builds. This meant there was an automatic check for code correctness with each commit to the repository, and then upon merging into the main branch, the website was built and uploaded via the GitHub pages system. As an open-source project, towards the end of the project timeline, as the project gained some exposure amongst the community, we received contributions in the form of pull requests from other researchers and academics. The regimented and well documented development process proved invaluable as the project scaled from a single developer to a small team of people working on this project. We would like to remark that all formalisation presented in this document was carried out by the author unless explicitly stated otherwise.

1.5 Documentation

Whilst sharing a large amount of process and philosophy with software engineering, a computer formalised library of mathematics differs in the fact that the primary purpose of the code is to be read and understood rather than executed. As such, documentation

[1] https://github.com/samtoth/agda-synthetic-categories

is an essential part of this project, on equal footing with the code itself. We borrow our methodology from Knuth, using a technique called literate programming.

Each Agda module is embedded within a markup language called Forester, an interactive notebook and mathematical documentation system developed by Jon Sterling [Ste23]. In the Forester system, the basic unit of composition is the *tree*. Each Agda module is a tree, and within a module, each definition, theorem, lemma or example has its own tree, with a informal mathematical statement together with the formalised Agda code. Inside the informal proofs and in the Agda code itself, other definitions and lemmas are referenced and links take you to the relevant tree where you can find more information. Forester also supports back-linking, a mechanism that allows one to see not only which other trees are referenced, but which other trees reference this one. Together these features allow a much higher degree of discoverability and ease of use within the library, both for general readers and maintainers of the codebase itself.

1.6 Summary of Contributions

This project contributes an Agda library in the 'book HoTT' style, featuring an interactive and discoverable web-frontend powered by the Forester tool as well as an Agda plugin (or 'backend') to interface the two technologies. This formalisation was facilitated by an extension to Agda which contributes the right-adjoint sharp modality of Shulman's cohesive type theory [Shu18]. Throughout this report, you will find cogs (⚙) which are hyperlinks to the relevant part of the formalisation. We summarise here a selection of key results that can be found in the library:

In homotopy type theory:

- A robust theory of reflective subuniverses and orthogonality conditions, including important closure properties of the above classes (⚙, ⚙)
- The localisation modal operator for an arbitrary class of maps as presented in [RSS20] (⚙)
- The theory of various kinds of colimits, and their descent properties (⚙)
- A formalisation of Rijke's [Rij17] modified join construction (⚙)
- A formalisation of the foundations of cohesive type theory, including the flat (⚙) and sharp (⚙) modality, as well as the shape modality (⚙), under certain axioms
- The theory of the algebra of distributive lattices, including the construction of the free algebra on a lattice (⚙)

In synthetic category theory:

- Notions of simplicial types (⚙), precategories (⚙) and groupoids as well as more refined notions such as triangulated types (⚙) and left Segal types (⚙)
- The definition of left/right and inner fibrations, covariant families and the closure properties that they all enjoy (⚙)
- The theory of universal elements and hence various categorical constructions such as limits, adjunctions and Kan extensions (⚙)

- Assuming a tiny interval, the construction of universes that classify certain 'nice' classes of fibrations (⚙)
- The construction of $\mathcal{S}$, the category of spaces, together with the proof that it is directed univalent (⚙)

Chapter 2

Modal Type Theory

Modalities have been well studied in the context of logic; however, there is some complication in combining modalities with dependant type theory. Shulman was one of the first to study modal type theory in combination with Homotopy Type Theory, where in his work on real-cohesive type theory [Shu18], the goal is to try and relate the synthetic circle generated by a higher inductive type to the analytic topological circle constructed as a subset of pairs of real numbers. The semantic tool that was used is called a cohesive topos, and notably it contains a string of adjoint endofunctors which are all furthermore idempotent monads or comonads: $\int \dashv \flat \dashv \sharp$. Shulman proves a no-go theorem stating that it is not possible to postulate the comonad $\flat$ internally and thus devises a bespoke modal type theory called cohesive type theory. The central obstruction that necessitates changing the type theory itself is that all internal definitions are valid in any context and so correspond externally to operations on all slice categories. Furthermore, any operation must be stable under substitution (corresponding externally to being stable under pullback). Modal type theory solves this problem with a modification to the context and variable rules so that only certain variables, which are now annotated with a modality, are usable. Later, Gratzer, Kavvos, Nuyts, and Birkedal [GKNB21] devise a type theory that is a vast generalisation of Shulman's cohesive type theory. In particular, they give a type theory which is parameterised by a 2-category called the mode theory. For a mode theory $\mathcal{M}$, the models of this type theory are $\mathcal{M}$-shaped diagrams into the 2-category of models of type theory and dependent right adjoints. The mode theory of crisp type theory is a particularly simple kind of 2-category called a locally-preordered 2-category with a single object and this results in a type theory with a significantly more simple syntax than in the general case. This project will largely focus on this subset of mode theories as their implementation is much better understood. Whilst Gratzer [Gra22] proved important meta-theoretic results such as strong normalisation, it still remains an open problem to give a usable and efficient implementation of the fully general multi-modal type theory.

2.1 MTT

We give a short presentation of the rules of modal type theory when specialised to a preordered mode theory. In the following, let $(\mathcal{M}, \cdot, \leq)$ be a preordered monoid.

Contexts The contexts in modal type theory are still generated by the empty context and context extension, but in addition to the type, context extension contains the data of the modality of a variable as well the modality under which it is locked. Note that unlike in the original presentation, we do not make locking a context an additional structural rule, but instead make it an admissible operation on the context. This more closely mirrors the experimental Agda implementation.

$$\frac{}{\vdash \cdot \ \mathrm{Ctx}} \qquad \frac{\vdash \Gamma \ \mathrm{Ctx} \qquad \Gamma/\mu \vdash A \ \mathrm{Type}}{\vdash \Gamma; x :^{\mu}_{\mathrm{id}} A \ \mathrm{Ctx}}$$

We define Γ/μ by recursion on contexts:

$$\cdot/\mu = \cdot$$
$$(\Gamma; x :^{\mu}_{\nu} A)/\sigma = (\Gamma/\sigma; x :^{\mu}_{\nu\cdot\sigma} A)$$

The variable rule The variable rule controls the access to modally locked variables:

$$\frac{\mu \leq \nu}{\Gamma; x :^{\mu}_{\nu} A; \Delta \vdash x : A}$$

Modal Π types We generalise the usual Π types of MLTT to include the possibility of assuming a modal variable. The computation and uniqueness rules are unchanged and hence omitted.

$$\frac{\Gamma/\mu \vdash A \ \mathrm{Type} \qquad \Gamma; x :^{\mu} A \vdash B \ \mathrm{Type}}{\Gamma \vdash (x :^{\mu} A) \to B} \qquad \frac{\Gamma; x :^{\mu} A \vdash f : B}{\Gamma \vdash (\lambda x.f) : (x :^{\mu} A) \to B[x]}$$

$$\frac{\Gamma \vdash f : (x :^{\mu} A) \to B \qquad \Gamma/\mu \vdash a : A}{\Gamma \vdash f(x) : B[x := a]}$$

Modal operators It is possible to give bespoke rules for the modal type operators at each modality as is done in [GKNB21]. In a type theory that contains inductive types, it is possible to define the modal operator for modality μ as the inductive type parameterised by a μ modal type $(\Gamma/\mu \vdash A)$ generated by the single constructor:

- $\mathrm{mod}_{\mu} : (x :^{\mu} A) \to \langle \mu | A \rangle$

2.2 Agda flat

Agda is a dependently typed programming language and proof assistant based off Martin-Löf's type theory. The original version was developed by Caterina and Thierry Coquand, the latter author who, alongside Huet and Paulin-Mohring, developed the proof assistant Rocq (formerly Coq). The current version of Agda (Agda 2) is laid out in detail in [Nor07], and it's key features include:

- Indexed inductive and coinductive types [1]
- Dependent (co)pattern matching
- Implicit arguments and type inference
- A robust metaprogramming and type-class like system
- An implementation of Cubical Type Theory which gives a fully computational account of the univalence axiom

Agda is considered a research language, and so has over the years been extended in many ways, allowing new ideas to be tested out in a language that has many convenience features (such as implicit arguments). This makes it a perfect candidate with which to experiment implementing new modal features – on top of the many co-existing modal systems that can be enabled in Agda via compiler switches. We give a brief overview of the existing implementation and detail how it has been altered to allow for more complex modal systems.

The existing Agda flat implementation In Agda, contexts are built up of variables, where each variable has data attached to it called *annotations*. Part of this annotation data is the modality data pertaining to cohesion – which is what Agda calls its flat modal feature, referring to Shulman's Cohesive Type Theory. Variables can be marked with either: flat ($\flat$), continuous (no marking) or squash $\top$. In the variable rule, Agda then checks that the modal annotation is less than or equal to the continuous modality. On this data there are a number of operations, most importantly composition, identity elements, ordering and *inverse composition*. The first of these operations, as expected, should satisfy the laws of a preordered monoid. The inverse composition, which we write $-/-$ should be part of a Galois connection:

$$a/x \leq b \iff a \leq x \cdot b$$

The inverse composition then acts as the locking operation on contexts described in §2.1, however in that setting we explicitly stored a pair of modalities in the context for each variable – the modality of the variable and modality of its locks. In this setting we only store the variable modality, and this is precisely what necessitates the squash modality, which otherwise appeared seemingly out of nowhere. The continuous modality, when inverse composed by the $\flat$ modality, becomes squash ($\top$) and the ordering dictates squash is the greatest modality and thus never usable[2]. This implementation is closer to the dual context version of cohesive type theory than to the modern and more flexible framework of MTT.

[1] As well as a zoo of other, often not well understood, induction schemes such as interleaved inductive-recursive or inductive-coinductive types

[2] This also has the effect that the Galois connection rule is actually only validated for non-squash modalities.

Agda permits modal annotations on all telescopes, which are used in Π-types, as well as in the specification of inductive types. This enables the modal Π-types and negative modal operators to be defined as in §2.1.

Extending Agda flat In the first prototype implementation, we add an extra modality to the variable annotation to keep track of the locks that a variable is under, in addition to its modality. Now inverse composition becomes simply *formal division*, a modality $\frac{\mu}{\nu}/\frac{\mu'}{\nu'}$ is just $\frac{\mu}{\nu} \cdot \frac{\nu'}{\mu'}$ where composition is as before but applied component wise. Now the variable rule says a variable $x :^{\mu}_{\nu} A$ is usable only if $\mu \leq \nu$. This way we can get rid of the squash modality as before, and the only thing needed to define the mode theory is the preordered monoid – no cleverness is required in defining a law abiding inverse composition operator.

This prototype implements the type theory presented in §2.1, and we instantiate it with the preordered mode theory presented by three generators $\{\flat, \sharp, op\}$, together with the following equations and relations:

$$\flat \leq \mathrm{id} \leq \sharp \qquad \flat \cdot \flat = \flat \qquad \sharp \cdot \sharp = \sharp \qquad op \cdot op = \mathrm{id}$$

Future outlook Currently, whilst the system is easily customisable to an arbitrary mode theory of suitable type, performing this customisation requires altering the Cohesion data type, composition operator and ordering relation, and then recompiling Agda. In an ideal implementation, this could be customised in user space, allowing a wider application of this system; however, since the word problem for monoids is undecidable, simply giving a presentation for the mode theory would not be enough, a decision procedure also needs to be implemented. Finding a way to allow the user to specify this would require careful consideration and design.

2.3 Cohesive type theory

Cohesive type theory, first introduced by Shulman [Shu18], is an alternative, but more narowly scoped framework for modal type theory. Sharing many features of the mode theory presented in §2.2, cohesive type theory gives us an idempotent comonadic modality $\flat$, together with it's right adjoint $\sharp$. To do this, it uses a dual context system whereby there is a context for *continuous* variables, and a context of *cohesive* variables. Locking the context by $\flat$ allows one to only use the cohesive variables, and locking by sharp moves all variables into the cohesive context. Prior to this work, Agda only implemented the $\flat$ modality of this system. Whilst I had developed a prototype extension for posetal MTT early in the project, due to a combination of some challenging bugs and the fact the extension was a significant alteration to the Agda modal system, it was not suitable to be merged into Agda at the time. Consequently, an extension was proposed to add only the $\sharp$ modality. As a comparatively simple and well understood change, this proposal was accepted and a PR[3] was merged and as of feburary 26th 2026, which will be a part of the upcoming Agda 2.9.0 release.

[3] https://github.com/agda/agda/pull/8409

An interesting aspect of this implementation[4] is that it uses a *postitive* modal operator for the sharp modality. In Agda, the sharp modal operator can be implemented as a record type with a single *sharp-modal* field. Making use of the modal Π-types in Agda, we may type the single destructor as follows:

$$-_\sharp : (\flat|\sharp A) \to A$$

In principle, this extension also allows modal record fields for any modality with *an internal left adjoint*, which is detected automatically in the code. Thus the work carried out in this extension may benefit future developers when new modalities are considered. As a record type, we may use co-pattern matching to define elements of $\sharp A$. To illustrate this, and to show how we may translate proofs about sharp into an informal setting we give the forward direction of the $\flat \dashv \sharp$ adjunction.

Proposition 2.3.1. *For all types A and B, we have a canonical map*

$$\sharp(\flat A \to B) \to (A \to \sharp B).$$

This map is an equivalence.

Proof. We just show that this map exists, the details of the equivalence may be found in the formalisation. Fixing $f : \sharp(\flat A \to B)$ and $a : A$, we need to give an element of $\sharp B$. By the definition of $\sharp$ as a coinductive type, we may given an element of B after making our context crisp. Now that f is a crisp element of $\sharp(\flat A \to B)$, we may use the destructor to obtain a genuine function $f_\sharp : \flat A \to B$. Now that a is also crisp, we may use the constructor of the $\flat$ modality so that $\mathrm{mod}_\flat(a) : \flat A$, to which we can apply $f_\sharp$ to obtain an element of B as required. More formally we may define this function, call it ϕ, by copattern matching as follows:

$$\phi(f, a)_\sharp := f_\sharp(\mathrm{mod}_\flat(a))$$

□

In either MTT or cohesive style modal type theory with only the $\flat$ modality, it is not possible to show that that $\flat$ types satisfy *modality extensionality* For any modality μ and μ-modal elements a and b, we may define a function $\mathrm{mod}_\mu(a) =_{\mu A} \mathrm{mod}_\mu(b) \to \mu(a =_A b)$, and analogously to function extensionality, we say μ satisfies modality extensionality if this map is an equivalence. In a note by Gratzer [Gra21], a simple semantic justification is given to show that it is independent of MTT that this map is an equivalence for all modalities μ. In our system with $\sharp$ and $\flat$ we show that both modalities enjoy *modal exensionality*. We begin by noting some basic facts about the sharp modality, we elide the proofs of the majority of these facts, the details of which may be found in the formalisation, but include the proof that $\sharp$ is lex, since we feel it is a simplification of the proof in [Shu18].

Proposition 2.3.2 (⚙). *$\sharp$ forms a Σ-closed reflective subuniverse*

Proposition 2.3.3 (⚙). *The family $(a, b : \sharp A) \mapsto \sharp(a_\sharp = b_\sharp)$ is an identity system for $\sharp A$, and thus sharp forms a* lex *modality in the sense of [RSS20].*

[4] as suggested to me by Amélia Liao

Proof. There is clearly a reflexive element of this family, given by $\mathsf{refl}(a : \sharp A)_\sharp := \mathsf{refl}$. This induces a map $a =_{\sharp A} b \to \sharp(a_\sharp = b_\sharp)$ by identity induction, and we can show it is an equivalence by using the fundamental theory of identity types. This aproach is the modernisation of a proof strategy known as encode-decode (see section 3.1.2 for a more detailed introduction). We must show for each $a : \sharp a$ that the type $\Sigma_b(\sharp(a_\sharp = b_\sharp))$ is a singleton. We know that $\sharp$ preserves Σ types, so this is just $\sharp(\Sigma_{b:A} a_\sharp = b)$, which is a singleton since, by the general theory of reflective subuniverses, $\sharp$ preserves singletons. □

We now hope to show an anologous property of the $\flat$ modality; to do so we will provide a modification of the usual identity induction rule for crisp identities.

Lemma 2.3.4 (Crisp identity induction, ⚙). *Let $P_a :^\flat (b :^\flat A) \to (\flat|a = b) \to \mathsf{Type}$ be a family of types. We can extend a crisp partial section of P at (a, refl) to a full section for any crisp elements b and crisp identification $a = b$. This section, once restricted appropriately satisfies a definitional computation rule. That is, we can construct the following function:*

$$J_a^\flat : (P :^\flat (b :^\flat A) \to (\flat|a = b) \to \mathsf{Type}) \\ \to P(a, \mathsf{refl}) \to \Pi_{(b,p:^\flat a=b)} P(b, p)$$

such that $J^\flat(P, p, (a, \mathsf{refl})) \equiv p$.

Proof. Define $P' : (b : A) \to a = b \to Type$ to be the family $(b, p) \mapsto \sharp(P(b, p))$. Notice that this type checks without issue since after applying the $\sharp$ modal operator, b and p become crisp. Now using identity induction, we may define a function $\Pi_b(a = b) \to P'(b, p)$, which we can then transpose across the $\flat \dashv \sharp$ adjunction, to obtain our $J^\flat$ as desired. □

Corollary 2.3.5 (⚙). *The canonical map $\mathrm{mod}_\flat(a) = \mathrm{mod}_\flat(b) \to \flat(a =_A b)$ defined by identity induction is an equivalence.*

Proof. We may define the inverse map using crisp identity induction, so that it suffices to give an element $\flat(a =_A a)$. To show this map is both a left and right inverse, we may use identity induction and crisp identity induction respectively, in combination with the definitional computation properties of both J and $J^\flat$. □

Chapter 3

Simplicial Homotopy Type Theory

We will now turn our attention to our motivating use case for modal type theory: simplicial (or triangulated) type theory. Initially explored by Riehl and Shulman [RS17], simplicial type theory is a foundational system built on top of HoTT where it is possible to work synthetically with ∞-categories. We expect our reader to be familiar with the basics of Homotopy Type Theory as laid out in [Uni13]. In this section we briefly recall some important notions of homotopy type theory, in particular highlighting techniques that facilitate computer formalisation. We also provide a brief overview of the foundations of STT after which we can develop some basic category theory and then finally show how the modal features developed in Chapter 2 enable us to construct the category of Spaces, the first concrete example of a non-trivial category in STT.

3.1 Formalising Homotopy Type Theory

3.1.1 Equivalances

We recall a few primitive notions from the HoTT book [Uni13].

Definition 3.1.1 (Quasi-isomorphism, ⚙). Given a map $f : A \to B$, a quasi-isomorphism witness of f consists of the data:

- A map $g : B \to A$
- A homotopy $\eta : fg \sim \mathrm{id}$
- A homotopy $\varepsilon : gf \sim \mathrm{id}$

◊

Whilst this is an appropriate definition of isomorphism in the context of sets, it turns out that this definitions lacks the coherence to appropriately capture the *higher categorical* notion of equivalence. There are many possible definitions of coherent isomorphism that one could choose. Our formalisation takes the following definition as primitive, because it has a canonical choice of *inverse* map, which is definitionally

preserved when upgrading a non-coherent isomoprhism into a coherent one.

Definition 3.1.2 (⚙). A map f is an equivalence if there is a quasi-isomorphism (g, η, ε), such that $f\varepsilon \sim \eta g$. ◊

Note the difference in terminology, there may be many quasi-isomorphism witnesses of f, but the data that a map is an equivalence is unique if it exists – that is, it is a (homotopy) proposition.

Proposition 3.1.3. *Let f be a map $A \to B$, then the following are equivalent propositions:*

1. *f is an equivalence*
2. *The type of quasi-isomorphism witnesses of f is merely inhabited (⚙)*
3. *For all $b : B$, the fibre of f at b is contractible (⚙)*
4. *f is a biinvertible map (⚙[1])*
5. *f is pathsplit – there is a section of f and a section of $f^* : a = b \to f(a) = f(B)$ (⚙)*

Notice that condition (2) states that any quasi-isomorphism can be upgraded to an equivalence. In practice this is used to forego the burden of providing the extra coherence, although we note that the process of upgrading a quasi-isomorphism cannot preserve all of the input data, and so when it is convenient to do so in the formalisation, we construct a fully coherent equivalence by hand.

In a pedagogical note by de Jong [dJon25], there is an exposition of the so-called 3-for-2 property which is presented as a general technique for proving maps are equivalences.

Proposition 3.1.4 (3-for-2, ⚙). *Let $f : A \to B$, $g : B \to C$ and $h : A \to C$ be a commuting triangle of maps, then if any two are equivalences, then so is the third.*

This theorem is used extensively throughout the formalisation, most notably, in the situation where you want to show f is an equivalence and C is a type for which paths are easier to describe than in B. We now describe a pair of similar properties which also find extensive use in the library.

Definition 3.1.5 (⚙). Given maps $f : A \to B$ and $g : X \to Y$, the type of squares from f to g is written $\mathsf{Arr}(f, g)$, and consists of maps $t : A \to X$, $b : B \to Y$ and a homotopy $gt \sim bf$ that *fills* the square. ◊

Corollary 3.1.6 (⚙). *Given a pair of maps f and g, we say that a map of arrows $F = (t, b, H) : \mathsf{Arr}(f, g)$ is an equivalence if both t and b are equivalences. It follows from the 3-for-2 property of equivalences that if $F : \mathsf{Arr}(f, g)$ is an equivalence, then f is an equivalence if and only if g is.*

Definition 3.1.7 (⚙). Given a map g, we say that a map f is a retraction of g if there is a pair of arrow maps $s : \mathsf{Arr}(f, g)$ and $r : \mathsf{Arr}(g, f)$ such that the composite $rs : \mathsf{Arr}(f, f)$ is homotopic to the identity (we may use 3.1.11). ◊

Note that to check this identity we need both that the top and bottom maps compose to the identity, but also that the squares paste together to form the reflexive homotopy. We would like to show that a retract of an equivalence is an equivalence, but it turns out this extra coherence is not necessary for this result to hold.

[1] Formalised by Fredrik Bakke

Definition 3.1.8 (⚙). Given g and f as above, then we call the data of a pair of arrow maps $s : \mathsf{Arr}(f, g)$ and $r : \mathsf{Arr}(g, f)$ such that the top and bottom maps compose to the identity an incoherent retraction of f onto g. We say g is an incoherent retract of f. ◊

Proposition 3.1.9 (⚙). *If f is an incoherent retract of an equivalence e, then f is an equivalence.*

3.1.2 Identity systems and encode-decode

When a new type is constructed it is natural to ask the question of when two elements of this type are equal. Upgrading ourselves to homotopical thinking, we may better ask the question of what does the *type* of equalities look like. For some types this question has a simple answer, for example we have the following equivalence:

$$(a, x) =_{A\times X} (b, y) \simeq (a = b) \times (x = y)$$

which tells us *an equality of pairs is a pair of equalities*. For other types, the question is not so easily answered [Wär25]. The so called encode-decode method, introduced in the HoTT book [Uni13], gives us a general technique for characterising path types.

Proposition 3.1.10 (The fundamental theorem of identity types, [Uni13, Thm 5.8.2 (iv)]). *Given a family $P : A \to \mathsf{Type}$ and an element $a : A$ any family of maps $\alpha_b : a = b \to P(b)$ is a fibrewise equivalence iff the family $\Sigma_a P(a)$ is contractible. Such a family P with this property is sometimes called* torsorial.

To demonstrate its use, we will work through an example of a simple but nontrivial path type, namely the type of arrow maps.

Proposition 3.1.11 (⚙). *Given a pair of arrow maps $F = (t, b, H), G = (t', b', H') : \mathsf{Arr}(f, g)$, then we have an equivalence, $(F = G) \simeq \mathsf{Arr}_=(F, G)$, where $\mathsf{Arr}_=$ is defined as follows:*

$$\mathsf{Arr}_=(F, G) := \Sigma_{p:t\sim t'}\Sigma_{q:b\sim b'}(H \cdot gp \sim qf \cdot H').$$

Proof. First note that we have a fibrewise map $F = G \to \mathsf{Arr}_=(F, G)$ since after using identity induction, it suffices to give a reflexive path $\mathsf{Arr}_=(F, F)$. Now we will use the fundamental theorem of identity types to show that this map is an equivalence. That is, fixing $F : \mathsf{Arr}(f, g)$, we need to show that the type $\Sigma_G \mathsf{Arr}_=(F, G)$ is contractible. Unfolding this data, we have:

$$\begin{aligned}
&\Sigma_{(t',b',H')}\Sigma_{p:t\sim t'}\Sigma_{q:b\sim b'}(H \cdot gp \sim qf \cdot H') \\
\simeq\ &\Sigma_{(b',H')}\Sigma_{q:b\sim b'}(H \sim qf \cdot H') \\
\simeq\ &\Sigma_{H'}(H \sim H')
\end{aligned}$$

where each successive line contracts away a pair of singleton, e.g. on the second line replacing (t', p) with $(t, \mathsf{refl}_\sim)$. □

Notice the compositional nature of this proof, each pair of singletons we contracted away were singletons because of the identity system on maps, due to function extensionaliy. To make the line of reasoning above fully formal and convenient in Agda, we provide the following lemma:

Lemma 3.1.12. *If* $P : A \to \mathsf{Type}$ *is torsorial and* $Q : \Pi_{(a:A)} P(a) \to B(a) \to \mathsf{Type}$ *is torsorial over* $B(a)$ *for any choice of element* $(a, b) : \Sigma_a P(a)$*, then the family* $(a, b : \Sigma_a B(a)) \mapsto \Sigma_{p:P(a)} Q(a, p, b)$ *is torsorial.*

This technique, alongside the use of Agda's instance search mechanism (analogous to type class resolution in Haskell), makes writing identity systems highly convenient.

3.1.3 Cartesian squares, cubes and orthogonality

Definition 3.1.13 (⚙). Given a map of arrows $F : \mathsf{Arr}(f : A \to B, g : X \to Y)$, the type of lifts of F consists of a diagonal map $l : B \to Y$, homotopies L and K witnessing that the upper left and lower right triangles commute, **and** a homotopy witnessing that these two triangle fillers when pasted together are homotopic to the original filler of the square. ◊

We may give a more concise definition as follows:

Proposition 3.1.14 (⚙). *Given a pair of maps* $f : A \to B$ *and* $g : X \to Y$*, we can form a square given witnessing the naturality of the functorial action of the exponential:*

$$\begin{array}{ccc} X^B & \xrightarrow{h_*} & Y^B \\ {\scriptstyle f^*}\downarrow & & \downarrow{\scriptstyle f^*} \\ X^A & \xrightarrow[h_*]{} & Y^A \end{array}$$

Taking the pullback of the lower span gives us exactly the data of a map of arrows from f *to* g*, and so there is a canonical* gap map *from* $X^B \to \mathsf{Arr}(f, g)$*, constructing the square for which the map* X^B *is trivially a filler. This gap map is written* $f \pitchfork g$*. The type of fibres of* $f \pitchfork g$ *at some* $F : \mathsf{Arr}(f, g)$ *is equivalent to the type of lifts of* F*.*

Definition 3.1.15 (⚙,⚙). Given a pair of maps f and g we say that f is left orthogonal to g (written $f \perp g$) if for any map $\mathsf{Arr}(f, g)$, the space if lifts is contractible. By proposition 3.1.14 we may equivalently say that the square, which by abuse of notation we also call $f \pitchfork g$ is cartesian. ◊

As we will see later, many concepts in mathematics, and particularly in synthetic higher category theory, can be expressed as orthogonality conditions. The following theorem gives us many strategies to prove that certain maps are orthogonal:

Theorem 3.1.16 (⚙). *Suppose* h *is some map, and each map* f_i *is assumed to be left-orthogonal to* h*, then:*

1. *For any equivalence* e*,* $e \perp h$
2. $f_1 \times f_2 \perp h$
3. $gf \perp h$ *iff* $g \perp h$
4. $\Sigma_i f_i \perp h$
5. *If* g *is a pushout of* f*, then* $g \perp h$
6. *If* g *is a (coherent) retract of* f*, then* $g \perp h$
7. *The pushout product of* f *with* any map g, $f \hat{\times} g \perp h$

Proof. We provide proofs of (5) and (6) which demonstrate the general strategy.

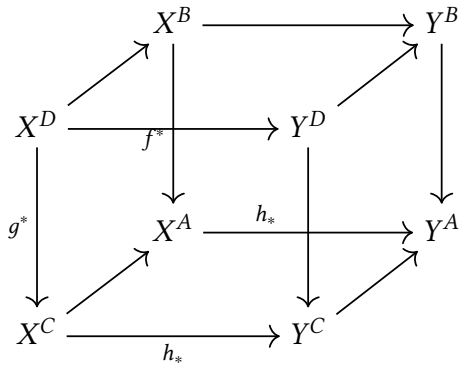
Suppose then that $f : A \to B$ and $h : X \to Y$ are maps such that $f \perp h$, or equivalently that the following square is a pullback:

$$\begin{array}{ccc} X^B & \xrightarrow{h_*} & Y^B \\ {\scriptstyle f^*}\downarrow & & \downarrow{\scriptstyle f^*} \\ X^A & \xrightarrow[h_*]{} & Y^A \end{array}$$

Now for (5.) let $F : \mathsf{Arr}(f, g)$ be a cocartesian square (pushout square). Note that the universal property of cocartesian squares says that for any type Q, the induced square:

$$\begin{array}{ccc} Q^D & \longrightarrow & Q^B \\ {\scriptstyle g^*}\downarrow & & \downarrow{\scriptstyle f^*} \\ Q^C & \longrightarrow & Q^A \end{array}$$

is cartesian. So then we may construct a commuting cube of maps as follows:



which we say is a map in the category of squares $\mathsf{Sq}(g \pitchfork h, f \pitchfork h)$. Since both the left and right squares of cube are cartesian, we may say that this is a cartesian map of squares. By the pullback-pasting property of cartesian squares, if $f \pitchfork h$ is cartesian then so is $g \pitchfork h$, and thus $g \perp h$.

For (6.), we start with a map of arrows $\mathsf{Arr}(g, f)$ together with a retraction $\mathsf{Arr}(f, g)$ so again we may construct a cube from $g \pitchfork h$ to $f \pitchfork h$. Every cube induces a map of arrows from the gap map of its domain to the gap map of its codomain square, and we hope to show that this square determines an incoherent retract so that when the gap map of $f \pitchfork h$ is an equivalence, then so is the gap map of $g \pitchfork h$ and thus $g \perp h$. This square is given by the mapping $X^D \to X^B$ in the top left corner, and for the bottom map, we use the functorial action of pullbacks with respect to spans. We may show that the top left map has a retraction, and that the map of spans also has coherent retraction which together show $g \pitchfork h$ is an incoherent retract of $f \pitchfork h$ as desired. □

3.2 Algebra, simplical shapes and duality

One point of view on topos theory is that each topos classifies a geometric theory. Given a theory $\mathbb{T}$ there is a topos $S[\mathbb{T}]$ with a universal model of $\mathbb{T}$ called U_T with the

property that any for any other topos $\mathcal{E}$ there is an equivalence between $\mathbb{T}$-models in $\mathcal{E}$ and geometric morphisms $\mathcal{E} \to S[\mathbb{T}]$. The induced $\mathbb{T}$-model is the inverse image of the universal model in $S[\mathbb{T}]$. One of our primary intended models, is simplicial spaces, which classify the theory of totally-ordered bounded distributive lattices.

Axiom 1 (The universal lattice)**.** *We assume a non-trivial bounded distributive lattice I. The underlying set of I is written Δ^1.*

From this we can define the n-simplices, starting with Δ^2, the underlying set of the free I algebra on a single generator. (⚙) We may more generaly define Δ^n to be the order preserving maps $[n] \to \Delta^1$. (⚙)

Definition 3.2.1. Let A be an I algebra. We define the spectrum of A to be the set of algebra homomorphisms $\mathrm{Hom}_I(A, I)$. This extends to a functor spec : I-alg $\to \mathrm{Set}^{op}$. ◊

We say that a type is affine if it arises at the spectrum of an I algebra. For example, since I is initial, 1 is spec(I), similarly all cubes I^n arise as the spectrum of free I-algebras $I[n]$.

Definition 3.2.2. Let X be a set, the pointwise I-algebra $\Pi_X I$ is called the algebra of observations: $\mathcal{O}(X)$. It determines a functor $\mathcal{O}(-) : \mathrm{Set}^{op} \to$ I-alg ◊

These two functors determine an adjunction $\mathcal{O}(-) \dashv \mathrm{spec}(-)$. We give explicit descriptions of the unit and counit maps. ϵ_X is a map $X \to \mathrm{spec}(\mathcal{O}(X))$, which unfolds to $X \to \mathrm{Hom}_I(\Pi_X I, I)$, and this is just given by evaluation. The unit η_A at an I-algebra A is a homomorphism $\mathrm{Hom}_I(A, \mathcal{O}(\mathrm{spec}(A))) = \mathrm{Hom}_I(A, \Pi_{Hom_I(A,I)} I) \simeq \Pi_{Hom_I(A,I)} Hom_I(A, I)$ defined in the obvious way.

Definition 3.2.3 (quasi-coherence)**.** We say that an I-algebra A is quasi-coherent if the unit of the adjunction at A is an equivalence of algebras. ◊

Example 3.2.4. I, as an I algebra, is quasi-coherent. ◊

We refer the reader to Sterling and Ye [SY25] for a comprehensive study of various forms of quasi-coherence, but for now we will assume the following:

Axiom 2 (Synthetic quasi-coherence)**.** *Every finitely presented I algebra is quasi-coherent.*

Theorem 3.2.5 (Phoa's principal)**.** *A simple consequence of axiom 2 is known as* Phoa's principal, which states that every map $I \to I$ is a polynomial, i.e. there is an equivalence $I^I \simeq I[x]$.

Proof. We instantiate the synthetic quasicoherence axiom at $I[x]$, giving an isomorphism $I[x] \cong \mathcal{O}(\mathrm{spec}(A)) = \Pi_{Hom_I(I[X],I)} I$ and by it's defining property, $Hom_I(I[x], I)$ is equivalent to just I. Because this is an equivalence of I-algberas and not just of sets, we get that any $f : I^I$ is equal to $i \mapsto f(1) \wedge i \vee f(0)$. □

3.3 Simplicial Foundations

We begin by generalising common limit and colimit constructions in Homotopy Type Theory (HoTT) to the directed case. We then carve out the subuniverses of simplicial types, precategories and groupoids, and give an overview of their basic properties. We first briefly the basic setup of synthetic morphisms in simplicial type theory.

Definition 3.3.1. For any type A, and pair of objects $a\ b : A$, we may define the type of synthetic momrphisms, written $\mathrm{Hom}_A(a, b)$ as the type of maps $f : \Delta^1 \to A$, together with paths $f(0) = a$ and $f(1) = b$ ◊

Already, this gives us an extremely desirable property for working with synthetic categories.

Proposition 3.3.2. *Every map $F : A \to B$ is functorial*

Proof. Suppose we have a synthetic morphism $f : \Delta^1 \to A$, with identities $p : f(0) = a$ and $q : f(1) = b$, then $Ff : \Delta^1 \to A$ together with $Fp : Ff(0) = F(a)$ and $Fq : Ff(1) = F(b)$ is the induced synthetic morphism $\mathrm{Hom}_B(F(a), F(b))$. □

3.3.1 Comma and Cocomma Types

Given a cospan diagram $S = A \xrightarrow{f} B \xleftarrow{g} C$, the type of cones over S at a type Q consists of maps $p : Q \to A$, $q : Q \to B$ together with a homotopy $fp \sim gq$ witnessing that the square commutes. A lax cone consists of the same data, but instead of an undirected homotopy, we have a morphism in the arrow category $Hom_{B^Q}(fp, gq)$. There is an obvious duality leading us to the definition of a lax cocone under span, as well as the corresponding oplax (co)cones.

Definition 3.3.3 (Comma objects, ⚙)**.** Let S be a cospan, and (P, p, q, H) be a lax cone over S. For any type Q, there is a canonical map $(Q \to P) \to \mathrm{LaxCone_S(Q)}$ given by postcomposing each leg of the cone at P. We say (P, p, q, H) is a comma cone when this map is an equivalence, and that P is a comma object. ◊

Just like in the undirected case, we can construct comma objects directly via Σ types. Given a cospan S as above, the lax cone $(\Sigma_{a:A}\Sigma_{c:C}\,\mathrm{Hom}_B(f(a), g(b)), \pi_1, \pi_2, \pi_3)$ is a comma cone. We use the notation $f \downarrow g$ for the apex of this cocone, and in the case A and C are categories, it is referred to as the *comma category* of f and g.

Common constructions with comma objects

- The slice category $C_{/c}$ is the comma object $\mathrm{id} \downarrow c$, where c is the map $1 \to C$. Dually, the coslice category $C_{c/}$ is $c \downarrow \mathrm{id}$.
- More generally, for any map $p : \mathcal{K} \to C$ we write $C_{/p}$ for the category $\Delta \downarrow_{C^{\mathcal{K}}} p$.
- The comma category $\mathrm{id}_C \downarrow \mathrm{id}$ is simply the space of morphisms in C

Cocomma objects are defined in the obvious way as dual to comma objects. In order to give an explicit construction of cocomma objects, we introduce the idea of directed higher inductive types. The cocomma object of a span $f : A \leftarrow B \to C : g$ is given by an inductive type $f \uparrow g$ generated by the following data:

- $\iota_1 : A \to f \uparrow g$
- $\iota_2 : C \to f \uparrow g$
- $\mathrm{glue} : \Pi_{b:B}\,\mathrm{Hom}_{f\uparrow g}(\iota_1(f(b)), \iota_2(g(b)))$

Remark 3.3.4. We can reduce the problem of directed higher inductive types to regular higher inductive types where each arrow constructor:

$$c_n : (x : \Gamma) \to \ldots \to \mathrm{Hom}_X(c_k(\ldots), c_l(\ldots))$$

is replaced by a constructor taking an interval argument and a pair of path constructors:

- $c_n : (x : \Gamma) \to \ldots \to (i : \Delta^1) \to X$
- $c_{n_0} : (x : \Gamma) \to \ldots \to c_n(0) = c_k(\ldots)$
- $c_{n_1} : (x : \Gamma) \to \ldots \to c_n(1) = c_l(\ldots)$

◊

Theorem 3.3.5 (⚙). *We can encode the cocomma object of $f : A \leftarrow B \to C : g$ via the undirected pushout as follows:*

$$\begin{array}{ccc} A \times \delta\Delta^1 & \xrightarrow{\overline{[f,g]}} & B + C \\ \downarrow & & \downarrow \\ A \times \Delta^1 & \longrightarrow & \bullet \end{array}$$

Proof. We show the cocones under the above span are equivalent to the lax cocones under the original span:

$$\begin{aligned} \mathrm{Cocone}(Q) &\simeq \Sigma_{p:B+C\to Q}\Sigma_{q:A\times\Delta^1\to Q}(p\iota_1 f \sim q(\mathrm{id},0) \times p\iota_2 g \sim q(\mathrm{id},1)) \\ &\simeq \Sigma_{p:B\to Q}\Sigma_{q:C\to Q}\Sigma_{r:A\to Q^{\Delta^1}}(\Pi_{a:A}(r(a)(0) = p(f(a)) \times r(a)(1) = q(g(a)))) \\ &\simeq \Sigma_{p:B\to Q}\Sigma_{q:C\to Q}(\Pi_{a:A}\,\mathrm{Hom}_Q(p(f(a)), q(g(a)))) \\ &\simeq \mathrm{LaxCocone}_{(f,g)}(Q) \end{aligned}$$

□

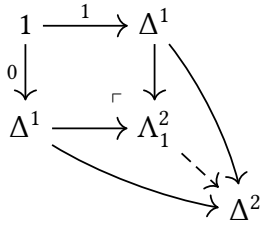
Common constructions with cocomma objects

- Given a pair of types A and B, we write $A \star B$ for the lax pushout $\pi_1 \uparrow^{A\times B} \pi_2$. This is called the blunt or directed join of A and B and is the disjoint sum of the two types with arrows added from each $a : A$ to each $b : B$.
- The above construction, when specialised to the case $B = 1$ is written $A^{\triangleright}$, and called the type of right cones of A. When A is 1 we write $B^{\triangleleft}$, called the type of left cones of B.
- For pointed types (A, a_0) and (B, b_0), we can define the directed wedge as $a_0 \uparrow^1 b_0$, which is the sum of A and B with an arrow added between $\iota_1(a_0)$ and $\iota_2(b_0)$.
- In particular, $1 \uparrow^1 1$ is just Δ^1

3.3.2 (Pre)categories

In this section we will describe those types that behave like precategories. As discussed in the introduction, this work is derived from the classical work on complete Segal spaces as a model for ∞-categories.

Definition 3.3.6 (The inner horn, ⚙)**.** We define the type Λ^2_1 by glueing a pair of intervals end to end. This type comes with a canonical map into the 2-simplex.



◊

Proposition 3.3.7 (⚙). *The map in 3.3.6 is an embedding and determines* Λ^2_1 *to be the subset of* $(i, j) : \Delta^2$ *where* $i = 0 \vee j = 1$.

Definition 3.3.8 (⚙)**.** We call a type Segal if it is local with respect to the horn inclusion $\Lambda^2_1 \hookrightarrow \Delta^2$. ◊

Decoding the above definition, we get that a type is a precategory precisely if for any pair maps $f : Y \to Z$ and $g : X \to Y$, the type $\Sigma_{h:X\to Z} fg \sim h$ is contractible. Since in homotopy type theory contractible means unique, we can extract the composition map from this data.

In the interest of generality, and in anticipation of the construction of the category of spaces, we do not globally assume that our interval is totally ordered. Nonetheless, we may consider those types which *see the interval as totally ordered.* We call these types simplicial types since, if every type was simplicial, then the interval would be totally ordered, as would be the case if we were working in the category of simplicial spaces.

Definition 3.3.9 (Simplicial types, ⚙)**.** A type A is said to be simplicial if it is null with respect to the family of propositions $(i \leq j) * (j \leq i)$ for each $i, j : I$. ◊

By the machinery developed in [RSS20], we have that there is a reflective subuniverse $\mathcal{U}_{⊡}$ of simplicial types, which is closed under Π and Σ as well as a localisation operation $⊡ : \mathcal{U} \to \mathcal{U}_{⊡}$ equiped with a universal map $\eta : A \to ⊡A$. We may now isolate the precategories as those types which are both simplicial and Segal. In the model of this type theory in simplicial spaces, these types correspond to not necessarily complete Segal spaces. In order to obtain the complete Segal spaces, we must consider one further property, namely the Rezk types; however we leave the formalisation of these for future work, and refer the reader to Riehl and Shulman's [RS17] original work on simplicial type theory for an overview.

Definition 3.3.10 (Groupoids, ⚙)**.** A type is a groupoid if, in addition to being simplicial, it is null with respect to the interval. Again, this gives rise to a Σ-closed reflective subuniverse of groupoids. ◊

It is worth noting that these are automatically *univalent* groupoids. To define pregroupoids you start with a precategory and ask each morphism to be an isomorphism, a univalent category with this property is equivalently a groupoid in the above sense.

3.3.3 Fibrations

Now we have isolated the types that behave like categories, we turn our attention to fibrations. Straightening/unstraightening, a consequence of the univalence axiom,

states that for a type B, the type of bundles over B (types equipped with a $\mathcal{U}$-small map into B) and families $B \to \mathcal{U}$ are equivalent. Given a bundle $E \to B$, we may ask for each fibre E_b to be a category, or groupoid. This corresponds exactly to asking for a family of types $P : B \to \mathcal{U}$ where each $P(b)$ is a category or groupoid. By the slogan *everything stated in simplicial type theory is automatically functorial*, we might expect that that for each moprhism $a \to b$ in B, we can obtain a map $P(a) \to P(b)$ by functoriality. Unfortunately, this is not the case. This is not due to a failure of our slogan, but instead by the fact that morphisms in $\mathcal{U}$ are not functions as we might expect. In order to correct this situation, we introduce left (right) fibrations and their straightened analogue, covariant (contravariant) families.

Definition 3.3.11 (Covariant families, ✿)**.** We say that a family of types P over some type A is covariant if for any arrow $f : \Delta^1 \to A$, and element $x_0 : P_{f(0)}$ in the fibre over $f(0)$, the type $\Sigma_{(x_1 : P_{f(1)})} Hom_P^f(x_0, x_1)$ is contractible. ◊

Definition 3.3.12 (Left fibrations, ✿)**.** A map is a *left fibration if it is right orthogonal to the map* $1 \xrightarrow{0} \Delta^1$*. Dually, we call a map a right fibration if it is right orthogonal to the map that picks out the right endpoint of* Δ^1*.* ◊

Theorem 3.3.13 (✿)**.** *A family is covariant iff it's unstraightening is a left fibration.*

Proof. Fix a family P over A. First we note that the data of a map of arrows from $\{0\} \to \Delta^1$ to $\tilde{P} \to A$ is exactly an arrow f in A together with an element $x_0 : P(f(0))$. Fixing this data, it remains to show that the type of lifts is equivalent to the type $\Sigma_{x_1 : P(f(1))} Hom_P^f(x_0, x_1)$. We calculate:

$$\begin{aligned}
\Sigma_{x_1 : P(f(1))} \mathrm{Hom}_P^f(x_0, x_1) &\simeq \Sigma_{x_1 : P(f(1))} \Sigma_{\bar{f} : \Pi_{i:\Delta^1} P(f(i))} (\bar{f}(0) = x_0 \times \bar{f}(1) = x_1) \\
&\simeq \Sigma_{\bar{f} : \Pi_{i:\Delta^1} P(f(i))} \bar{f}(0) = x_0 \\
&\simeq \Sigma_{\bar{f} : \Delta^1 \to \Sigma_A B} \Sigma_{H : \pi_1 \bar{f} = f} (H(0)^*(\pi_2 f'(0)) = x_0) \\
&\simeq \Sigma_{\bar{f} : \Delta^1 \to \Sigma_A B} \Sigma_{H : \pi_1 \bar{f} = f} \Sigma_{K_1 : \pi_1(f'(0)) = f(0)} \Sigma_{K_2 : K_1^*(\pi_2 \bar{f}(0)) = x_0} H(0) = K_1 \\
&\simeq \Sigma_{\bar{f} : \Delta^1 \to \Sigma_A B} \Sigma_{H : \pi_1 \bar{f} = f} \Sigma_{K : \bar{f}(0) = (f(0), x_0)} \pi_1(K) = H(0) \\
&\simeq \mathrm{Lift}(x_0, f, \mathrm{refl})
\end{aligned}$$

□

Proposition 3.3.14 (✿)**.** *A left fibration over* 1 *is a groupoid*

Corollary 3.3.15. *The fibres of a covariant family are all groupoids.*

The notion of being a Segal type is not stable under forming Σ-types; however, by the closure properties of right orthogonal classes of maps, we may isolate the families $B : A \to \mathcal{U}$ over Segal types A, for which the type $\Sigma_a B(a)$ is itself Segal. These are exactly the inner-fibrations, maps which are right orthogonal to the inner horn inclusion.

Theorem 3.3.16 (✿)**.** *A left (right) fibration is an inner fibration.*

Proof. Notice that the pushout product of the left enpoint inclusion with the right endpoint inclusion of the 1-simplex is computed to be the inclusion of the inner horn into the square. The inner horn inclusion into the 2-simplex is manifestly a retract of this map and hence, by the closure properties of right orthogonal maps, the result is obtained. □

Corollary 3.3.17 (⚙). *The sum of a covariant family over a precategory is itself a precategory.*

For a standard 1-category C there is a *canonical* (co)-presheaf associated to each object c, namely (co)representable family $\mathrm{Hom}(c, -)$. As a left fibration, this corresponds to the map $C_{c/} \to C$. We would like to show that, at least for our precategories (simplicial and Segal types), that this map is a left fibration. In Riehl and Shulman [RS17], this result is given, making large use of the strict extension types and shape layer. In our setting, a direct translation of this proof would be quite unwieldy, and it turns out that without the global assumption of simpliciality we can give a more refined hypothesis to this proposition. First we must expose some definitions.

Definition 3.3.18 (⚙). We call a type left Segal if it is local with regards to the inclusion $\Lambda^2_1 \hookrightarrow 1 \rtimes \Delta^1$. Dually we say a type is right Segal if it is local to $\Lambda^2_1 \hookrightarrow \Delta^1 \rtimes 1$. If a type is both left and right Segal, we say it is **strong** Segal. ◊

Lemma 3.3.19 (⚙). *Given a type B, we may view the directed join $- \rtimes B$ as a functor* $\mathsf{Type} \to \mathsf{Type}_{/B}$. *Similarly, the slice construction can be viewed as a functor* $\mathsf{Type}_{/B} \to \mathsf{Type}$ *which takes a map $p : C \to B$ to the type $C_{/p}$. There is an adjunction $- \rtimes B \dashv \mathit{Slice}_B$.*

Proof. This follows from a short calculation:

$$\begin{aligned} A \to C_{/p} &\simeq \Sigma_{f:C^A} \Pi_{a,b} \,\mathrm{Hom}(f(a), p(b)) \\ &\simeq \Sigma_{f:C^A} \Sigma_{g:C^B} (\Pi_{a,b} \,\mathrm{Hom}(f(a), g(b))) \times (g \sim p) \\ &\simeq \Sigma_{f:C^{A \rtimes B}} f\iota_2 \sim p \\ &\simeq \mathsf{Type}_{/B}(A \rtimes B, p) \end{aligned}$$

□

Theorem 3.3.20 (⚙). *Given a type A, A is left Segal iff for all $x : A$, $A_{x/} \to A$ is a left fibration.*

Proof. Consider the following square:

$$\begin{array}{ccc} 1 \rtimes \Delta^1 \to_\bullet A & \xrightarrow{\sim} & {A_{x/}}^{\Delta^1} \\ \downarrow & & \downarrow \\ \Lambda^2_1 \to_\bullet A & \xrightarrow[\sim]{} & A^{\Delta^1} \times_A A_{x/} \end{array}$$

By definition, the map on the right is an equivalence iff $A_{x/} \to A$ is a left fibration, and the left map is an equivalence for all points $x : 1 \to A$ iff the unpointed map

$A^{1 \star \Delta^1} \to A^{\Lambda^2_1}$ is an equivalence, which is exactly the condition that A is left Segal. Now we use proposition 3.1.6 to show that the right map is an equivalence if and only if the left one is. The top map is an equivalence by lemma 3.3.19 and the bottom map is can easily be seen to be an equivalence. □

We may recover the statement of Riehl and Shulman by the observation that every left Segal type is Segal (since the horn inclusion is a retract of the left segal inclusion), and that every simplicial, Segal type is left Segal. We also remark that this proof can be dualised for right Segal types and right fibrations.

3.4 Category Theory

3.4.1 Yoneda

The Yoneda lemma is a cornerstone theorem in category theory, and we follow Riehl and Shulman [RS17] in our formalisation, starting with a surprisingly trivial lemma:

Lemma 3.4.1 ([RS17, Thm 9.7], ⚙). *Given a covariant family $F : A \to \mathsf{Type}$, if $x : A$ is an initial object, then the evaluation map $(\Pi_a F(a)) \to F(x)$ is an equivalence.*

Proof. Since x is initial, we have a $\mathrm{unique}_a : \mathrm{Hom}(x, a)$. Then covariant transport gives us a map $F(x) \to F(a)$, which allows us to construct the inverse map. The uniqueness of covariant transport, and of the morphisms out of x then tells us that this map is both a left and right inverse. □

Corollary 3.4.2 (The dependent Yoneda lemma, ⚙). *Given a left segal type A, and a covariant family $F : A_{a/} \to \mathsf{Type}$ over the coslice of A at any $a : A$, then evaluation at (a, id) determines an equivalence $(\Pi_{(b,f)} F(b, f)) \xrightarrow{\sim} F(a, \mathrm{id})$.*

Corollary 3.4.3 (Yoneda, [RS17, p. 9.1], ⚙). *Given a left Segal type A, and a covariant family F over A, the evaluation map $\mathrm{ev\text{-}id}_a : (\Pi_{\{x:A\}} \mathit{Hom}(a, x) \to F(x)) \to F(a)$ is an equivalence.*

Proof. We observe that if $F : A \to \mathsf{Type}$ is covariant, then the base change $\pi_1^* F : A_{/c} \to \mathsf{Type}$ must also be covariant, so lemma 3.4.1 applies.

□

3.4.2 Universal constructions

Universal constructions are ubiquitous in categorical thinking. We make a choice of framework (which is far from unique) in which we can define a host of different universal properties.

Definition 3.4.4 (⚙). Given a covariant family F over a type C, we write $\int F$ for the sigma type $\Sigma_c F(c)$ in an analogy to the category of elements construction in traditional category theory. Given an element $(x, x') : \int F$, covariant transport induces a map $\Pi_{a:C} \mathrm{Hom}(x, a) \to F(a)$. (x, x') is said to be a universal element of F if this induced map is a fibrewise equivalence. ◊

This definition has an obvious dualisation for universal elements of contravariant families, for which we call elements with the dual universal property co-universal elements. In the following, when necessary, let all types be strong Segal so that the (co)representable families are all co(ntra)variant.

Example 3.4.5 (Coproducts, ⚙)**.** Let x and y be two elements of C, and let F be the contravariant family that sends $c : C$ to $\mathrm{Hom}(x, c) \times \mathrm{Hom}(x, c)$. Now suppose $(p, (\iota_1, \iota_2))$ determines an element $\Sigma_{p:C} F(p)$, and observe that the fact that this element is universal expresses the usual property that p is a coproduct of x and y: $\Pi_{a:C} \mathrm{Hom}(p, a) \xrightarrow{\sim} (\mathrm{Hom}(x, a) \times \mathrm{Hom}(y, a)))$. ◊

Example 3.4.6 (Colimits, ⚙)**.** One way of phrasing colimit properties is asking that the diagonal functor $\Delta : C \to C^{\mathcal{I}}$ has a left adjoint. If this adjoint exists, then C has all colimits of shape $\mathcal{I}$. Instead, we start with a functor $F : \mathcal{I} \to C$, we can ask for a universal object of the family $\mathrm{Hom}(F, \Delta -)$. A universal object is an object $\mathrm{colim}(F)$ and a natural transformation $\pi : \mathrm{Hom}(F, \Delta_{\mathrm{colim}(F)})$ such that the induced family of maps $\mathrm{Hom}(\mathrm{colim}(F), a) \to \mathrm{Hom}(F, \Delta_a)$ is an equivalence for all a. ◊

Example 3.4.7 (Adjunctions)**.** Given $R : D \to C$ and $c : C$, consider the covariant family $\mathrm{Hom}(c, R-)$. Let (l, η) be an element $\Sigma_l \mathrm{Hom}(c, Rl)$, if this is element is universal, then $\Pi_d \mathrm{Hom}(l, d) \xrightarrow{\sim} \mathrm{Hom}(c, Rd)$. Suppose there is a corresponding universal element for each $c : C$, then this data assembles into a map $L : C \to D$ and a natural transformation $\eta_X : \mathrm{Hom}(X, RLX)$ such that $L \dashv R$. Similarly, (co)-universal elements for $Hom(R-, c)$ determine a right adjoint to R. ◊

Example 3.4.8 (Kan extensions)**.** Let p be some map $A \to B$. In HoTT, for a particular type D the fibre of the map $p^* : D^B \to D^A$ at a map $f : D^A$ are given by maps $f' : D^B$ that *extend* f along p - in the sense that $f'p \sim f$. Then p^* is an equivalence just when extension problems along p have unique extensions for all maps $f : D^A$. In the setting of synthetic category theory, we can weaken the requirement of p^* being an equivalence to just requiring a left (right) adjoint. This means that extensions exist not uniquely, but universally: there is a natural transformation from the universal extension to any other. In the categorical literature, these are called left (right) Kan extensions. Just as in the case of limits we can turn a global property in terms of maps having adjoints into a local one, allowing us to express that certain universal extensions exist, without requiring universal extensions of any possible map.

Given a p as above together with a particular map $f : A \to D$, consider the covariant family $\mathrm{Hom}(f, p^* -)$. An element of this family is a map $f' : B \to D$ together with a natural transformation $\mathrm{Hom}(f, f'p)$, such that there is a family of equivalences $\Pi_{g:D^B} \mathrm{Hom}(f', g) \xrightarrow{\sim} \mathrm{Hom}(f, gp)$. A universal element is often written $\mathrm{Lan}_p(f)$. ◊

We hope to prove an analogous theorem to the fundamental theorem of identity types, where we have that for any $x : C$, and map $f : \Pi_a(a = x) \to F(a)$, f is a fibrewise equivalence iff the type $\Sigma_a F(a)$ is contractible. In the directed setting we ask not that the total type is contractible, but instead that it has an initial object. We first endeavour to characterise the morphisms in the category of elements of some contravariant family F.

Lemma 3.4.9 (⚙)**.** $\mathrm{Hom}((x, x'), (y, y')) \simeq \Sigma_{f:\mathrm{Hom}(x,y)} F(f)(x') = y'$

Proof. Since we are proving a fibrewise equivalence over x, x', y, y', we can instead show that there is an equivalence of total types. We calculate:

$$\begin{aligned}(\Sigma_a F(a))^I &\simeq \Sigma_{f:A^I} \Pi_i F(f(i)) \\ &\simeq \Sigma_{f:A^I} \Sigma_{x'} \Sigma_{y'} \operatorname{Hom}_F^f(x', y') \\ &\simeq \Sigma_{f:A^I} \Sigma_{x'} \Sigma_{y'} F(f)(x') = y' \\ &\simeq \Sigma_{x,y} \Sigma_{f:\operatorname{Hom}(x,y)} \Sigma_{x',y'} F(f)(x') = y' \\ &\simeq \Sigma_{(x,x'),(y,y')} \Sigma_{f:\operatorname{Hom}(x,y)} F(f)(x') = y'\end{aligned}$$

We note that the equivalence on the second third line makes use of the observation that $\operatorname{Hom}_F^f(x', y') \simeq (F(f)(x') = y))$. This follows from an application of the fundamental theorem of identity types to the family $\operatorname{Hom}_F^f(x', -)$. ⚙ □

Proposition 3.4.10 (⚙). *An element is universal iff it is an initial object in the category of elements.*

Proof. Let $(x, x') : \int F$ be an element, and first suppose that the map induced map $\Pi_a \operatorname{Hom}(x, a) \to F(a)$ is a fibrewise equivalence. Then there is an equivalence of total types, $C_{x/} = \Sigma_a \operatorname{Hom}(x, a) \to \Sigma_a F(a)$, which takes the initial object (x, id) to (x, x') witnessing its initiality as desired.

Now suppose (x, x') is an initial object, then we need to show that the fibrewise map $\alpha : \Pi_a \operatorname{Hom}(x, a) \to F(a)$ is an equivalence. Consider the fibre of this map at some element (a, a'):

$$\begin{aligned}fib_{\alpha_a}(a') &= \Sigma_{f:\operatorname{Hom}(x,a)} \alpha_a(f) = a' \\ &\simeq \Sigma_{f:\operatorname{Hom}(x,a)} F(f)(x') = a' \\ &\simeq \operatorname{Hom}((x, x'), (a, a'))\end{aligned}$$

Then by the fact that (x, x') is initial, the fibre is contractible, and so the map is a fibrewise equivalence. □

Remark 3.4.11. Whilst the notion of univalent categories is not yet formalised in our library, we make the following remark regarding universal elements in univalent categories. It follows from prop 3.4.10 that a universal element is unique up to unique isomorphism in a precategory or actually *unique* if it is in fact a category. It also follows that for co/contravariant families over a category, it is a proposition for the family to be representable. This fact, together with the construction of $\mathcal{S}$, would allow us to define the opposite of a univalent category as the full-subcategory of the co-representable co-presheaves even if in general, we do not postulate an opposite type constructor. ◊

3.5 Constructing $\mathcal{S}$

In this section we construct a type $\mathcal{S}$ together with a map $\mathcal{S} \to \mathcal{U}$, for which the objects are groupoids and crisp maps $A \to \mathcal{S}$ are in one to one correspondence with covariant families $A \to \mathcal{U}$. We then show that this type satisfies *directed univalence*, that the

morphisms in $\mathcal{S}$ are exactly the functions between groupoids. We first introduce the theory of *tiny types*, initially considering the external notion, and then exploring the techniques for internalising this into type theory.

Definition 3.5.1. In a cartesian closed category C, for each object I, there is an adjunction $- \times I \dashv -^I$. The object is called *internally-tiny* or sometimes *atomic* if the functor $-^I$ has a further right adjoint, which we will call $\sqrt[I]{-}$. This right adjoint is also referred to as *the amazing right adjoint* - terminology which stems from synthetic differential geometry. ◊

In any presheaf category over a cartesian base, the representable functors are all tiny objects, and furthermore, tiny objects are closed under retracts. Tiny objects are well behaved in the setting of presheaves over a cartesian base category; and this is what forces us to work in the category of cubical spaces, whilst using the notion of simplicial types to recover the behaviour we expect from categories. We would like to axiomatize a tiny type internally; however, in [LOPS18], a no-go theorem is proved, stating that any right adjoint to $-^I$ is isomorphic to the identity functor, and hence that I is isomorphic to the terminal object. The solution to this problem is to make use of the interpretation of the $\flat$ modality as the global sections functor, and ask only for a 'global' right adjoint $\flat(\Gamma^I \to A) \cong \flat(\Gamma \to A_I)$ for crisp types Γ.

Definition 3.5.2 (⚙)**.** Fix a type I, and for each $A : \mathcal{U}$, let A_I be a type with a map $\epsilon : (A_I)^I \to A$. We say that this data (A_I, ϵ) is a tiny structure if the induced map:

$$\flat(\epsilon \circ (-)^I) : \flat(\Gamma \to A_I) \to \flat(\Gamma^I \to A)$$

is an equivalence. If for all A, there is a pair (A_I, ϵ) which is a tiny structure, then we say I is a tiny type. ◊

By convention, with I a tiny type, given $x : \flat(\Gamma^I \to A)$, we write $\bar{x} : \flat(\Gamma \to A_I)$ for the inverse to the defining equivalence. Then for a crisp $x :^\flat \Gamma^I \to A$, $\epsilon \circ (\bar{x}^I)$ is homotopic to x. To demonstrate working with a tiny type former, we prove the following lemma.

Lemma 3.5.3. *Suppose I is tiny type, then $(-)_I$ is a limit preserving functor* $\mathsf{Type} \to \mathsf{Type}$.

Proof. First we give the action of $(-)_I$ on morphisms as the composite:

$$\flat(A \to B) \xrightarrow{\flat(\epsilon^*)} \flat((A_I)^I \to B) \xrightarrow{(\bar{-})} \flat(A_I \to B_I)$$

and note that it satisfies the following naturality law:

$$\begin{array}{ccc} \flat((A_I)^I) & \xrightarrow{\epsilon} & \flat A \\ {\scriptstyle (f_I)^I}\downarrow & & \downarrow{\scriptstyle \flat f} \\ \flat((B_I)^I) & \xrightarrow[\epsilon]{} & \flat B \end{array}$$

Now the standard argument for right adjoints shows that it preserves limits, for example, for any type Γ, we have the following chain of natural equivalences.

$$\begin{aligned}
\flat(\Gamma \to (A \times B)_I) &\simeq \flat(\Gamma^I \to A \times B) \\
&\simeq \flat(\Gamma^I \to A \times \Gamma^I \to B) \\
&\simeq \flat(\Gamma^I \to A) \times \flat(\Gamma^I \to B) \\
&\simeq \flat(\Gamma \to A_I) \times \flat(\Gamma \to B_I) \\
&\simeq \flat(\Gamma \to A_I \times \Gamma \to B_I) \\
&\simeq \flat(\Gamma \to A_I \times B_I)
\end{aligned}$$

□

Lemma 3.5.4. *Tiny types are closed under products and retracts.*

Proof. If I and J are both tiny types, then we may immediately see that $A_{I\times J} := (A_I)_J$ can be equipped with a universal map making it into a tiny structure. Define $\epsilon : (A_{I\times J})^{I\times J} \to A$ to be the composite:

$$(((A_I)_J)^J)^I \to (A_I)^I \to A$$

For the case of retracts we refer the reader to [SGWB, Lemma A.2]

□

Corollary 3.5.5. *Supposing that Δ^1 is a tiny type, then it follows that all cubes I^n are tiny, as are the standard simplices Δ^n.*

3.5.1 Dependent right adjoints

We assume I is a tiny type. The goal of this section is to show that we can extend the *amazing right adjoint* to a dependent version and give a procedure for constructing classifying maps for certain *nice* classes of maps.

Definition 3.5.6 (⚙). For some class of maps $\mathcal{F}$, we say that a map $U^\bullet_{\mathcal{F}} \to U_{\mathcal{F}}$ classifies $\mathcal{F}$ it is terminal in the category with objects maps in $\mathcal{F}$, and morphisms given by cartesian squares. ◊

Example 3.5.7 (⚙). For each universe $\mathcal{U}$, the projection $\mathcal{U}^\bullet \xrightarrow{\pi} \mathcal{U}$ from pointed types, classifies the class of $\mathcal{U}$-small maps. ◊

Proof. We want to show that the type of cartesian squares $f \to \pi$ is contractible when $f : E \to B$ is $\mathcal{U}$-small. We first can employ univalence to straighten this type into a map of families where the dependent map is an equivalence. Now $\mathsf{Fam}_{\mathrm{cart}}(\mathit{fib}_f, \mathcal{U}^\bullet)$, can be unpacked to the data of a map $P : B \to \mathcal{U}$ together with a fibrewise equivalence $\Pi_b\ \mathit{fib}_f(b) \simeq P(b)$. By the type-theoretic principal of choice, we see that this is equivalently the type $\Pi_b\ \Sigma_{P:\mathcal{U}}\ \mathit{fib}_f(b) \simeq P$, which is nothing but the proposition that f is a $\mathcal{U}$-small map. □

Proposition 3.5.8 (⚙). *There is a type former $\sqrt{-} : \flat(\Gamma^I \to \mathsf{Type}) \to \flat(\Gamma \to \mathsf{Type}^+)$ such that crisp sections $\Pi_\Gamma \sqrt{A}$ are in bijection with sections $\Pi_{\Gamma^I} A$.*

Proof. By the functoriality of $-_I$, we have a $\mathcal{U}^+$-small map $U_I^\bullet \xrightarrow{\pi_I} U_I$. By lemma 3.5.7, we then have a unique cartesian square from π_I to π, and so we have an equivalence of lifts of π_I and of π. Now since $\mathcal{U}^+$ is a universe, lifts of π correspond exactly to sections (Π types), so it remains to show that a lift π_I is equivalently a section $\Pi_x A$. The lift in question can be seen pictorially as on the left; we can extend the adjunction to transpose the lift as in the rightmost diagram. Then again since $\mathcal{U}$ is a universe, we get a section $\Pi_{x:\Gamma^I} A(x)$.



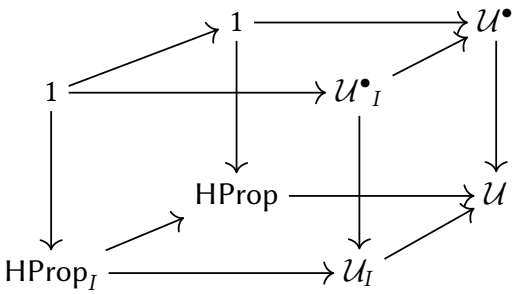
□

Remark 3.5.9. In [GWB26a], it is shown that under the axiom cubes-separate, that the map $\mathcal{U}^\bullet{}_I \to \mathcal{U}_I$ is $\mathcal{U}$-small and hence no bump of the universe level is needed. For the sake of maximum generality in the formalisation we do not assume cubes separate, and hence our dependent amazing right adjoint lives in the successor universe. It would be interesting to investigate if it can be shown that this map is $\mathcal{U}$-small without any additional assumptions. ◊

Definition 3.5.10 (Amazing conditions, ⚙)**.** We know that the map $1 \to \mathsf{HProp}_{\mathcal{U}}$, which picks out the true proposition is a classifying map for $\mathcal{U}$-small embeddings. We can repeat the construction of 3.5.8 but now for the amazing right adjoint for families of propositions. We may show that $1 \xrightarrow{\top_I} \mathsf{HProp}_I$ is a $\mathcal{U}^+$-small embedding and hence there is a unique cartesian square $\mathsf{Arr}_{\mathsf{cart}}(\top_I, \top)$. This gives us a map $\mathsf{is\text{-}amazingly} : \mathsf{HProp}_I \to \mathsf{HProp}$ such that for any crisp family of propositions $A : (\Gamma^I) \to \mathsf{HProp}$, $\forall(x : \Gamma^I), A(x) \iff \forall(x : \Gamma), \mathsf{is\text{-}amazingly}(A)(x)$. ◊

Proposition 3.5.11. *If $A : \Gamma^I \to \mathsf{Type}$ is a family of propositions, then $\sqrt[I]{A}$ factors through* HProp *via is-amazingly(A).*

Proof. We generalise and show that the following cube commutes:



Notice that each vertical square in this diagram is cartesian, since all vertical maps are small, and the vertical maps on the left face are small-embeddings. Thus the back squares and the front squares are both cartesian maps of arrows into $\pi_{\mathcal{U}}$, and by the universality of $\mathcal{U}$, such maps are unique. □

Definition 3.5.12 (I relative families of maps, ⚙). Let $\mathcal{F}$ be some class of maps given by $\mathsf{Arr} \to \mathsf{HProp}$. We say that $\mathcal{F}$ is I-relative if for any map $f : E \to B$, $f \in \mathcal{F}$, if and only if after pulling back by each $\sigma : I \to B$, we have $\sigma^* f \in \mathcal{F}$. ◊

Clearly any I-relative family is closed under change of base. We remark that in the case $I = 1$ such relative propositions are simply the 'fibrewise' conditions. Such examples are numerous, including embeddings and equivalences, as well as more generally classes of maps null with respect to a family of types. For such properties we may always build universes that classify these maps and the following theorem generalises this situation (since 1 is always a tiny type with the right adjoint given by the identity functor).

Theorem 3.5.13 (⚙). *Fix an I-relative family of maps, writing $P \in \mathcal{F}$ for $P : E \to A$. There are universal maps $\mathcal{U}_{\mathcal{F}}^{\bullet} \to \mathcal{U}_{\mathcal{F}}$ which classify $\mathcal{U}$-small maps in $\mathcal{F}$.*

Proof. Define $\mathcal{U}_{\mathcal{F}}$ to be the types $A : \mathcal{U}$ such that, after applying the dependent right adjoint, where A becomes a map $I \to \mathcal{U}$, the un-straightening is a map in $\mathcal{F}$. □

3.5.2 Amazing covariance

Now with the theory of tiny types in hand, we instantiate it to the case of $I = \Delta^1$.

Axiom 3 (⚙). *Δ^1 is tiny*

Following Gratzer, Weinberger, and Buchholtz [GWB26a], we define amazing covariance (contravariance), a property of a type that can be seen as saying a type is covariant (contravariant) over the entire context. We show that this property satisfies a universal property that any crisp families of types $P :^{\flat} A \to \mathcal{U}$ has amazingly covariant fibres iff it is covariant.

Definition 3.5.14 (Amazing covariance, ⚙). A type X is amazingly covariant if after applying the root former, whereby X becomes an I indexed family of types, X is covariant. ◊

Definition 3.5.15 (Spaces, ⚙). We define $\mathcal{S}$ to be the type $\mathcal{U}_{\mathcal{F}}$ constructed in thm 3.5.13, where $\mathcal{F}$ is the class of left fibrations. The first projection into U is a covariant family of types, and any crisp covariant family $P : A \to \mathcal{U}$ lifts to a map into $\mathcal{S}$. ◊

Lemma 3.5.16. *$\mathcal{S}$ is univalent. Equivalences of the underlying type form an identity system for $\mathcal{S}$.*

Proof. By proposition 3.5.11, we get that the map $\mathcal{S} \to \mathsf{Type}$ is an embedding, and we can therefore pull the identity system on Type back to get our desired identity system. □

Theorem 3.5.17 (Closure of amazing covariance, ⚙). *Following [GWB26a], we formalise a host of closure properties of amazing covariance. Suppose $A\ B : \mathcal{S}$, $P : A \to \mathcal{S}$, then the following types are also in $\mathcal{S}$:*

1. *$A \times B$*

2. $\Sigma_{a:A} P(a)$
3. $a =_A b$ *for all* $a\ b : A$
4. *Assuming duality,* $i =_{\mathbb{I}} 0$ *for any* $i : \mathbb{I}$
5. $X \to A$ *for an amazingly* ***contravariant*** *type* X

Proof. The proof strategy for all of the results are the same: in general we may reduce the problem to the analogous closure conditions for covariant families. As an example, we provide the proof for closure under equality.

Let Γ be the type $\Sigma_{A:\mathcal{S}} A \times A$, and σ the map $\Gamma \to \mathrm{Type}$ defined as $(A, a, b) \mapsto a = b$. We need to show that $\Pi_{x:\Gamma} \mathrm{isAcov}(\sigma x)$, which by unfolding the definition of amazing covariance is just $\Pi_{x:\Gamma} \sqrt[!]{\text{is-covariant}}(\sigma x)$. By naturality of the $\sqrt[!]{-}$ former, this is the same as $\Pi_{x:\Gamma} \sqrt[!]{\text{is-covariant} \circ \sigma_*}(x)$, and we may then use the universal property of $\sqrt{-}$ so that it suffices to give an element $\Pi_{x:\Gamma^{\Delta^1}} \text{is-covariant}(\sigma_* \circ x)$.

Now we have dealt with the root former, we may proceed less formally. We have a map $A : \Delta^1 \to \mathcal{S}$, which means A is a covariant family of types over Δ^1, and a pair of sections $a\ b : \Pi_{\Delta^1} A$, we want to show $i \mapsto a_i = b_i$ is covariant, and this follows by general closure properties of orthogonality. □

3.5.3 Directed univalence

We now have all the prerequisites to formalise directed univalence. In this section we will make the following assumptions, in addition to Axiom 1, 2 and 3:

Axiom 4 (Global points of Δ^1)**.** *The inclusion of endpoints* $2 \to \Delta^1$ *induces an equivalence on* $\flat$*-types. This means that for crisp elements of* Δ^1 *we may use the induction principal of the booleans to define sections* $\Pi_{a:\Delta^1} P(a)$ *just by giving* $P(0)$ *and* $P(1)$*.*

Axiom 5 (Cubes separate)**.** *Given a crisp map* $f : A \to B$*, if the induced map* $\flat(I^n \to A) \to \flat(I^n \to B)$ *is an equivalence, then* f *itself is an equivalence.*

Remark 3.5.18. In the formalisation we assume a slightly stronger axiom, namely that cubes detect continuity, in the sense of Myers and Riley [MR23]. ◊

Axiom 6 (Detecting connectivity)**.** *The* $\flat$ *discrete types are exactly the* Δ^1 *null types. In particular, we have that all propositions are* $\flat$ *discrete.*

We begin with some lemmas.

Lemma 3.5.19 (⚙)**.** *Given a pair of families* $F\ G : \Delta^1 \to \mathcal{S}$*, a natural transformation* $\alpha_i : F_i \to G_i$ *is an equivalence if the restriction of* α *at each point* $0, 1 : \Delta^1$ *is an equivalence.*

The proof of which is almost an exact translation of [GWB26a], and makes essential use of Axiom 5. We also require the following lemma, the proof of which is true in type theory with an interval without needing to assume duality (axiom 2) or any other axiom, however the assumption of duality somewhat simplifies the proof.[2]

Lemma 3.5.20 (⚙)**.** *Given a covariant family,* $F : \Delta^1 \to \mathrm{Type}$*, and any* $i \leq j$*, we may define a function* $coe_{i \leq j}(F) : F(i) \to F(j)$ *such that* $coe_{i \leq i}(F) \sim id$ *and* $coe_{0 \leq 1}(F) \sim F_*(\mathrm{id})$*, where* id *is viewed as a morphism* $\mathrm{Hom}_{\Delta^1}(0, 1)$

[2] For the proof without duality, see the formalisation

Proof. Fix $i \leq j$, consider the map $p : \Delta^1 \to \Delta^1$ which sends $l \mapsto (l \vee i) \wedge (i \vee j)$. By the general algebraic properties of lattices, we may show that at the endpoints, $p(0) = i$ and $p(1) = j$, and hence by p determines a morphism in Δ^1 from i to j. Now F is covariant so we can transport along this morphism to obtain our desired map. Duality says that Δ^1 is a preorder, so $p_{i\leq i}$ must be id_i and then functoriality of covariant transport tells us this must be the identity map. Similarly for $p_{0\leq 1} = \mathrm{Hom}(0, 1)$. □

Theorem 3.5.21 ([GWB26a, Thm 6.13], ⚙). *$\mathcal{S}$ is directed univalent. That is, we have an equivalence*

$$\mathrm{Hom}_{\mathcal{S}}(A, B) \simeq A \to B.$$

Proof. There is a canonical map in the forward direction given by covariant transport. We show that this map is an equivalence on total types $\mathcal{S}^{\Delta^1} \xrightarrow{\sim} \Sigma_{A:\mathcal{S}}\Sigma_{B:\mathcal{S}}B^A$.

The inverse map is given by the type family $\mathrm{Gl}(A, B, f) : \Delta^1 \to \mathcal{S}$ defined by $i \mapsto \Sigma_{b:B}(i = 0) \to \mathit{fib}_f(b)$. Now we know this map actually lands in $\mathcal{S}$ by the closure properties in 3.5.17.

Left inverse Given $A\ B : \mathcal{S}$ and a map $f : A \to B$, we need to show that

$$(Gl(A, B, f)(0), Gl(A, B, f)(1), Gl(A, B, f)_*) = (A, B, f).$$

By characterisations of paths in sigma types and in $\mathcal{S}$, we need equivalences $e_1 : Gl(A, B, f)(0) \simeq A$ and $e_2 : Gl(A, B, f)(1) \simeq B$ such that $e_2 Gl(A, B, f)_* \sim f e_1$. For e_1 we may take the evaluation of the second projection at refl, which is an equivalence by the familiar fact that the type $\Sigma_b \mathit{fib}_f(b)$ is equivalent to the domain of f. For e_2 the first projection $\Sigma_b(1 = 0) \to \mathit{fib}_f(b)$ is an equivalence since $1 = 0 \to Q$ is a singleton. What remains then is to show that for any $b : B$, and $k : 0 = 0 \to \Sigma_{a:A}(f(a) = b)$ $\pi_1 Gl(A, B, f)_*(b, k) = f(\pi_1(k(\mathrm{refl})))$.

Now, since $0 = 0$ is a singleton (Δ^1 is a set), we may assume elements $a : A$ and $p : f(a) = b$. We may then use identity induction on p to assume b is just $f(a)$ and p is refl. Then we need to show that $\pi_1(Gl(A, B, f)_*(f(a), (a, \mathrm{refl}))) = f(a)$. To this end we may use the uniqueness of covariant transport and construct a dependent morphism over $Gl(A, B, f)$ from some $x : Gl(A, B, f)(0)$ to $(\pi_1(x), 0 \neq 1)$. It is easy to check that such a morphism is given by the constancy at $\pi_1(x)$, $\Delta_{(\pi_2(\mathrm{refl}))}$.

Right inverse We hope to show that for any map $f : \Delta^1 \to \mathcal{S}$,

$$Gl(f(0), f(1), f_*(0 \leq 1)) = f.$$

By univalence, it suffices to give a fibrewise equivalence from $f(i)$ to $Gl(f(0), f(1), f_*)(i)$. We first will construct such a map and use lemma 3.5.19 to check it is an equivalence. Suppose we are given $x : f(i)$, then we need to give an element of $\Sigma_{b:f(1)} i = 0 \to \mathit{fib}_{f_*}(b)$. The first component can be given by coercing x along the morphism $i \leq 1$. To give the second component, we may assume i is 0, whence we can give $x : f(0)$ since transporting along the morphisms $i \leq 1$ where $i = 0$ is the same as transporting along $\mathrm{Hom}_{\Delta^1}(0, 1)$ by lemma 3.5.20. It can be checked that these maps are equivalences at the 0 and 1 endpoints. □

Remark 3.5.22. Whilst this result has been formalised in our library, we make a pair of assumptions in addition to the axioms stated in this section. Namely, that for any amazingly covariant family A and $i : \Delta^1$, the family $(i = 0) \to A(-)$ is amazingly covariant; and that for a family of propositions P, $\sqrt[\prime\prime]{P}$ is also a family of propositions. The latter result should follow from the fact amazing right adjoints preserve pullback, an argument which we hope to formalise soon. The former assumption follows from (4.) and (5.) of theorem 3.5.17. Whilst (4.) follows straightforwardly from duality, the proof of (5.) found in [GWB26a] provides two challenges to formalisation. Firstly, it makes use of the op modality, which we do not use in our formalisation. This is however easily overcome by the notion of amazingly contravariant types; even if we do not assume that opposite types exist in general, we may use the classifier of right-fibrations to construct the opposite of $\mathcal{S}$. The second challenge lies in the fact that the informal proof sweeps some of the coherence under the rug, as is common in the informal HoTT style. We suspect that in a system like Rzk [KRW23] with a strict interval that this proof would formalise without issue, but so far in our setting, this proof has been extremely challenging to formalise in full detail; however, we expect that a higher level proof that argues via the adjoint relationship between pullback and pushforward should give us an argument that formalises more easily. This is currently left for future work. ◊

Chapter 4

Conclusion

4.1 Technical Evaluation

4.1.1 Formalisation

As discussed in the introduction, one of the goals of this project was to assess the feasibility of formalising simplicial type theory in Agda – without using any custom type theory like in the Rzk proof assistant. I believe this project demonstrates that it is both possible and practical to use plain Agda for simplicial type theory. We briefly list some of the advantages and challenges that we discovered throughout the process of the development.

Computation and the opaque pragma In the initial planning stages, an important question was: "How can we represent the interval and should it obey any definition equalities?". Agda has a feature which allows users to mark certain equalities as *rewrite rules*, allowing propositional equalities to be upgraded to definitional equalities. In principle this would give us some of the power of working in a system like Rzk. There are limitations to this system however, for example using them would have forced us to postulate a global interval, instead of the approach we ultimately decided to take, making our theories generic over an arbitrary distributive lattice. Throughout the formalisation process, we discovered that the opposite of our initial instinct served us better in the long term – namely instead of trying to make things compute as much as possible, we should limit the computation. Fortunately, Agda also supports this; by marking a definition as opaque, we prevent the system from *unfolding* or normalising this term. Judicious use of opaque has numerous advantages: It can speed up type checking times, improve goal display, and can eliminate the tight coupling of code. This latter point is particularly important as the library grows, making definitions opaque, but exposing the essential behaviour via interfaces means that one can refactor the precise proofs and definitions without needing to change down-steam usage – this

problem is exacerbated by working in homotopy type theory, where even *proofs* can carry meaningful data.

Relatedly, we also noticed that certain proofs from the original work on simplicial type theory [RS17] were particularly challenging to translate to our setting, even if they had been successfully formalised in the Rzk proof assistant. The proofs that would typically cause most trouble were those that explicitly relied on manipulating extension types, which are strict in the setting of Riehl and Shulman, but not ours. For instance, the data of a square $\square \to X$ which extends some boundary $\partial\square^2 \to X$ in the strict setting consists of only the data of the square, subject to strict side conditions. In the non-strict setting, we have the data of the square, the paths witnessing the extension of the left, right, top and bottom edges, as well as further coherence witnessing that the paths agree at each vertex. As a result, we often needed to find alternative arguments and we have found the tools of orthogonality particular useful in this regard. Compare our proof of theorem 3.3.20 with the original proof [RS17, Prop 8.13] to see an example of a typical difference in strategy.

4.1.2 Modal extensions to Agda

Initially a larger and more ambitious extension to the Agda proof assistant was envisioned and prototyped. Whilst this prototype was functioning, there were some bugs in the implementation that I was not fully able to iron out, especially due to my relatively small experience with the actual codebase of Agda, which is both large and technically complicated. As a proof assistant, bugs in Agda can lead to invalid proofs being validated, or worse, soundness bugs which allow a user to prove an absurd statement, such as $1 = 0$; this in turn means that any theorem may be proved trivially. Consequently, correctness is highly valued in the codebase and so, in discussion with the Agda maintainers and development team, it was agreed that this extension was somewhat too ambitious and experimental to be merged into the codebase in its current state. Over the course of the project, I had discovered that a slightly simpler modal type theory, called cohesive type theory was sufficient to prove many of the desired results. Whilst it did not incorporate an 'op' modality, the proof of directed univalence [GWB26a] could nonetheless be adapted to this setting. This prompted the second experimental extension of Agda being prototyped, and with some guidance from the Agda team, the bugs, which were of a similar nature to the first extension, were able to be ironed out. This extension was upstreamed and will be part of the forthcoming release of Agda 2.9.0.

This extension to Agda may find use not only in simplicial type theory, but in a wide array of contexts. Shulman first introduced cohesive type theory for the purpose of doing algebraic topology internally to HoTT, a programme called real-cohesive type theory. More recently, Myers and Riley [MR23] discuss a wide array of applications for cohesive type theory, although the main focus of that article is to develop a system where multiple 'cohesions' can be used simultaneously. We note that this could be an alternative future direction for this extension to Agda.

4.2 LSEPI

Due to its theoretical nature, this project does not directly address issues of laws, social, ethical or professional issues; however, the project contributes an open source library and resource, a contribution to an open source programming language and proof assistant (Agda) as well as contributing more generally to our mathematical understanding of a field which is at the forefront of a number of technical areas of mathematics. Finally, we remark that the free online learning resource also benefits the democratisation of mathematics, a cause which proof assistants have naturally contributed to, particularly in recent years.

4.3 Future outlook

There are many categorical notions that have been explored in SHoTT, including adjunctions and Kan extensions [RS17], limits and colimits [Mar24], opposite categories and full subcategories, and the generalisation of covariant families to cocartesian fibrations [Wei22]. We would like to extend our formalisation to include the theory of cocartesian fibrations, so that we can construct the category of categories, following a recent preprint of Gratzer, Weinberger, and Buchholtz [GWB26b]. In addition, with the construction of $\mathcal{S}$ in hand, there are many areas of category theory that have yet to be explored in SHoTT. Of particular interest is the meta theory of type theory; whilst we may understand models of HoTT via model categories (a 1-categorical presentation of an ∞-category), it would be desirable to have a native ∞-categorical story of this theory. Nguyen and Uemura [NU22] began this investigation studying ∞-categorical models of intentional type theory, and we believe aspects of this work could be adapted to the settings of simplicial type theory, so that even if HoTT cannot eat itself, simplicial type theory may be able to eat HoTT.

Declaration of AI Usage

No AI tools were used in the writing or preparation of this document, nor in the development of software artefacts.

Bibliography

[Buc19] Ulrik Buchholtz. "Higher Structures in Homotopy Type Theory". In: *Reflections on the Foundations of Mathematics: Univalent Foundations, Set Theory and General Thoughts*. 2019, pp. 151–172 (cit. on p. 2).

[CCH24] Felix Cherubini, Thierry Coquand, and Matthias Hutzler. "A Foundation for Synthetic Algebraic Geometry". In: *Mathematical Structures in Computer Science* 34.9 (Oct. 2024), pp. 1008–1053. ISSN: 0960-1295, 1469-8072 (cit. on p. 2).

[CCNW26] Denis-Charles Cisinski, Bastiaan Cnossen, Kim Nguyen, and Taishi Walde. *Synthetic Higher Categories*. 2026 (cit. on p. 4).

[dJon25] Tom de Jong. "Formalizing Equivalences Without Tears". In: *30th International Conference on Types for Proofs and Programs (TYPES 2024)*. Vol. 336. 2025, 1:1–1:6 (cit. on p. 15).

[GKNB21] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. "Multimodal Dependent Type Theory". In: *Logical Methods in Computer Science* Volume 17, Issue 3 (July 2021), p. 7571 (cit. on pp. 8–9).

[Gra21] Daniel Gratzer. "Crisp Induction for Intensional Identity Types". 2021. URL: https://www.danielgratzer.com/papers/crisp-induction-for-intensional-identity-types.pdf (cit. on p. 12).

[Gra22] Daniel Gratzer. "Normalization for Multimodal Type Theory". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '22. New York, NY, USA: Association for Computing Machinery, 2022 (cit. on p. 8).

[GWB25] Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. *The Yoneda embedding in simplicial type theory*. en. May 2025. DOI: 10.1109/LICS65433.2025.00017 (cit. on pp. i, 5).

[GWB26a] Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. *Directed univalence in simplicial homotopy type theory*. July 2026. arXiv: 2407.09146 (cit. on pp. i, 4, 30–34, 36).

[GWB26b] Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. *The ∞-Category of ∞-Categories in Simplicial Type Theory*. Feb. 2, 2026. DOI: 10.48550/arXiv.2602.02218 (cit. on p. 37).

[Hau25] Rune Haugseng. *YET ANOTHER INTRODUCTION TO ∞-CATEGORIES*. Apr. 2025. URL: http://folk.ntnu.no/runegha/naivecat_web.pdf (cit. on p. 4).

[HS94] Martin Hofmann and Thomas Streicher. "The groupoid model refutes uniqueness of identity proofs". In: *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science (LICS 1994)*. July 1994, pp. 208–212 (cit. on p. 2).

[Joy08] Andre Joyal. *THE THEORY OF QUASI-CATEGORIES*. 2008. URL: https://math.uchicago.edu/~may/PEOPLE/JOYAL/0newqcategories.pdf (cit. on p. 3).

[Koc06] Anders Kock. *Synthetic Differential Geometry*. London Mathematical Society Lecture Note Series 333. 2006. ISBN: 978-0-521-68738-6 978-1-107-37007-4 (cit. on p. 2).

[KRW23] Nikolai Kudasov, Emily Riehl, and Jonathan Weinberger. *Formalizing the ∞-Categorical Yoneda Lemma*. 2023. arXiv: 2309.08340 [math.CT] (cit. on pp. i, 4, 34).

[LOPS18] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. "Internal Universes in Models of Homotopy Type Theory". en. In: *LIPIcs, Volume 108, FSCD 2018* 108 (2018), 22:1–22:17 (cit. on pp. 4–5, 28).

[Lur08] Jacob Lurie. *Higher Topos Theory*. arXiv, July 31, 2008. arXiv: math/0608040 (cit. on p. 3).

[Mar22] Louis Martini. *Yoneda's Lemma for Internal Higher Categories*. Apr. 1, 2022. DOI: 10.48550/arXiv.2103.17141. Pre-published (cit. on p. 3).

[Mar24] César Bardomiano Martínez. *Limits and Colimits of Synthetic ∞-Categories*. July 25, 2024. DOI: 10.48550/arXiv.2202.12386. Pre-published (cit. on p. 37).

[MR23] David Jaz Myers and Mitchell Riley. *Commuting Cohesions*. Jan. 31, 2023. DOI: 10.48550/arXiv.2301.13780 (cit. on pp. 32, 36).

[MW24] Louis Martini and Sebastian Wolf. *Colimits and Cocompletions in Internal Higher Category Theory*. Feb. 13, 2024. DOI: 10.48550/arXiv.2111.14495. Pre-published (cit. on p. 3).

[NA25] Jacob Neumann and Thorsten Altenkirch. "Synthetic 1-Categories in Directed Type Theory". In: *30th International Conference on Types for Proofs and Programs (TYPES 2024)*. 2025 (cit. on p. 3).

[Nor07] Ulf Norell. "Towards a practical programming language based on dependent type theory". PhD thesis. Chalmers University of Technology and Göteborg University, 2007 (cit. on p. 10).

[Nor18] Paige Randall North. *Towards a Directed Homotopy Type Theory*. July 27, 2018. DOI: 10.48550/arXiv.1807.10566 (cit. on p. 3).

[NU22] Hoang Kim Nguyen and Taichi Uemura. *∞-Type Theories*. May 2, 2022. DOI: 10.48550/arXiv.2205.00798. Pre-published (cit. on p. 37).

[Rez00] Charles Rezk. "A Model for the Homotopy Theory of Homotopy Theory". In: *Transactions of the American Mathematical Society* 353.3 (June 20, 2000), pp. 973–1007. DOI: 10.1090/S0002-9947-00-02653-2 (cit. on p. 3).

[Rij17] Egbert Rijke. *The Join Construction*. Jan. 26, 2017. arXiv: 1701.07538 [math]. Pre-published (cit. on p. 6).

[RS17] Emily Riehl and Michael Shulman. *A type theory for synthetic ∞-categories*. May 2017. DOI: 10.48550/arXiv.1705.07442 (cit. on pp. i, 14, 22, 24–25, 36–37).

[RSS20] Egbert Rijke, Michael Shulman, and Bas Spitters. "Modalities in homotopy type theory". In: *Logical Methods in Computer Science* 16 (2020). DOI: 10.23638/LMCS-16(1:2)2020 (cit. on pp. 6, 12, 22).

[RV22] Emily Riehl and Dominic Verity. *Elements of ∞-Category Theory*. 1st ed. Cambridge University Press, Jan. 31, 2022. ISBN: 978-1-108-93688-0 978-1-108-83798-9 (cit. on p. 3).

[SGWB] Rob Schellingerhout, Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. *For $\flat$-annotated types, Segal implies simplicial*. URL: https://www.danielgratzer.com/papers/segal-implies-simplicial.pdf. unpublished (cit. on p. 29).

[Shu18] Michael Shulman. *Brouwer's fixed-point theorem in real-cohesive homotopy type theory*. en. Apr. 2018. DOI: 10.1017/S0960129517000147 (cit. on pp. i, 5–6, 8, 11–12).

[Ste23] Jonathan Sterling. *Forester*. 2023. URL: https://www.forester-notes.org/0065/index.xml (cit. on p. 6).

[SY25] Jonathan Sterling and Lingyuan Ye. *Domains and Classifying Topoi*. May 19, 2025. DOI: 10.48550/arXiv.2505.13096. Pre-published (cit. on p. 19).

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013 (cit. on pp. 2, 14, 16).

[Wär25] David Wärn. *Path Spaces of Pushouts*. Aug. 5, 2025. DOI: 10.48550/arXiv.2402.12339. Pre-published (cit. on p. 16).

[Wei22] Jonathan Weinberger. "A Synthetic Perspective on $(\infty, 1)$-Category Theory: Fibrational and Semantic Aspects". 2022 (cit. on p. 37).

[WL20] Matthew Z. Weaver and Daniel R. Licata.
"A Constructive Model of Directed Univalence in Bicubical Sets".
In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. July 2020, pp. 915–928 (cit. on p. 4).