Syntax and semantics of modal type theory

Daniel Gratzer

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

Syntax and semantics of modal type theory

A Dissertation
Presented to the Faculty of Natural Sciences
of Aarhus University
in Partial Fulfillment of the Requirements
for the PhD Degree

by
Daniel Gratzer
August 29th, 2023

Abstract

One idiosyncratic framing of type theory is as the study of operations invariant under substitution. Modal type theory, by contrast, concerns the controlled integration of operations—modalities—into type theory which violate this discipline, so-called *non-fibered connectives*. Modal type theory is therefore built around a fundamental tension: the desire to include modalities and powerful principles for reasoning with them on one hand, and the need to maintain the conveniences and character of Martin-Löf type theory which stem from substitution invariance.

In this thesis, we thoroughly explore and discuss this contradiction. We discuss several different formulations of modal type theory, explore their various syntactic properties, and relate them through their categorical semantics. In particular, we show that most modal type theories that have arisen in the last two decades can be understood through the abstraction of *weak dependent right adjoints*. We also put forward a new *general* modal type theory, MTT, based on this abstraction.

The generality of MTT means that, without any additional work, it can be specialized to an arbitrary collection of type theories related by modalities and natural transformations between them. It is therefore easy to obtain a type theory for a comonad, an adjunction, a local topos, or any other number of complex and realistic scenarios. In addition to showing that many modal type theories are closely related to specific instantiations of MTT, we thoroughly explore the syntax and semantics of MTT itself. We prove that MTT enjoys an unconditional normalization result and decidable type-checking under mild assumptions. We show how MTT may be interpreted into a wide variety of structured categories and use this to study the expressive power of the type theory and various extensions thereof.

Finally, we explore several concrete applications of MTT in the context of guarded type theory and guarded denotational semantics. We propose a highly usable language for guarded recursion and explore its particular models and metatheorems. We show a relatively sharp result bounding the extent to which classical guarded recursion can be added to any type theory with decidable type-checking and propose a system to mitigate this issue. Finally, we conduct an in-depth case study using guarded MTT to obtain a fully synthetic account of the Iris program logic, proving adequacy in a fully internal manner.

Resumé

Typeteori kan betragtes som studiet af operationer, der er invariante under substitution. Modal typeteori bekymrer sig derimod om, hvordan operationer, der bryder med dette princip—såkaldte “non-fibered connectives”—kan integreres i typeteori gennem modaliteter. Modal typeteori er derfor bygget på en fundamental spænding: modaliteter og deres kraftfulde ræsonneringsprincipper ønskes integreret i typeteori, men det er nødvendigt at bevare bekvemmelighederne og karakteren af Martin-Löf typeteori, som stammer fra substitutionsinvariansen.

I denne afhandling udforsker og diskuterer vi dybdegående denne modstrid. Vi diskuterer flere formuleringer af modal typeteori, vi undersøger deres syntaktiske egenskaber, og vi relaterer dem gennem deres kategoriske semantik. Frem for alt viser vi, at de fleste modale typeteorier, der er opstået i løbet af de sidste to årtier, kan forstås gennem såkaldte “weak dependent right adjoints”. Vi fremsætter også en ny *generel* modal typeteori, MTT, baseret på denne abstraktion.

Generaliteten af MTT betyder, at teorien uden videre kan specialiseres til en vilkårlig samling af typeteorier, der er forbundet gennem modaliteter og naturlige transformationer mellem dem. Det er derfor nemt at opnå en typeteori for en comonade, en adjunktion, en lokal topos, eller i en række andre komplekse og realistiske scenarier. Udover at vise hvordan mange modale typeteorier er tæt forbundne til specifikke instantieringer af MTT, så udforsker vi også syntaksen og semantikken af selve MTT. Vi viser, at MTT overholder et uforbeholdent normaliseringsresultat og afgørligt typecheck under simple antagelser. Vi viser endvidere, hvordan MTT kan fortolkes i en bred vifte af såkaldte “structured” kategorier, og vi benytter selvsamme til at studere udtrykskraften af typeteorien og flere udvidelser heraf.

Slutteligt udforsker vi adskillige konkrete anvendelser af MTT i kontekst af “guarded” typeteori og guarded denatotionel semantik. Vi fremsætter et yderst brugbart sprog for guarded rekursion og udforsker dets modeller og metateori. Vi viser et forholdsvist skarpt resultat, der afgrænser omfanget af, hvorvidt klassisk guarded rekursion kan tilføjes i en vilkårlig typeteori med afgørligt typecheck, og vi fremsætter et system, der afhjælper denne problematik. Til sidst udfører vi en dybdegående case study ved hjælp af guarded MTT og opnår en fuldkommen syntetisk redegørelse af programlogikken Iris, der viser logikkens tilstrækkelighed på fuldkommen intern vis.

Publications

Below is a record of the articles published during my PhD.

1. Iron: Managing Obligations in Higher-Order Concurrent Separation Logic [Biz+19]
2. Cubical Syntax for Reflection-Free Extensional Equality [SAG19]
3. Implementing a Modal Dependent Type Theory [GSB19a]
4. **Multimodal Dependent Type Theory** [Gra+20a] (Chapters 6 and 7)
5. Transfinite Iris: Resolving an Existential Dilemma of Step-Indexed Separation Logic [Spi+21]
6. **Multimodal Dependent Type Theory** [Gra+21] (Chapters 6, 7 and 9)
7. **Modalities and Parametric Adjoints** [Gra+22] (Chapter 5)
8. A Cubical Language for Bishop Sets [SAG22]
9. **Normalization for multimodal dependent type theory** [Gra22] (Chapter 8)
10. **A stratified approach to Löb induction** [GB22] (Chapter 9)
11. A flexible multimodal proof assistant [SGB23a]
12. Under Lock and Key: A Proof System for a Multimodal Logic [KG23]

The following articles were submitted during my PhD and are presently under review.

13. Strict universes for Grothendieck topoi [GSS22]
14. Unifying cubical and multimodal type theory [Aag+22]
15. Denotational semantics of general store and polymorphism [SGB23b]
16. **Normalization for multimodal type theory** [Gra23]
17. Controlling unfolding in type theory [Gra+23]
18. Free theorems from univalent reference types [SGB23c]

Chapter 8 is a lightly revised version of Gratzer [Gra23] and Section 9.6.4 contains some significantly revised text from Gratzer and Birkedal [GB22]. Both of these articles were written either primarily or solely by the author of this thesis. The text of all other chapters is original though many results appeared in some form in the bolded articles.

To my sister, Anna, for love and inspiration

Acknowledgments

First and foremost, I am indebted to Lars Birkedal for guidance, advice, and encouragement. Though the results of five years of mentorship are difficult to sum up, this thesis is my best attempt at the task. If my conversations with Lars have taught me anything, it is the value of a well-motivated example. This work includes far fewer of these than I would like, but it would have included none at all without Lars's gentle encouragement.

I have found nothing so important to good research as talented and insightful collaborators, and I have had more than my fair share of them. My sincere thanks go to Frederik Lerbjerg Aagaard, Carlo Angiuli, Aleš Bizjak, Evan Cavallo, Thierry Coquand, Derek Dreyer, Favonia (Kuen-Bang Hou), Lennard Gäher, Adrien Guatto, Robbert Krebbers, Magnus Baunsgaard Kristensen, Anders Mörtberg, Andreas Nuyts, Alex Kavvos, Christian Sattler, Michael Shulman, Simon Spies, Bas Spitters, Philipp Stassen, Jonathan Sterling, and Joseph Tassarotti. Of this exceptional list, I would like to extend particular thanks to Carlo and Jon whose advice and critique have been instrumental to my growth and to Alex, who in many ways has served as a second adviser to me. I am grateful to Carlo Angiuli, Lars Birkedal, Simon Gregersen, Jonas Kastberg Hinrichsen, Jonathan Sterling, and Simon Friis Vindum for their comments on this thesis.

My studies have benefited from the incredible Logic and Semantics and Programming Languages groups at Aarhus University. My thanks for five years worth of thought-provoking and fun conversations over coffee. I am also grateful to Thierry Coquand and to the Logic and Types group for hosting me at Chalmers during the Spring of 2022.

On a more personal note, I am truly fortunate to have a mother who has encouraged and supported my curiosity and passion throughout the years. I hope this thesis will finally clear up what, exactly, I have been doing all this time. I am similarly blessed with a wonderful sister, to whom this thesis is dedicated. I hope it will live up to your standard of writing, but I doubt it.

Lastly, my time in Aarhus has included something outside of research and this is due to a circle of wonderful people. I'd express my gratitude to Sarafina McPherson Kimø for love and support alongside witty and biting commentary. Those who know me know I tend to ramble about mathematics and she has certainly suffered from this the most. I would also like to acknowledge Alfred, Philipp Haselwarter, Irina Kakhoun, Klara Nymo Møller, Sydney Paugh, Anna Buhl Pedersen, Thor Bagge, Simon Gregersen, Sabine Oechsner, Marit Edna Ohlenbusch, Søren Eller Thomsen, and many others. Thank you all, truly. Without you, I would have been far more productive and far less happy.

Daniel
Aarhus, Denmark

Contents

1 Introduction

> The time really seems to be ripe for a fruitful development of modal logic, if only we take care to purify and simplify the foundations.
>
> Dana Scott
> *Advice on modal logic*

While the contents of this thesis are, in all honesty, technical, the goals and problems which motivate this work are familiar to any type theorist. In this chapter, we endeavor to give a flavor for the problems that motivate this thesis and an idea of the technical solutions which provide its substance.

At a core level, we are interested in type theory and like nearly all works on type theory we are therefore motivated by two distinct but inextricable goals:

1. We want a simpler way to write computer programs *while also being certain those programs perform as intended.*

2. We wish to make the process of doing ordinary mathematics more straightforward.

A priori, nothing links these two goals and it is a remarkable fact that problems can be attacked through the same tool: type theory [Mar82]. This coincidence has sparked the rich and productive interplay between constructive mathematics and programming. Our goal is to make type theory into a sharper tool for specific applications both in mathematics and computer science. In particular, we will study extensions of type theory by one or more *modalities*—an operation assigning types to types—to make it better suited for both of these purposes. Such extensions can lead to particularly short and elegant constructions, but extending type theory by a modality is a difficult process.

We aim to make it easier to construct modal extensions of type theory in order to make it possible for ordinary users to benefit from the convenience of such specialized type theories.

What is a modality? We have just described a modality as a unary connective which assigns types to types and this is a fair first definition. What really distinguishes a modality from other connectives of type theory, however, is its poor behavior. Ordinary connectives in type theory satisfy various properties and modalities are distinguished by their failure to satisfy one or more of these properties (e.g., stability under substitution).

This *negative* characterization of modalities makes the study of type theories with modalities (*modal type theories*) a very broad tent. It also means, however, that it is utterly hopeless to attempt to develop any sort of general framework for modal type theories. Accordingly, every work on modal type theory must begin by carving out the class of modalities (or, more often, the single modality) that it is concerned with to develop their particular extension of type theory. The extension will hinge on some of the particular properties of their modalities and this has led to a proliferation of different and incomparable modal type theories. This is to say nothing of the rich interaction between modalities and different substructural type theories!

As a first step in our goal of providing a general modal type theory that subsumes some of the more specialized theories, we take a step back to survey some of the applications of modalities. Our goal is to cultivate some intuition for what reasonable base assumptions a putative general modal type theory might make on all modalities.

1.1 Modalities in computer science

Modalities are slightly easier to motivate from the point of view of computer science, so we begin there. We consider three distinct applications: synchronous programming, distributed computation, and staged programming. We will argue that all three applications benefit from modalities. Our goal in this section is not to describe precise systems, but rather to give a flavor for what problems modalities have been applied to in computer science.

1.1.1 Synchronous programming

Consider the task of writing a program for some reactive system i.e., a program that must read from a variety of inputs and produce output continuously. A concrete example of such a program might be the speed control feature on an automobile; the program must read the vehicle's present speed and use this to manage the acceleration of the vehicle.

One useful way to capture this process within a program is to design programs that operate on streams of values. On one hand, a program accepts the stream of data produced by the sensors monitoring the vehicle's speed and, on the other, it produces a stream of values to adjust the fuel consumption and engine output. In an overly idealized situation, one value read from the sensor could result in one value outputted to the engine but this is overly naïve. Perhaps the sensor is noisy and its values must be averaged over an extended period to ensure accuracy or perhaps the engine responds slowly and this must be factored into the algorithm. Regardless, reactive programs of this form have several invariants which are imperative to their correct functioning:

- Responsiveness: even if the ratio is not precisely 1:1, any reactive system must produce output at some rate determined by the rate of input and must not spin indefinitely attempting to compute a single value.
- Constant-space: most reactive systems will run for protracted periods and therefore cannot afford to store space related to all previously received input values. Ideally, the output at any given stage can depend on only a constant amount of prior input.

This is in addition to invariants we insist all programs satisfy (safety, correctness, etc.).

A type system for reactive programs aims to force the first two invariants mentioned above to hold *by construction* for any well-typed reactive program. A recent approach that has proven successful is to provide modifiers on types—modalities— which indicate the rate at which they are consumed and produced [BGM19; BGM21; Gua18; Jef12; Jef14; Jel13; KB11; Kri13; KBH12]

For instance, it is generally unsound for a stream $M : A$ to depend on itself; this might violate the reactivity guarantee and cause the program to spin forever attempting to produce the next value of M. However, we may allow M to refer to itself only under a particular modality, ensuring that the present value of M depends not on itself, but on prior values in the stream. By controlling these modalities more carefully, we may further tune the dependence to allow for dependence on only the immediately preceding value, etc.

The result is that we are able to express complex and recursive reactive programs without compromising the reactivity guarantee. Further considerations on what operations are allowed by modalities can even ensure the constant-space invariant. For instance, by forbidding streams from being arbitrarily delayed, it is possible to ensure that memory cannot be accumulated indefinitely [BGM19; BGM21].

Many of these guarantees are provided by existing synchronous languages such as Lustre [Hal+91]. This modality-centric approach, however, ensures that these systems can be scaled up to accommodate higher-order programs, including complex nested data types, etc. without undue complexity or severe restrictions on expressible programs.

1.1.2 Distributed computation

Let us suppose we are designing a programming language for distributed computing. We will obviously wish to extend our language with primitives for sending and receiving data as inter-node communication is a fundamental component of distributed computation. What interface ought we to provide for these operations? Traditional interfaces provide varying levels of abstraction on top of the API provided by sockets; some might enforce typed sockets, others might abstract over local intra- and inter-node communication between threads, etc.

One strikingly higher-level API was proposed by Tom Murphy VII and collaborators [Mur08; MCH05; MCH07; Mur+04]. Essentially, code becomes wholly unaware of sockets and instead interacts with various nodes through a modal abstraction. At a high level, the typing judgment is $M : A$ replaced with a new judgment $M : A \mathbin{@} w$ signifying that M is a computation of type A taking place at a node designated w. Terms and types at world w can mention those from world w' through a *modality* $A \,\mathsf{at}\, w'$ whose elements, in effect, correspond to values of type A which could be computed at world w'. Various operations on these modalities abstract over the classical operations of sending and receiving between nodes. Moreover, they do so in a fully-typed and safe manner.

For instance, in order to implement receiving, a new singleton type $\mathsf{Addr}\, w$ is introduced whose (necessarily unique) value is the address needed to send messages to w. Given elements of $\mathsf{Addr}\, w'$ and $A \,\mathsf{at}\, w'$ at some world w, one can perform a get operation which produces a normal value of type A. Operationally, this corresponds to sending a request to a node and receiving a corresponding response, but none of these details are manifested within the code. From the point of view of the user, this behaves like the ordinary manipulation of certain modalities.

Further enhancement and refinements of the system are used to facilitate richer forms of distributed computation and communication while simultaneously enforcing various properties. We refer the reader to the cited work for a full account of the details and provide only one illustrative example. Note that allowing arbitrary types to be transported in the manner described is unlikely to be sound; not every value can be sensibly transported from one node to another (e.g. file pointers or references). Murphy [Mur08] shows that these limitations can be abstracted over in a simple and conceptual manner by ensuring that *constructing* a value of $A \,\mathsf{at}\, w$ is only possible when A is *mobile* and excluding node-local types from mobility.

Such a high-level interface may not be appropriate for all distributed systems, but the modal discipline given by Murphy [Mur08] gives a remarkably coherent and conceptual abstraction on top of lower-level primitives.

1.1.3 Staged programming and partial evaluation

We conclude this brief exploration of modalities within computer science with one of the earliest and longest-running areas of application: staged programming. At a high level, staged programming is concerned with controlling when code executes to ensure faster and more predictable evaluation. The impact of staging considerations is hard to overstate; Haskell introduced the (in)famous monomorphism restriction more-or-less to ensure better staging of evaluation.

Staging and the control of evaluation is one of the earliest and most widely studied applications of modalities in computer science [DP01; Jan+22; KI19; Kov22; NP05; NPP08]. The basic idea is illuminated by Davies and Pfenning [DP01]. Essentially, we wish to give those programs which are not yet evaluated a different type than those which are presently being computed. This discipline ensures that no unevaluated code depends incorrectly on code being evaluated at the current stage.

To separate the two, Davies and Pfenning [DP01] introduce a modality $\square$ and ensure that $\square A$ classifies programs awaiting evaluation. In other words, $\square A$ classifies *programs which compute an* A rather than simply classifying elements of A directly. Notice that there is already some benefit to this idea; by rendering this as an ordinary type, a user can write programs that accept and freely manipulate code.

The relation between $\square A$ and A is subtle. Given $\square A$—a piece of syntax which builds an element of A—we ought to be able to produce A. This precisely corresponds to evaluation. The reverse direction should not be generally possible; we should not be able to extract the code of a program currently running. The central thesis of Davies and Pfenning [DP01] is that there is a logical thread connecting all of the conditions that $\square A$ ought to satisfy: it should be the case that $\square$ satisfies the axioms of the S4 modality familiar to logicians.

One consequence of this viewpoint is an elegant calculus and Davies and Pfenning [DP01] show that it behaves as expected. In particular, they endow their calculus with an operational semantics that realizes the promised staging. Elements of $\square A$ are treated as “stuck code” and the introduction rule for $\square A$ does not have a congruence rule. They prove that this reduction relation satisfies the standard criterion used to judge staged evaluation [Pal93]. For instance, they show that code is only computed from code and that their operational semantics evaluates non-code to a value.

Remarkably, aside from the operational semantics nothing in the calculus intrinsically refers to staging. As a result, a programmer does not need to be overly concerned with the details of staging; the system does not contain any surprising restrictions which might confuse a user as all restrictions flow from the statement that $\Box$ is an S4 modality.

The robustness of this idea has allowed many subsequent researchers to extend the work by Davies and Pfenning [DP01] to complex languages and with additional powerful features. Not only by integrating staging into more realistic languages but by allowing the user to scrutinize elements of $\Box A$ as code to provide a logical account of metaprogramming [Jan+22].

1.2 Modalities within mathematics

While modalities arise frequently in computer science, it is harder to extract precise assumptions on modalities from these examples. There is so much flexibility in how one chooses to capture a computational situation within type theory that e.g., crisp rules for synchronous type systems are far from immediate. In some instances, e.g., with staged programming, the application is also subject to change so that both endpoints are unfixed. In order to find some more concrete rules we turn to applications of modalities within mathematics where the situation is more rigid.

Before diving into our examples, however, we must make a broader observation about how type theory is used to study mathematics. In particular, we must stress the difference between approaching mathematics *analytically* as opposed to *synthetically*. The analytical approach is more familiar. We use types as basic building blocks to build up the familiar objects of mathematics (groups, rings, spaces, sheaves, etc.) more-or-less as is done classically. In the synthetic approach, by contrast, we regard types as having additional intrinsic structure and add axioms to the theory to enforce this.

The synthetic approach can often lead to extremely terse and elegant proofs when a suitable axiomatization can be found. For examples of synthetic mathematics within type theory, consider the following synthetic versions of homotopy theory [Uni13], differential geometry [Shu18], algebraic geometry [CCH23], and guarded domain theory [Bir+12]. The formula for carrying this out in type theory is to first identify a good category $\mathcal{E}$ which contains the objects we wish to study as at least as a full subcategory, then to interpret type theory in $\mathcal{E}$. One then adds axioms and reasoning principles specific to the category to type theory, forming it into an effective internal language for $\mathcal{E}$.

Modalities enter the picture when we encounter a feature of $\mathcal{E}$ which cannot be internalized merely by postulating a new term or type. Most commonly, this occurs because we wish to include an operator which cannot be sensibly included with arbitrary contexts. For instance, an operation which applies only on global points or which cannot be made to be invariant under pullback. Let us introduce a few examples to illustrate this phenomenon as it arises in practice.

1.2.1 Synthetic guarded domain theory

Guarded domain theory [Bir+12] is an adaptation of domain theory designed to solve certain “domain equations” which arise in e.g., the semantics of higher-order references.

Essentially, rather than working with sets equipped with a partial order, each guarded domain is instrumented with a structured notion of *nearness* or *indistinguishability* and morphisms between guarded domains must preserve nearness. Rather than taking fixed points of continuous functions, one is able to form (unique!) fixed points of maps between guarded domains which draw points closer together.

There have been many concrete realizations of guarded domain theory (in complete bisected bounded ultrametric spaces, complete ordered families of equivalences, sheaves on well-founded ordinal, etc.). One particularly appealing aspect of the theory is the comparatively good behavior of synthetic guarded domain theory when compared with synthetic domain theory.

Roughly, synthetic guarded domain theory takes place inside a topos $\mathcal{E}$ equipped with the following structure:

1. a left exact endofunctor $\blacktriangleright : \mathcal{E} \longrightarrow \mathcal{E}$ (pronounced *later*),
2. a point $\mathsf{next} : \mathsf{id} \longrightarrow \blacktriangleright$,
3. and a collection of morphisms $\mathsf{loeb}_A : A^{\blacktriangleright A} \longrightarrow A$ along with an identification between $\mathsf{loeb}_A \circ f$ and $\epsilon \circ \langle f, \mathsf{next} \circ \mathsf{loeb}_A \circ f \rangle$.

Intuitively, each object of $\mathcal{E}$ is equipped with an intrinsic notion of nearness respected by all morphisms and $\blacktriangleright$ preserves the global structure of an object while drawing the points closer together. The ability to take fixed points of contractive functions is then encoded by the family of constants loeb.

Example 1.2.1. *The canonical model of synthetic guarded domain theory is given by* $\mathbf{PSh}(\omega)$*, the topos of trees. In this model,* $\blacktriangleright$ *is realized as the right adjoint to precomposition by* $-+1$*. In other words, it sends a presheaf* X *to a presheaf* X' *such that* $X'(0) = \mathbf{1}$ *and* $X'(n+1) = X(n)$*.*

Remark 1.2.2. We often ensure that a model of synthetic guarded domain theory is non-trivial by requiring the global points of $\blacktriangleright X$ to be canonically isomorphic to those of X. More precisely, we ask that next induces an isomorphism $\hom(\mathbf{1}, X) \longrightarrow \hom(\mathbf{1}, \blacktriangleright X)$. This rules out, for instance, the degenerate model found in $\mathbf{PSh}([n])$ or similar. ◇

One appealing aspect of this formulation of synthetic guarded type theory is how well it can be captured by the internal type theory of $\mathcal{E}$. Indeed, one can realize $\blacktriangleright$ as a certain type constructor and add loeb directly as an axiom and carry out substantial case studies in guarded denotational semantics [Bir+12; PMB15].

This internalization of synthetic guarded domain theory is inadequate, however, for capturing certain phenomena. Roughly, the internal language allows us to describe only local properties and is inadequate for global observations. The most frequently cited example within the context of guarded denotational semantics is *termination*. Famously, when working analytically in guarded domain theory, occurrences of the $\blacktriangleright$ modality correlate with steps and one must take advantage of the fact that $\hom(\mathbf{1}, \blacktriangleright X) \cong \hom(\mathbf{1}, X)$ to “see past” these $\blacktriangleright$s to determine if a program does terminate.

The occurrence of $\hom(\mathbf{1}, -)$ in this process suggests a possible solution: we could extend the language of synthetic domain theory to include a type constructor which roughly internalizes $\hom(\mathbf{1}, -)$. More precisely, if we assume $\mathcal{E}$ is cocomplete, there is a canonical functor $\Delta : \mathbf{Set} \longrightarrow \mathcal{E}$ sending a set S to the colimit $\coprod_{s \in S} \mathbf{1}$. We can then hope to internalize $\Box = \Delta \circ \hom(\mathbf{1}, -)$ and then postulate an isomorphism $\Box \circ \blacktriangleright \cong \Box$.

Unfortunately, $\Box$ simply does not internalize as well as $\blacktriangleright$; it cannot be modeled directly as a type constructor because the following rule is not semantically valid:

$$\frac{\Gamma \vdash A\,\mathsf{type}}{\Gamma \vdash \Box A\,\mathsf{type}}$$

Indeed, if only $\Gamma \vdash A\,\mathsf{type}$ it is not at all clear what the global points of A over Γ should be; what if, for instance, Γ has non-trivial local points but is globally empty? There are a number of ad-hoc fixes one can propose, but fundamentally there can be no operation $\Box : \mathcal{U} \to \mathcal{U}$ which correctly internalizes the desired functor [Shu18] and there is no sensible way to bolt in $\Box$ in a manner which is coherent with respect to substitution.

Summarizing, synthetic guarded domain theory internalizes remarkably well, with the glaring omission of the ability to speak of global behavior through $\Box$. In fact, we shall see that both $\blacktriangleright$ and $\Box$ are instances of what we will eventually call modalities and it is possible to alter type theory to include both. While it is possible to directly postulate $\blacktriangleright$, this will certainly disrupt canonicity while also making it extremely difficult to correctly control the interaction of $\Box$ and $\blacktriangleright$. We will therefore opt to organize both $\Box$ and $\blacktriangleright$ through the same mechanism and, consequently, we can derive $\Box\blacktriangleright A \simeq \Box A$ rather than needing to take it as an axiom. Even better, there are a myriad of other modalities which could permissibly be utilized in the study of guarded domain theory [Gua18] and we shall eventually see that each of these constitutes a modality.

1.2.2 Cohesive homotopy type theory

Despite the fact that synthetic guarded domain theory is essentially purely mathematical, its motivations are still drawn from computer science (the denotational semantics of programming languages). We now turn to an example motivated by more familiar mathematics: a synthetic account of topology. We will follow Shulman [Shu18] who in turn draws on many others [Gro+17; Law07; SS20; Sch13; SS12].

As mentioned above, homotopy type theory (HoTT) has proven to be a widely-studied synthetic account of homotopy theory. While HoTT includes many familiar objects from geometry, they can behave quite differently as HoTT captures only their underlying ∞-groupoid. For instance, one can define a higher inductive type (HIT) $\mathbb{I}$ intended to represent the closed unit interval, but $\mathbb{I}^n \simeq \mathbb{I} \simeq \mathbf{1}$. These equivalences are perfectly correct when regarding $\mathbb{I}$ as an ∞-groupoid, but quite inadequate for topology where all three of the aforementioned spaces behave quite differently.

The goal of cohesive HoTT is to allow one to describe the interval as a HIT and as a space within the same framework. The inspiration for this comes from Lawvere [Law07]. If we completely disabuse ourselves of the identification of ∞-groupoids with spaces and instead treat the former more like sets, we are naturally led to wonder if (smooth) topological spaces can ever be organized into an ∞-topos—a category which models HoTT—and, if they can, what properties of that ∞-topos can be axiomatized to reason about spaces. This question is remarkably similar to the very classical question of embedding smooth manifolds into an (ordinary) topos and several solutions exist.

Schreiber [Sch13] proposes one method which generalizes well to the ∞-categorical setting: taking (∞-)sheaves $\mathbf{Sh}(\mathbf{Cart})$ on a site that contains all the information of the smooth category (the subcategory of **Smth** spanned by spaces of the form $\mathbb{R}^n$).[1] Certain well-behaved colimits of representables in this category fully faithfully capture smooth manifolds. Even better, one can recover much of the reasoning carried out in $\mathbf{Sh}(\mathbf{Cart})$ through the string of four adjoints connecting it to the ∞-category of ∞-groupoids $\mathcal{S}$:

$$\begin{tikzcd}
\mathbf{Sh}(\mathbf{Cart}) \arrow[d, bend left=15] \arrow[d, bend left=60] \arrow[d, phantom, shift right=8, "\dashv"] \arrow[d, phantom, "\dashv"] \arrow[d, phantom, shift left=8, "\dashv"] \\
\mathcal{S} \arrow[u, bend left=60] \arrow[u, bend left=15]
\end{tikzcd}$$

This string is often notated $p_! \dashv p^* \dashv p_* \dashv p^!$ and this general situation is often summarized by stating that $\mathbf{Sh}(\mathbf{Cart})$ is a *cohesive ∞-topos*. By asking for specific properties of this adjoint chain, one can recover much of the flavor of $\mathbf{Sh}(\mathbf{Cart})$ purely synthetically. We refer the reader to Schreiber [Sch13] or Shulman [Shu18] for a discussion of the particular axioms to set up real cohesive homotopy type theory.

Now that we have a putative synthetic theory of topology, it remains to somehow internalize this into (homotopy) type theory. Shulman [Shu19] has shown how to interpret HoTT into $\mathbf{Sh}(\mathbf{Cart})$, but of course that gets us no closer to internalizing p which ranges between two distinct categories. Shulman [Shu18]—following Schreiber [Sch13]—solves this by passing from considering p to the monads and comonads it induces: $\int = p_! \circ p^*$, $\flat = p_* \circ p^*$, and $\sharp = p_* \circ p^!$.

Just as was done with $\blacktriangleright$, it turns out that $\sharp$ can be added to type theory rather directly and the monadic operations on it can then be postulated. The other two $\flat$ and $\int$ are more complex; fortunately, the axioms for cohesion will force $\int$ into a more tractable form, but $\flat$ is just as resistant to directly adding to type theory as $\Box$ proved to be. In fact, they share many structural properties as both are lex idempotent comonads.

Just as before, we cannot hope to realize $\flat$ as an operation on the universe. Unlike before, $\sharp$ offers one potential avenue for escape (we could realize $\flat : \sharp\mathcal{U} \to \sharp\mathcal{U}$). This formulation is unsatisfactory however; it leaves us the task of axiomatizing and proving a great deal of structure on $\sharp\mathcal{U}$ and it does not allow us to ever obtain $\flat A : \mathcal{U}$ even in situations where this would be semantically valid. For this reason, Shulman [Shu18] modifies the structure of the homotopy type theory to realize both $\flat$ and $\sharp$ uniformly as a pair of adjoint modal operators. A major pay-off of this approach is the amount that can be derived from the relatively little added structure. A secondary pay-off is the large suite of definitional equalities available which simplifies several of these proofs.

A further wrinkle presents itself in this example. Cohesion properly involves two distinct categories ($\mathcal{S}$, $\mathbf{Sh}(\mathbf{Cart})$) and it is somewhat unnatural to limit ourselves to just working with $\mathbf{Sh}(\mathbf{Cart})$. In this instance, we were able to limit ourselves to the monads and comonads induced by an adjunction without losing information, but ideally modal type theory would allow for modalities linking together distinct internal languages.

[1] We refer the reader to Sati and Schreiber [SS20] for a more direct account.

1.2.3 Relative realizability

Our final example from mathematics is drawn from Birkedal [Bir00] and concerns realizability theory. Briefly, ordinary realizability theory enables one to take a description of a language (a partial combinatory algebra or PCA) and convert it into an elementary topoi which reflects a certain logic of computable functions relative to the supplied PCA. These elementary topoi can be useful tools both for the study of type theory and for the analysis of the underlying PCA.

Relative realizability is concerned with the study of relative PCAs $(A, A_\sharp)$ which consist of an ordinary PCA A along with a sub-PCA $A_\sharp$ closed under multiplication and containing S and K. Intuitively, $A_\sharp$ consists of the computable objects while A may include additional infinite and non-computable objects. Having both objects exist within the same PCA allows us to describe computable processes which act on potentially non-computable data. For instance, we might extend the ordinary λ-calculus with infinite choice sequences so $A_\sharp$ consists of ordinary closed λ-terms and A is extended with these infinite sequences. By mixing both together, we can state and prove theorems such as "λ-terms act continuously on choice sequences".

Of course, both A and $A_\sharp$ are ordinary PCAs when viewed separately and both therefore induce realizability topoi $\mathbf{RT}(A)$ and $\mathbf{RT}(A_\sharp)$. Given the close relationship between $A_\sharp$ and A, we may define a realizability topos $\mathbf{RT}(A, A_\sharp)$ that mixes the two. In both $\mathbf{RT}(A)$ and $\mathbf{RT}(A, A_\sharp)$ objects are given by objects realized by equivalences classes of objects from A while in $\mathbf{RT}(A_\sharp)$ and $\mathbf{RT}(A, A_\sharp)$ morphisms are given by functional relations realized by objects from $A_\sharp$. This means that $\mathbf{RT}(A, A_\sharp)$ is exactly the setting required to make statements about computable algorithms on non-computable data.

Given the close relationship between A and $A_\sharp$, it is natural to wonder what the relationship is between these three topoi. Birkedal [Bir00] shows that the resulting topoi are connected by a rich network of adjoints and that, in particular, $\mathbf{RT}(A_\sharp)$ and $\mathbf{RT}(A, A_\sharp)$ are linked by a localic geometric morphism which gives rise to a structure similar to that encountered in Section 1.2.2. *Op. cit.* then shows that this structure can be incorporated into the internal logic of $\mathbf{RT}(A, A_\sharp)$ through a pair of adjoint modalities, just as in Section 1.2.2.

Just as in cohesive HoTT, presenting these modalities within the logic is challenging. In Birkedal [Bir00], the author obtains a workable logic but it cannot be scaled up to dependent types; the rules governing the modalities simply insist that certain inferences be valid only in an empty context or similar which would cause serious problems within type theory.[2] These modalities in internal logic remain very useful. With them, for instance, one may state and prove propositions governing computation on non-computational objects, but without a dependent type theory this is necessarily stratified into two distinct languages.

1.3 Modal type theory

Thus far we have explored several applications of modal type theories and programming languages, but we have come no closer to isolating what, exactly a modality should be. This situation is indicative of a truism in the study of modal type theory: practitioners

[2] I have it on good authority that Birkedal is aware of this deficiency.

suffer from an abundance of putative semantic models and a lack of good syntax. The goal of isolating a particular class of modalities is to cut away enough potential models to allow for a good syntax while ensuring that the motivating examples remain.

Our starting point is the observation that modalities described in Section 1.2 can be treated as functors on a model of type theory. Now, often such functors descend to each slice and thereby can be internalized directly as a map on the universe $\bigcirc : \mathcal{U} \to \mathcal{U}$. We refer to these operations as *fibered modalities* and they constitute an important special class of modalities. The principal appeal of fibered modalities is that they require no alteration to the structure of the type theory. Consequently, the study of fibered modalities is presently far more advanced than the study of general modalities and there is a well-developed theory of such modalities already available [RSS20].

Unfortunately, as Section 1.2 shows, this situation is not universal and not every functor of interest can be captured by a fibered modality. We are therefore interested in studying a class of modalities which does not require fiberedness. The contribution of this thesis is to defend the following pair of hypotheses:

Hypothesis 1. Weak dependent right adjoints are a good class of non-fibered modalities.

Hypothesis 2. Weak dependent right adjoints give rise to a usable general syntax.

Defining weak dependent right adjoints (wDRAs) and justifying these hypotheses occupies the bulk of this thesis. Both are fundamentally empirical claims and cannot be either proven or refuted mathematically. Moreover, it seems unlikely that there will ever be an unequivocal "best" solution to the question these claims aim to answer. Indeed, we will encounter examples of modalities in the literature which are not wDRAs and we shall see special cases where the general syntax could be refined.

What we can and will do in this thesis is to develop evidence for both our hypotheses by the properties of the resulting syntax, exploring the various relevant categories of models, and examining case studies and examples. If we cannot convince the reader of our claim that wDRAs form a good level of abstraction, we will surely provide them with ample evidence from which to draw this conclusion.

1.3.1 From functors to weak dependent right adjoints

Our assumption that F is a functor already provides some leverage even without the assumption that F descends to slices. For a start, it ensures that modalities come with an action on contexts and substitutions ($\Box\Gamma$ and $\Box\gamma$, respectively). Without anything further, however, there is no way to parlay this into an action on types and terms.

In order to accomplish this, we must contemplate how types are interpreted in models like those described in Section 1.2. Roughly, a type $\Gamma \vdash A$ is interpreted as a family over $[\![\Gamma]\!]$; a well-behaved morphism or *display map* $[\![\Gamma.A]\!] \longrightarrow [\![\Gamma]\!]$.[3] To lift the action of our modality F to types, we require that F sends display maps to display maps (types to types) and take $[\![\Box A]\!] = F\big([\![\Gamma.A]\!] \longrightarrow [\![\Gamma]\!]\big)$. We emphasize that this does *not* imply that $\Box A$ will inhabit the same context as A. Indeed, examining the interpretation again, we obtain the following formation rule for modal types:

$$\frac{\Gamma \vdash A}{\Box\Gamma \vdash \Box A}$$

[3] Properly speaking, one must employ a coherence theorem to organize display maps into a model of type theory. We ignore these details in this introduction.

This definition also gives a canonical action of $\Box$ on terms. In a model, a term $\Gamma \vdash M : A$ corresponds to a section of the canonical map $[\![\Gamma.A]\!] \longrightarrow [\![\Gamma]\!]$. As any functor preserve sections, our identification of $\Box A$ with $F([\![\Gamma.A]\!]) \longrightarrow F([\![\Gamma]\!])$ ensures that each term $\Gamma \vdash M : A$ induces a term $\Box\Gamma \vdash \mathsf{mod}(M) : \Box A$. We render this as an inference rule:

$$\frac{\Gamma \vdash M : A}{\Box\Gamma \vdash \mathsf{mod}(M) : \Box A}$$

More generally, our chosen interpretation of $\Box A$ ensures that (by definition) $\Box\Gamma.\Box A \cong \Box(\Gamma.A)$. This can also be used to obtain something akin to an elimination rule for $\Box A$.

What more can be said about $\Box$? Precious little without further assumptions. For instance, we have essentially no control over $(\Box A)[\gamma]$. Semantically, this corresponds to a pullback of a morphism $F([\![\Gamma.A]\!]) \longrightarrow F([\![\Gamma]\!])$ but with our present set of assumptions all we can conclude is that this is a display family.

In general, connectives in type theory are stable under substitution e.g., $(A \times B)[\gamma] = A[\gamma] \times B[\gamma]$. The above discussion is our first encounter with a phenomenon that will occur over and over again in our discussion on modalities: modalities do not generally enjoy stability under substitution.

Examining the formation rule given above for $\Box A$ it is clear that $(\Box A)[\gamma]$ can never be of the form $\Box B$ unless the domain of γ is also of the form $\Box\Delta$. As it stands, we do not even obtain a substitution lemma for $\Box$ in this special case but we can obtain one after imposing a further requirement on F. We will assume that F preserves pullbacks of display maps. This assumption enables us to validate the following in our model:

$$\frac{\Delta \vdash \gamma : \Gamma \qquad \Gamma \vdash A\,\mathsf{type}}{\Box\Delta \vdash (\Box A)[\Box\gamma] = \Box(A[\gamma])\,\mathsf{type}}$$

In a typical semantic model, all morphisms will be display families and so these requirements boil down to the assumption that a modality is a pullback-preserving functor.

Remark 1.3.1. Even this broadest assumption rules out certain examples referred to as modalities in the literature (e.g., some formulations of contextual types or the leftmost adjoint in a cohesive situation), but it preserves the examples listed above and the others we consider in this thesis. ◇

The assumption that modalities are realized by pullback-preserving functors is fairly standard, but the syntax which results from it is far from optimal. The resulting calculus is often termed a calculus of *delayed substitutions* because $\Box A$ only enjoys a substitution lemma for a specific class of substitutions. The remaining substitutions are then "delayed" on $\Box A$ and cannot be resolved. The resulting calculus must be equipped with complex rules for manipulating these substitutions or require the user to reason about substitutions directly [Bd00], neither of which is optimal. To obtain a more optimal syntax, each modal type theory in the literature imposes some further requirements on F or the category of contexts in order to improve the situation. We could either

1. alter contexts and substitutions to force a better substitution lemma,

2. or assume F satisfies additional properties to obtain a better substitution lemma.

The first approach is exemplified by the so-called dual-context type theories [dR15; Kav17; PD01; Shu18; Zwa19] and the second by Fitch-/Kripke-style modal type theories [BGM17; Bir+20; Clo18; Gra+22; GSB19a; HP23; VRT22]. We shall ultimately see that both approaches are manifestations of weak dependent right adjoints, but we discuss them separately for now.

Dual-context type theories Roughly, one might summarize the issues with $\Box A$ and substitutions with the observation that not every substitution γ is of the form $\Box\gamma_0$. The idea of dual-contexts is to weaken the requirement to ensure that while γ may not literally be of the form $\Box\gamma_0$, there is at least a canonical choice of γ_0. To accomplish this, both contexts and substitutions are split into pairs $\Delta;\Gamma$ and $\delta;\gamma$.[4] Semantically, $\Delta;\Gamma$ represents an object $[\![\Delta;\Gamma]\!]$ equipped with a map to $F([\![\Delta]\!])$ so that, intuitively, variables and terms Δ and δ are treated as though they are under $\Box$.

The reward for making this stratification is a more well-behaved interface for the modality. It is no longer necessary to assume the entire context is of the form $\Box\Gamma$ nor is it necessary to assume a substitution $\gamma = \Box\gamma_0$. Instead, we use the first half of these dual-contexts and substitutions:

$$\frac{\mathbf{1};\Delta \vdash A}{\Delta;\Gamma \vdash \Box A} \qquad (\Box A)[\delta;\gamma] = \Box(A[!;\delta])$$

Returning to our earlier notation with unified context, we can summarize the pay-off of dual contexts. First, by splitting contexts, there is a canonical map $\Gamma \longrightarrow \Box\Delta$ for any given context Γ and we can use this to give a version of the formation rule for $\Box A$ which applies in any context. Second, by ensuring that substitutions also split, the aforementioned choice of $\Box\Delta$ is stable under substitution, and this ensures that $\Box A$ enjoys an unconditional substitution lemma. The remaining rules of the system, particularly the elimination rule for $\Box A$, allow one to move variables back and forth between the two contexts.

Fitch-style type theories Dual-context type theories provide for a canonical choice of map $\Gamma \longrightarrow \Box\Delta$ and Fitch-style type theories extend this further by providing an *initial* map. Essentially, having a canonical choice of map is sufficient for ensuring a substitution lemma, but it has a major downside. It is far from obvious how to scale up this technique to accommodate more than one modality. One is quickly led to not just having two context zones, but a context zone for each possible combination of modalities. All too quickly, this leads to requiring infinitary contexts or similarly complex structures. While this is mathematically possible and useful in certain coherence theorems [Shu23], it is not the foundation for a usable syntax.

The idea is that by ensuring that $\Gamma \longrightarrow \Box\Delta$ has a universal property, its existence can become a property rather than a structure recorded in the context. This should then scale up to multiple modalities simply by asserting more and more properties hold, rather than having to add more and more structure to each context.

[4] That substitutions must be divided along with contexts is often left implicit, but it is vital.

The requirement that there exists a Δ and substitution $\Gamma \longrightarrow \Box\Delta$ initial among such pairs can be rephrased in a more familiar categorical manner: we require that F is a right adjoint upon contexts. Unlike dual-context type theories, this is merely imposing additional requirements on F and therefore more in line with what the story we have seen thus far.

Remark 1.3.2. As with any requirement imposed upon a class of modalities, it is worth examining what examples are lost. In this case, most of the examples presented so far can be adapted with a few notable exceptions. For instance, in the cohesive situations detailed in Sections 1.2.2 and 1.2.3, the leftmost adjoint must be excluded. ⋄

Just as we have internalized the action of F on contexts and substitutions ($\Box\Gamma$, $\Box\gamma$), we will do the same for the left adjoint and denote its action on e.g. Γ by $L(\Gamma)$. Remarkably, with just this additional requirement, we are able to *derive* several seemingly new rules from the existing rules governing $\Box$.

To begin with, the existence of a left adjoint allows us to simplify the formation rule for $\Box$. Just as with dual-context type theories, we are able to allow for an arbitrary context in the conclusion, though this time we use the left adjoint and unit rather than a chosen context and substitution. The rules governing terms of $\Box A$, however, can also be simplified into a pair of "transposition-like" rules. All told we arrive at the following:

$$\frac{L(\Gamma) \vdash A\,\mathsf{type}}{\Gamma \vdash \Box A\,\mathsf{type}} \qquad \frac{L(\Gamma) \vdash M : A}{\Gamma \vdash \mathsf{mod}(M) : \Box A} \qquad \frac{\Gamma \vdash M : \Box A}{L(\Gamma) \vdash \mathsf{unmod}(M) : A}$$

One can take these rules as primitive and ask what structure we then require on the original functor F to validate them. The resulting structure is called a *dependent right adjoint* by Birkedal et al. [Bir+20].

The functoriality of the left adjoint $L(-)$ can be used to give the first two rules a simple substitution principle, but the last rule for $\mathsf{unmod}(-)$ is more difficult. A variety of approaches are possible [Gra+22; GSB19a; HP23] but each solution either does not scale to multiple modalities or requires even more properties on the original F than merely being a right adjoint (an already somewhat stringent requirement).

Weak dependent right adjoints Defining weak dependent right adjoints is a substantial technical undertaking and we will not attempt it yet. We can, however, already provide a flavor of what position they occupy in relation to the two approaches to modalities discussed above.

Essentially, weak dependent right adjoints split the difference between these two approaches. Historically, weak dependent right adjoints were first isolated when attempting to generalize Fitch-style type theories to allow for multiple modalities [Gra+21]. Accordingly, op. cit. weakened the elimination rule—the source of trouble—to a "pattern-matching" rule more reminiscent of the elimination rule from dual-contexts.

In more detail, just like with Fitch-style type theories we assume that each modality determines a left adjoint which acts on contexts and substitutions. We then adopt the specialized formation and introduction rules from Fitch-style type theories as-is. Weak dependent right adjoints deviate in their the elimination rules.

Just as with dual-contexts, we also enrich the context so that variables carry with them an additional piece of information specifying which modalities modify them. In dual-context type theory, this was accomplished by splitting the context into two halves (annotated with the modality or not). Just as was the case in dual-context type theories, the elimination rule is then used to pass between variables with a modal type and variables with a modal annotation.

It turns out that weak dependent right adjoints are then broad enough to encompass both dual-context type theories and Fitch-style type theories. The former becomes a special *coherence* theorem promoting a model of type theory equipped with a lex functor to a model of type theory with a weak dependent right adjoint. The latter becomes a strengthening where there is no difference between a modal-annotated variable and a variable of modal type. In particular, this class of modalities can accommodate the examples discussed above in Section 1.2. Most importantly—unlike dual-context type theories or Fitch-style type theories—weak dependent right adjoints can form the basis for MTT, the first dependent type theory which adapts fluidly to an arbitrary collection of modalities while maintaining a good substitution lemma.

1.4 Contributions

In this thesis, we offer one possible approach to the foundations of modal type theory which is exemplified by MTT, general modal dependent type theory. This modal type theory is the first to simultaneously balance full-spectrum dependent types and arbitrary collections of interacting modalities. To highlight one example, MTT is the first system to be able to seamlessly include both $\square$ and $\blacktriangleright$ from synthetic guarded domain theory as described in Section 1.2.1.

The key ingredient to MTT's flexibility is our proposed foundation of modal type theory (Hypotheses 1 and 2) which identifies modalities with weak dependent right adjoints. We shall see how this definition incorporates many of the best aspects of Fitch-style and dual-context type theories. We systematically develop the categorical theory of weak dependent right adjoints and use this to characterize the relationship between MTT and other type theories appearing in the literature. For instance, we recover dual-context type theories as particular mode of use of MTT. Similarly, Fitch-style type theories can be systematically described as special cases of MTT extended with a further definitional identification. We derive these results through purely semantic methods, thereby avoiding tedious syntactic arguments.

We do more, however than merely providing a common point of generalization between two strands of research. We also show that MTT is an extremely well-behaved and usable system. For instance, we show that the syntax of MTT is remarkably well-behaved and satisfies essentially every metatheorem one could ask for. Among others, we show that MTT is consistent regardless of which modalities are used to instantiate it. More than this, we prove that MTT satisfies canonicity at a similar level of generality. We also develop a novel extension of type-theoretic gluing techniques to prove that MTT satisfies normalization and conclude that its type-checking problem is decidable when the underlying collection of modalities is also decidable.

We also explore the usability of MTT through numerous smaller examples. In particular, we show how the system can be used to internalize various subsystems of IS4 and describe at some length the behavior of adjoint modalities in MTT. In the latter case, we show that many non-trivial results (preservation of colimits by left adjoints, etc.) can be derived purely within MTT without additional assumptions.

We also explore the behavior of MTT through two in-depth case studies. For instance, we provide an in-depth study of guarded recursion within MTT. We do this first in a more classical context: retrofitting MTT with extensional equality and axiomatizing Löb induction. We then show how to recover many of the classical results in a more systematic and streamlined manner. It is then trivial to enrich the resulting guarded type theory with many other modalities which follow naturally from the pair of $\square$ and $\blacktriangleright$. This allows us to give cleaner and more conceptual proofs of e.g., the construction of coinductive types from their guarded counterparts. This last point also relies on the ability to include multiple distinct type theories (modes) which are connected by modalities, a possibility that had not been previously explored in this context. In the same case study, we introduce a novel *stratified* guarded type theory built on MTT which features a static/dynamic distinction allowing a user to program in a system with decidable type-checking, but execute that code in a language with a (guarded) canonicity result. We are able to prove a crisp *no-go* theorem showing that the resulting stratified system occupies a canonical position among strategies to guarded recursion. In both cases, however, we benefit from the uniform metatheorems of MTT which apply directly and ensure e.g., the existence of a well-behaved substitution principle and category of models.

The ability to include multiple interacting modalities also allows us to develop the first fully synthetic account of the Iris concurrent separation logic [Jun+18]. We work inside MTT instantiated to connect two distinct type theories: one for guarded recursion and one where types merely act like sets. Using guarded recursion, we redevelop the Iris program logic entirely synthetically. The recasting of the logic to use a full dependent type theory offers many conceptual simplifications and unifications, enabling us to omit the complex domain equation in favor of a simple guarded fixed-point on the universe. The inclusion of the set mode is crucial, however, for proving the resulting program logic adequate. Being the first modal type theory capable of smoothly including all the necessary modalities, we are able to provide the first synthetic reconstruction of Iris which includes a proof of adequacy.

1.5 Structure of this thesis

This thesis is divided into three parts: preliminaries (Part I), modalities in type theory (Part II), and applications of multimodal type theory (Part III).

Preliminaries The first part of this thesis consists of Chapters 2 to 4 and describes some of the tools and techniques used throughout the remainder of the thesis. Much of this material has appeared before in the literature and references to longer expositions are given throughout. Rather than being read linearly and in-depth, this material is best skimmed and revisited as needed.

Chapter 2 introduces this thesis's perspective on type theory generally. For instance, logical frameworks, the relationship between syntax and semantics, and the collection of type formers we shall include. The chapter also includes a discussion of the various metatheorems of type theory we return to again and again in this thesis. In particular, it includes an explanation of the importance of canonicity, normalization, and the admissibility of substitution.

Chapter 3 introduces the basic notations and definitions from category theory we rely upon. In addition to standard concepts within category theory, a great deal of attention is paid to the machinery and results of categorical type theory. Among other topics, categories with families, natural models, coherence theorems, and constructions of universes are discussed in some depth. As in the discussion above, it is nearly impossible to separate semantic considerations from syntactic ones in the context of modal type theory and so this material is used throughout the thesis.

Chapter 4 is more self-contained than the other two chapters. It discusses *synthetic Tait computability*, a proof technique used to establish metatheorems like canonicity and normalization and works through a simple application proving canonicity for Martin-Löf type theory. This chapter is recommended only for those interested in the applications of synthetic Tait compatibility in Chapter 8 and Chapter 9.

Modalities in type theory The second part defends Hypotheses 1 and 2 and consists of Chapters 5 to 8. It is concerned with developing the syntax of MTT and semantics of weak dependent right adjoints, as well proving several important results such as various recognition principles for models and a normalization result for the type theory.

Chapter 5 extends the discussion of modalities we began in Section 1.3.1. It gives a full account of the syntax and semantics of many of the different varieties of modal type theories proposed in the literature and their relation to each other. In particular, we describe in some depth the relationship between dual-context type theories and Fitch-style type theories and their shared relationship to delayed substitutions.

Chapter 6 draws on the ideas outlined in the prior chapter to introduce the syntax of MTT. The chapter begins with a non-technical and informal introduction to MTT illustrating how it is used on paper. It then turns to the formal syntax of the type theory as well as a variety of possible extensions to the system. It concludes with an in-depth case study covering adjoint modalities to illustrate how the theory is used in practice.

Chapter 7 introduces the semantics of MTT and, with it, the full definition of weak dependent right adjoints. In addition to a natural model definition of a model of MTT, this chapter contains numerous recognition theorems for models of MTT. They are then put to use to prove the independence of a variety of extensions.

Chapter 8 is devoted to the proof of a single result: MTT admits a normalization function. To this end, we develop *multimodal synthetic Tait computability*, an extension of the techniques from Chapter 4 to apply to MTT. We conclude that MTT enjoys decidable type-checking under mild conditions. We further generalize this result to include a variety of extensions of MTT.

Applications of multimodal type theory The final part of this thesis deals with two in-depth case studies of MTT. Chapter 9 is studies MTT as it applies to guarded recursion. In fact, it includes several different views on guarded recursion. The first uses an extensional variant of MTT to reproduce and refine several classical results in guarded type theory, while the second introduces a novel stratified type theory that enjoys decidable type-checking while also allowing for a limited form of guarded canonicity. Finally, Chapter 10 uses a version of extensional guarded MTT to give a fully synthetic reconstruction of the Iris program logic, including the all-important adequacy theorem which has proven to be a sticking point to internalize.

Part I

Preliminaries

2 Type theory

> What is essential is to single out important concepts and to investigate their properties.
>
> Dana Scott
> *Advice on modal logic*

The subject of this thesis is *modal type theory*. In order to properly situate the study of modal type theory, we must first have a firm handle on what characterizes (Martin-Löf) type theory. This is an impossibly broad task for a single chapter, so we set ourselves two slightly more modest goals:

1. Introduce the overarching assumptions and methods we will use throughout this thesis to handle type theories (modal or not). The goal is less to introduce type theory to those without background so much as it is to introduce this thesis's perspective on type theory to those reasonably acquainted with the subject.
2. Discuss what aspects of type theory are essential and must be carried forward as we alter it to accommodate modalities. In particular, the metatheory of ordinary type theory is rich and well-studied. In order to facilitate an informed discussion of the trade-offs involved in modal type theory, we will discuss the function of theorems like canonicity, normalization, decidability of type-checking, and the substitution lemma.

2.1 Logical frameworks and syntax, formal and informal

We begin by noting that we actually use type theory in two somewhat distinct ways throughout this thesis; we are generally studying a particular type theory as a mathematical object but in pursuit of this goal, we sometimes work within type theory and treat it as a concrete and imprecise language.

This dual role is quite common in works on type theory, but unless carefully addressed it raises a whole host of ambiguities. When studying type theory as a mathematical object, we want to be able to relate the syntax of type theory to other objects through a category of models where syntax is initial. At a high level, this essentially states that we ought to have an induction principle for our type theory, enabling us to state and prove properties of the theory by showing that each term and type former preserves the property and that the property respects equality. As one can imagine, without such a uniform induction principle, proofs about type theory can become quite arduous.

Unfortunately, the informal discipline used to write type theory when taken literally cannot be easily connected to the category of models. The informal discipline elides annotations and other information which can make it difficult to explain how, precisely, one should interpret a given term into a model. There is, however, a good reason for these elisions: actually writing out every necessary annotation on every connective would be incredibly verbose and often superfluous.

This has traditionally caused a great deal of angst among those striving to use categorical methods to analyze the syntax of type theory; if we treat concrete and informal syntax as the main object of study, a great deal of difficult and subtle lemmas must be carried out to connect it to syntax qua initial object in the category of models. Sometimes in order to construct such an interpretation we *require* normalization or similar results to hold for the informal syntax. This is a non-starter if, as in this thesis, we intend to use semantic methods to establish normalization.

Moreover, informal syntax is a moving target: there are countless different ways of presenting syntax: more annotations, fewer annotations, restricting application to avoid cuts, etc. This does not even touch upon the conveniences that a modern proof assistant like Coq or Agda might implement to facilitate writing terms in type theory (unification, implicit arguments, type classes, modules, etc.).

In this thesis, we shall make the blanket assumption that when we study formulate and study type theories, we are studying objects generalized algebraic theories [Car78].[1,2] In particular, this means that a type theory is a particular algebraic theory where sorts may depend on terms of other sorts and operations are equipped with suitable dependent types.

This assumption ensures that whenever we specify a type theory, there is automatically a category of models equipped with an initial object which we term *syntax*. Using the general results from Cartmell [Car78] (or Kaposi et al. [KKA19]), one can construct this initial object as as fully-annotated terms or derivations up to definitional equality, but we will have no need to do this throughout this thesis. All the theorems we wish to derive about the syntax of a type theory will be derived through its universal property.

Our usual task then when we define a type theory is *not* to define the syntax and the category of models, but to give a specification in our logical framework and give a more concise and digestible formulation of the output of *turning the crank* to obtain a category of models.

In contrast, whenever we *use* type theory we will use a thoroughly informal discipline which not only elides type annotations but uses implicit arguments, pattern-matching, and other conveniences. This discipline can be translated into the precise and formal syntax specifying various type theories, but we will largely leave this to the reader and not deal with the question of defining a precise elaboration algorithm for the procedure.

Remark 2.1.1. This is similar to the typical discipline adopted by mathematical logicians studying ZFC or similar. The beginning of the paper will state that they are working with some formal system to analyze another formal system, but the body of the paper is

[1] All of the results and properties we require of generalized algebraic theories are satisfied by the special case of quotient inductive inductive types, so the reader who prefers to do so may also refer to Kaposi et al. [KKA19] as a reference.

[2] This is similar to the approach taken by Nordström et al. [NPS90], though we opt for a different logical framework whose category of models is easier for us to handle.

not a nest of formulae in first-order logic. Instead, the contents of the paper are written in informal mathematics which the authors argue can be converted into formulae if necessary. ◇

2.2 The judgments and basic inferences of type theory

We will follow the presentation of Martin-Löf type theory given by Dybjer [Dyb96] and others. Martin-Löf type theory consists of four judgments, each of which is rendered by a sort in our underlying logical framework:

- The sort of contexts $\mathbf{Cx}$.
- The sort of substitutions $\mathbf{Sb}(\Delta, \Gamma)$ which depends on two elements $\Delta, \Gamma : \mathbf{Cx}$.
- The sort of types $\mathbf{Ty}(\Gamma)$ now depending on an element $\Gamma : \mathbf{Cx}$.
- The sort of terms $\mathbf{Tm}(\Gamma, A)$ depending on $\Gamma : \mathbf{Cx}$ and $A : \mathbf{Ty}(\Gamma)$.

We then animate the type theory by closing each of these sorts under various operations and stipulating equations that these operators are subject to. For instance, we will specify that substitutions can be composed through the following composition operator

$$\mathsf{comp} : (\Gamma_0, \Gamma_1, \Gamma_2 : \mathbf{Cx})(\gamma_1 : \mathbf{Sb}(\Gamma_0, \Gamma_1))(\gamma_2 : \mathbf{Sb}(\Gamma_1, \Gamma_2)) \to \mathbf{Sb}(\Gamma_0, \Gamma_1)$$

We also then specify an *equation* governing this composition operator to force it to be associative:

$$\begin{aligned}&(\Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3 : \mathbf{Cx})(\gamma_1 : \mathbf{Sb}(\Gamma_0, \Gamma_1))(\gamma_2 : \mathbf{Sb}(\Gamma_1, \Gamma_2))(\gamma_3 : \mathbf{Sb}(\Gamma_2, \Gamma_3))\\ &\ \to \mathsf{comp}(\Gamma_0, \Gamma_2, \Gamma_3, \mathsf{comp}(\Gamma_0, \Gamma_1, \Gamma_2, \gamma_1, \gamma_2), \gamma_3)\\ &\quad = \mathsf{comp}(\Gamma_0, \Gamma_1, \Gamma_3, \gamma_1, \mathsf{comp}(\Gamma_1, \Gamma_2, \Gamma_3, \gamma_2, \gamma_3))\end{aligned}$$

Remark 2.2.1. In particular, we note that our equality judgments are typed and undirected. They do not, for instance, necessarily arise from some untyped rewriting system defined on preterms and then restricted to well-typed terms. Indeed, by working within a logical framework as we have done there is no notion of *preterm* to speak of.

Moreover, this lack of preterms means that all constituents of a type theory—terms, types, etc—are always regarded up to definitional equality and all operators of the theory therefore respect definitional equality. This means that principles such as congruence or the conversion rule (stipulating that a term of type A is automatically of type B if A and B are equal) are automatic. ◇

Already with these two small examples, it is clear that this process can become verbose quite quickly. Accordingly, we adopt some notations to help specify operators and equations. First, we will often avail ourselves of implicit arguments when these can be easily inferred by the reader. For instance, we might write $\gamma_2 \circ \gamma_1$ rather than $\mathsf{comp}(\dots)$ as the first three arguments can be inferred from γ_2 and γ_1. Second, rather than writing operators and equations as dependent functions, we will use the more standard inference rule and judgment notation. Rather than writing $\gamma : \mathbf{Sb}(\Delta, \Gamma)$, we will write $\Delta \vdash \gamma : \Gamma$ and specify comp and its equation using the following inference rules:

$$\frac{\Gamma_0 \vdash \gamma_1 : \Gamma_1 \qquad \Gamma_1 \vdash \gamma_2 : \Gamma_2}{\Gamma_0 \vdash \gamma_2 \circ \gamma_1 : \Gamma_2} \qquad \frac{\Gamma_0 \vdash \gamma_1 : \Gamma_1 \qquad \Gamma_1 \vdash \gamma_2 : \Gamma_2 \qquad \Gamma_2 \vdash \gamma_3 : \Gamma_3}{\Gamma_0 \vdash \gamma_3 \circ (\gamma_2 \circ \gamma_1) = (\gamma_3 \circ \gamma_2) \circ \gamma_1 : \Gamma_3}$$

We adopt similar familiar notations for the other sorts of our type theory and write $\vdash \Gamma \, \mathsf{cx}$, $\Gamma \vdash A \, \mathsf{type}$, and $\Gamma \vdash M : A$ for $\Gamma : \mathbf{Cx}$, $A : \mathbf{Ty}(\Gamma)$, and $M : \mathbf{Tm}(\Gamma, A)$ respectively.

Standard rules of type theory The inference rules for these judgments vary depending on which particular formulation of Martin-Löf type theory we are considering, but a few rules are ubiquitous. For instance, in addition to the composition operator for substitutions we have just described, we also include an *identity* substitution which serves as a unit for $\circ$:

$$\frac{\vdash \Gamma \, \mathsf{cx}}{\Gamma \vdash \mathsf{id} : \Gamma} \qquad \frac{\Gamma \vdash \delta : \Delta}{\Gamma \vdash \delta \circ \mathsf{id} = \delta : \Delta \qquad \Gamma \vdash \mathsf{id} \circ \delta = \delta : \Delta}$$

We note that this ensures that contexts and substitutions form a category.

Another important structural principle of dependent type theory is *substitution.* There are type and term formers $A[\gamma]$ and $M[\gamma]$ which allow one to apply a substitution to a type or term, shifting it from one context to another:

$$\frac{\Delta \vdash \gamma : \Gamma \qquad \Gamma \vdash A \, \mathsf{type}}{\Delta \vdash A[\gamma] \, \mathsf{type}} \qquad \frac{\Delta \vdash \gamma : \Gamma \qquad \Gamma \vdash M : A}{\Delta \vdash M[\gamma] : A[\gamma]}$$

We impose equations upon this substitution operator to ensure that it respects the composition and identity substitution operators e.g.:

$$\frac{\Gamma_0 \vdash \gamma_1 : \Gamma_1 \qquad \Gamma_1 \vdash \gamma_2 : \Gamma_2 \qquad \Gamma_2 \vdash A \, \mathsf{type}}{\Gamma_0 \vdash A[\gamma_2 \circ \gamma_1] = A[\gamma_2][\gamma_1] \, \mathsf{type}} \qquad \frac{\Gamma \vdash A \, \mathsf{type}}{\Gamma \vdash A[\mathsf{id}] = A \, \mathsf{type}}$$

The two final collections of rules are those governing the formation of contexts. We specify two operations for creating contexts: the operation creating an empty context and the operation extending an existing context with a type:

$$\frac{}{\vdash \mathbf{1} \, \mathsf{cx}} \qquad \frac{\vdash \Gamma \, \mathsf{cx} \qquad \Gamma \vdash A \, \mathsf{type}}{\vdash \Gamma.A \, \mathsf{cx}}$$

In addition to this pair of operations, we have several auxiliary operations and equations governing what substitutions into these contexts exist. The class of rules for $\mathbf{1}$ is the most direct; it specifies that there is a unique substitution mapping into $\mathbf{1}$:

$$\frac{}{\Gamma \vdash ! : \mathbf{1}} \qquad \frac{\Gamma \vdash \delta : \mathbf{1}}{\Gamma \vdash \delta = ! : \mathbf{1}}$$

One can summarize this tersely by stating that $\mathbf{1}$ is *terminal* in the aforementioned category of contexts and substitutions. The rules governing $\Gamma.A$ are slightly more complex. Essentially they state that $\Gamma.A$ behaves like a form of dependent sum such that a substitution into $\Gamma.A$ can be decomposed into a substitution into Γ along with a term of A:

$$\frac{\Delta \vdash \gamma : \Gamma \qquad \Delta \vdash M : A[\gamma]}{\Gamma \vdash \gamma.M : \Gamma.A} \qquad \frac{}{\Gamma.A \vdash \mathbf{p} : \Gamma} \qquad \frac{}{\Gamma.A \vdash \mathbf{v} : A[\mathbf{p}]}$$

$$\frac{\Delta \vdash \gamma : \Gamma \qquad \Delta \vdash M : A[\gamma]}{\Delta \vdash \gamma = \mathbf{p} \circ (\gamma.M) : \Gamma \qquad \Delta \vdash M = \mathbf{v}[\gamma.M] : A[\gamma]} \qquad \frac{\Delta \vdash \gamma : \Gamma.A}{\Delta \vdash \gamma = (\mathbf{p} \circ \gamma).\mathbf{v}[\gamma] : \Gamma.A}$$

This basic collection of rules completes the core principles of Martin-Löf type theory though without any additional operators giving rise to terms or types the theory is slightly degenerate.

Definition 2.2.2. We will refer to any type theory extending the above rules with various type and term operators as a *Martin-Löf type theory.*

2.3 A tour of various connectives of Martin-Löf type theory

In this section, we briefly survey some of the connectives that we will typically work within this thesis. We also include a somewhat extensive discussion on the different formulations of *universes* within type theory, as this is a somewhat complex aspect of Martin-Löf type theory with several trade-offs available.

2.3.1 The sum, product, and unit types

We begin with three standard connectives in dependent type theory: dependent sums, dependent products, and unit types. We begin with the *formation rules*, the operators constructing elements of **Ty**:

$$\frac{\Gamma \vdash A\,\mathsf{type} \qquad \Gamma.A \vdash B\,\mathsf{type}}{\Gamma \vdash \sum_A B\,\mathsf{type} \qquad \Gamma \vdash \prod_A B\,\mathsf{type}} \qquad \frac{}{\Gamma \vdash \mathsf{Unit}\,\mathsf{type}}$$

As a first instance of what will quickly become a routine matter, we must also impose equations on these formation operators. Recall that we are always able to apply a substitution to a type, shifting it from one context to another $A[\gamma]$. Each time we specify an operator of the theory, here formation operators, we will add rules specifying how substitution interacts with the operator:

$$\frac{\Delta \vdash \gamma : \Gamma \qquad \Gamma \vdash A\,\mathsf{type} \qquad \Gamma.A \vdash B\,\mathsf{type}}{\Delta \vdash \big(\sum_A B\big)[\gamma] = \sum_{A[\gamma]} B[\mathsf{id}.\gamma]\,\mathsf{type} \qquad \Delta \vdash \big(\prod_A B\big)[\gamma] = \prod_{A[\gamma]} B[\mathsf{id}.\gamma]\,\mathsf{type}}$$

$$\frac{\Gamma \vdash \gamma : \Gamma}{\Delta \vdash \mathsf{Unit} = \mathsf{Unit}[\gamma]\,\mathsf{type}}$$

These rules, together with all the others we shall eventually add, ensure that substitutions can pass through term and type formers in order to resolve at variables. Even these three rules illustrate the general pattern for substitution rules, so we shall avoid mentioning them further in this overview.

The operators governing terms in all three types are particularly simple to state because they feature both a computation and unicity equality (a β and η law). Accordingly, we can perfectly well organize the data into an isomorphism. For instance, the following isomorphisms fully specify the introduction, elimination, computation, and unicity rules for $\prod$, $\sum$, and Unit:

$$\begin{gathered}\mathbf{Tm}\big(\Gamma, \textstyle\prod_A B\big) \cong \mathbf{Tm}(\Gamma.A, B)\\ \mathbf{Tm}\big(\Gamma, \textstyle\sum_A B\big) \cong \textstyle\sum_{M:\mathbf{Tm}(\Gamma,A)} \mathbf{Tm}(\Gamma, B[\mathsf{id}.M])\\ \mathbf{Tm}(\Gamma, \mathsf{Unit}) \cong \mathbf{1}\end{gathered}$$

Of course, it is often more convenient to unpack the data of these isomorphisms to obtain more familiar syntax and rules. We carry out this process for dependent products to illustrate the general process:

Lemma 2.3.1. *Given an isomorphism $\alpha : \mathbf{Tm}\big(\Gamma, \prod_A B\big) \cong \mathbf{Tm}(\Gamma.A, B)$, the standard operators for lambda-abstraction and function application are definable and satisfy the expected equations.*

Proof. We begin by defining $\mathsf{lam} = \alpha^{-1} : \mathbf{Tm}(\Gamma.A, B) \to \mathbf{Tm}\big(\Gamma, \prod_A B\big)$. The definition of application is only slightly more involved: $\mathsf{app}(M, N) = \alpha(M)[\mathsf{id}.N]$. The calculation of the β and η rules is routine. □

2.3.2 Inductive types

The next class of types to consider are those that have a computation rule but lack a corresponding unicity rule. For reasons we will address in Section 2.4, these tend to be types with a *mapping-out* elimination principle which resembles an induction scheme: to construct $P(x)$ for all $x : A$, it suffices to construct $P(\dots)$. Throughout this thesis, we will make use of several of these *inductive types*, but we will not attempt to provide a general schema for all inductive types here. Instead, we will illustrate the rules for the booleans and natural numbers and note that the other instances in this thesis can be incorporated into type theory by similar rules.

Both Bool and Nat have straightforward formation rules and introduction rules:

$$\frac{}{\Gamma \vdash \mathsf{Nat}, \mathsf{Bool}\ \mathsf{type}} \qquad \frac{}{\Gamma \vdash \mathsf{tt}, \mathsf{ff} : \mathsf{Bool} \qquad \Gamma \vdash \mathsf{z} : \mathsf{Nat}} \qquad \frac{\Gamma \vdash M : \mathsf{Nat}}{\Gamma \vdash \mathsf{suc}(M) : \mathsf{Nat}}$$

One might hope that the elimination principle for booleans states something akin to the elimination rules for dependent products or sums, which ensure that every element of a type came from one of its introduction rules. Unfortunately, such a rule is not possible with booleans; we will not typically have an isomorphism $\mathbf{2} \cong \mathbf{Tm}(\Gamma, \mathsf{Bool})$ (indeed, not what if Γ contained a variable of type Bool?). What we can hope for, however, is that *from the perspective of a type*, there are only two elements of Bool. This somewhat informal statement is meant to capture a *weak orthogonality* condition that we shall revisit more fully in Chapter 3. For now, we shall be more concrete and specify the elimination rule for Bool and Nat as follows:

$$\frac{\Gamma.\mathsf{Bool} \vdash B\,\mathsf{type} \qquad \Gamma \vdash N_0 : B[\mathsf{id}.\mathsf{tt}] \qquad \Gamma \vdash N_1 : B[\mathsf{id}.\mathsf{ff}] \qquad \Gamma \vdash M : \mathsf{Bool}}{\Gamma \vdash \mathsf{if}(B, N_0, N_1, M) : B[\mathsf{id}.M]}$$

$$\frac{\Gamma.\mathsf{Bool} \vdash B\,\mathsf{type} \qquad \Gamma \vdash N_0 : B[\mathsf{id}.\mathsf{tt}] \qquad \Gamma \vdash N_1 : B[\mathsf{id}.\mathsf{ff}]}{\Gamma \vdash \mathsf{if}(B, N_0, N_1, \mathsf{tt}) = N_0 : B[\mathsf{id}.\mathsf{tt}] \qquad \Gamma \vdash \mathsf{if}(B, N_0, N_1, \mathsf{ff}) = N_0 : B[\mathsf{id}.\mathsf{ff}]}$$

$$\frac{\begin{array}{c}\Gamma.\mathsf{Nat} \vdash B\,\mathsf{type} \\ \Gamma \vdash N_0 : B[\mathsf{id}.\mathsf{z}] \qquad \Gamma.\mathsf{Nat}.B \vdash N_1 : B[\mathbf{p}^2.\mathsf{suc}(\mathbf{v}[\mathbf{p}])] \qquad \Gamma \vdash M : \mathsf{Nat}\end{array}}{\Gamma \vdash \mathsf{rec}(B, N_0, N_1, M) : B[\mathsf{id}.M]}$$

$$\frac{\Gamma.\mathsf{Bool} \vdash B\,\mathsf{type} \qquad \Gamma \vdash N_0 : B[\mathsf{id}.\mathsf{tt}] \qquad \Gamma \vdash N_1 : B[\mathsf{id}.\mathsf{ff}] \qquad \Gamma \vdash M : \mathsf{Nat}}{\begin{array}{c}\Gamma \vdash \mathsf{rec}(B, N_0, N_1, \mathsf{z}) = N_0 : B[\mathsf{id}.\mathsf{z}] \\ \Gamma \vdash \mathsf{rec}(B, N_0, N_1, \mathsf{suc}(M)) = N_1[\mathsf{id}.M.\mathsf{rec}(B, N_0, N_1, M)] : B[\mathsf{id}.\mathsf{suc}(M)]\end{array}}$$

2.3.3 Identity types

Thus far the connectives discussed have been relatively uncontroversial; the formulation of dependent sums, booleans, etc. is essentially ubiquitous. We arrive at our first major point of divergence with the identity type. This type is meant to internalize the equality between terms as a type, but there are two distinct formulations of the identity type (intensional and extensional). The situation is somewhat fraught: extensional identity types are semantically simpler and more powerful absent additions like univalence, but have a far inferior proof theory compared with intensional identity types. In particular, they cannot be implemented in a proof assistant like Coq or Agda, so we are often forced to work with the intensional formulation instead.[3] Within this thesis we shall have use for both forms of identity type: extensional identity types will be used when reasoning within type theory, while intensional identity types will be used for the type theories about which we will reason.

Both the intensional identity type Id and the extensional identity type Eq have identical formation and introduction rules, so we introduce them together:

$$\frac{\Gamma \vdash A\,\mathsf{type} \qquad \Gamma \vdash M, N : A}{\Gamma \vdash \mathsf{Eq}(A, M, N), \mathsf{Id}(A, M, N)\,\mathsf{type}} \qquad \frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{refl} : \mathsf{Eq}(A, M, M) \qquad \Gamma \vdash \mathsf{refl} : \mathsf{Id}(A, M, M)}$$

Notation 2.3.2. When no ambiguity can arise, we will often suppress the A in $\mathsf{Id}(A, M, N)$ or $\mathsf{Eq}(A, M, N)$.

The distinction between the two lies in their respective elimination principles. The extensional equality type is the simpler of the two, with a principle allowing one to parlay a proof of $\mathsf{Eq}(A, M, N)$ into a definitional equality:

$$\frac{\Gamma \vdash P : \mathsf{Eq}(A, M, N)}{\Gamma \vdash M = N : A \qquad \Gamma \vdash P = \mathsf{refl} : \mathsf{Eq}(A, M, N)}$$

[3] Somewhat surprisingly, the intensional identity type induces an incredibly rich homotopical structure within type theory but we shall not pursue this perspective.

Phrased differently, the extensional identity type has an introduction and elimination rule which can be bundled into a single isomorphism:

$$\mathsf{Tm}(\Gamma, \mathsf{Eq}(A, M, N)) = \{\star \mid M = N\}$$

The intensional identity type, by contrast, has an elimination principle encoding a weak orthogonality condition similar to Bool or Nat:

$$\frac{\begin{array}{c}\Gamma.A.A[\mathbf{p}].\mathsf{Id}(\mathbf{v}[\mathbf{p}], \mathbf{v}) \vdash B\,\mathsf{type} \\ \Gamma.A \vdash N : B[\mathsf{id}.\mathbf{v}.\mathbf{v}.\mathsf{refl}] \qquad \Gamma \vdash M_0, M_1 : A \qquad \Gamma \vdash P : \mathsf{Id}(M_0, M_1)\end{array}}{\Gamma \vdash \mathsf{J}(B, N, P) : B[\mathsf{id}.M_0.M_1.P]}$$

$$\frac{\Gamma.A.A[\mathbf{p}].\mathsf{Id}(\mathbf{v}[\mathbf{p}], \mathbf{v}) \vdash B\,\mathsf{type} \qquad \Gamma.A \vdash N : B[\mathsf{id}.\mathbf{v}.\mathbf{v}.\mathsf{refl}] \qquad \Gamma \vdash M : A}{\Gamma \vdash \mathsf{J}(B, N, \mathsf{refl}) = N[\mathsf{id}.M] : B[\mathsf{id}.M.M.\mathsf{refl}]}$$

2.3.4 Universes

The final connective left to describe is the universe of small types. Essentially, this is a type whose elements codify certain types. Already several points of divergence emerge: some presentations of type theory insist that elements of the universe are *literally* types. In our setting, this is ill-formed: terms are of a different sort. Accordingly, we require an explicit decoding function converting between an element of the universe and a type (this is the primordial dependent type):

$$\frac{}{\Gamma \vdash \mathfrak{U}\,\mathsf{type}} \qquad \frac{\Gamma \vdash M : \mathfrak{U}}{\Gamma \vdash \mathsf{El}(M)\,\mathsf{type}}$$

Ideally, a universe would have a code for every type in the system—El would be invertible—but this runs afoul of Cantor's argument. The best we can hope for is a universe that contains a code for every small type i.e. those which do not mention the universe itself. Unfortunately, we run into a deficiency of standard non-homotopical type theory: there is no way to equip the universe with a suitable universal property. The ideal universal property would ensure that elements of the universe determine types up to equivalence, but the former is too small to contain the latter. Indeed, in standard type theory we have only one equality between a pair of elements but we may have many equivalences. Accordingly, we compromise and merely ensure that we have enough elements of the universe by requiring it to be *closed* under various type-formers.

For every connective e.g. dependent products, we add a corresponding operation on the universe:

$$\frac{\Gamma \vdash M : \mathfrak{U} \qquad \Gamma.\mathsf{El}M \vdash N : \mathfrak{U}}{\Gamma \vdash \mathsf{ProdCode}\,M\,N : \mathfrak{U}}$$

The idea is that this simulates the operation of forming dependent products on elements of the universe. This intuition is given force through the following equation:

$$\frac{\Gamma \vdash M : \mathfrak{U} \qquad \Gamma.\mathsf{El}M \vdash N : \mathfrak{U}}{\Gamma \vdash \prod_{\mathsf{El}M} \mathsf{El}N = \mathsf{El}(\mathsf{ProdCode}\,M\,N)\,\mathsf{type}}$$

This process is then repeated for every connective, save the universe itself.

Remark 2.3.3. In some instances, we will have occasion to relax equation governing El to a specified isomorphism—so-called *weak Tarski universes*—but this is a technical detail that can be generally ignored. We will return to it properly in Chapter 8 where we will actually take advantage of this weakening to simplify certain arguments. ◇

2.4 Metatheorems and their motivation

We conclude this section by discussing some of the main metatheorems proven about type theory and what purpose they serve. In particular, we are concerned with decidability of conversion, canonicity, and admissibility of substitution.

2.4.1 Decidability of conversion

The easiest property to state and motivate is the decidability of conversion:

Property 2.4.1 (Decidability of conversion)**.** *The equality in the sorts* $\mathbf{Ty}(\Gamma)$ *and* $\mathbf{Tm}(\Gamma, A)$ *is decidable.*

This property is necessary, fundamentally, for implementation purposes. The central complexity of standard implementions of type theory is the type-checker: the portion of the implementation which decides whether the input term has the proposed type. Type-checking dependent type theory is extremely difficult because of the following conversion rule, enabling one to silently replace a type by an equal type:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A = B\,\mathsf{type}}{\Gamma \vdash M : B}$$

This rule is present in any formal system, dependent or not, but MLTT has an especially complex theory of type equality owing to dependence. In particular, while we only require decidability of types in order to implement type-checking, dependence ensures that this is only possible if the equality of terms is also decidable. To see this in action, consider the term refl and the type $\mathsf{Id}(A, M, N)$. This could only be well-typed if $M = N$ so that the above conversion rule applies and $\mathsf{Id}(A, M, N) = \mathsf{Id}(A, M, M)$. This then places an implementation in the unenviable position of having to decide the equality of arbitrary terms at arbitrary type, which can quickly become undecidable.

Many of the design considerations for type theory are motivated by decidability considerations. We have, in fact, already encountered such a trade-off in the formulation of identity types. Extensional identity types lead to an undecidable type-checking problem, which is why they are typically eschewed in favor of the less convenient intensional identity type.

Normalization In practice, one establishes the decidability of conversion through a *normalization function*. Tersely, a normalization function embeds the sets of terms, types, etc. into sets with an equality easily seen to be decidable, thereby yielding the necessary result. In fact, this process is somewhat reversible:

Lemma 2.4.2. *If equality of types is decidable, there is a computable embedding of types into the natural numbers.*

Proof. Let us note that types can be presented as a quotient of derivation trees, by a general result of either GATs or QIITs. We will define a map from $\mathbf{Ty}(\Gamma)$ first at the level of derivation trees and then argue that it respects the equality and therefore extends to a map out of $\mathbf{Ty}(\Gamma)$.

Fix a bijective Gödel numbering on derivations (through any of the standard methods). We define $F(\mathcal{D}) : \mathsf{Nat}$ to the be minimal natural number n such that the derivation $\mathcal{D}'$ corresponding to n represents the same type as $\mathcal{D}$. This function is computable; we iterate through the natural numbers starting with 0 and using the assumed conversion algorithm to test equality at each numeral. By construction, derivations representing the same type are sent to the same number so this induces a computable function on types. It follows by definition, moreover, that this function is an embedding. □

While the above lemma shows that the existence of *some* normalization algorithm is equivalent to the decidability of conversion, we often wish for a more structured algorithm. Indeed, the embedding and its codomain are often chosen to make other properties easy to deduce. For instance, in order to minimize the annotations needed in a proof assistant, one may which to show that $\prod_{A_0} B_1 = \prod_{A_0} B_1$ implies $A_0 = A_1$. This sort of injectivity result is not a priori a consequence of normalization, but by carefully controlling the behavior of the normalization function it can fall out more-or-less immediately from the behavior of the normalization function on $\prod$.

2.4.2 Canonicity

Canonicity is a result which applies to closed terms of type theory:

Property 2.4.3 (Canonicity). *If* $\mathbf{1} \vdash M : \mathsf{Bool}$ *then* $M = \mathsf{tt}$ *or* $M = \mathsf{ff}$

Intuitively, this result codifies the *computational content* of type theory. We must be somewhat indirect in how this is stated; type theory does not come equipped with some form of operational semantics or reduction relation, only equality. However, by asking for a constructive proof of canonicity, we are essentially requiring an interpreter. This enables us to evaluate closed booleans all the way down to either tt or ff. The data of such an interpreter is a map from $\mathbf{Tm}(\mathbf{1}, \mathsf{Bool}) \to \mathbf{2}$ and so, from this viewpoint the requirement that $M = \mathsf{tt}$ or $M = \mathsf{ff}$ states that the interpreter must be 'correct'. We are generally only able to extract such an interpreter from a constructive proof of canonicity but even without constructivity canonicity does yield useful information.

The computational point of view is the main motivation for proving canonicity: type theory is intended to be a computational foundation for mathematics and canonicity provides a concrete check in this regard. More than this, however, canonicity is also a check to ensure that sufficiently many equations have been added to the type theory. This is why we insist on obtaining tt or ff in particular: we want to avoid a situation where the equational theory is decidable through the absence of equations. After all, if we had refused to include any equations at all the theory, we would certainly be able to show equality is decidable, but canonicity would fail.

2.4.3 Admissibility of substitution

Canonicity and decidability of conversion are difficult to prove. They usually come from sophisticated arguments involving logical relations or gluing and often establishing one of these results for a theory is an entire publication's worth of material on its own. The last metatheorem under discussion—the admissibility of substitution—is quite different.

Firstly, it is often quickly stated as a lemma and proving that it holds is usually a simple exercise. More concretely, however, canonicity and decidability of conversion are properties that can be stated *objectively*—without reference to a particular presentation of syntax. One way to see this is that both can be characterized as a property of a model and the metatheorem can be rephrased as *this property holds of the initial model.* Admissibility of substitution is quite different and, thus far, has only been characterized as a subjective property of a presentation of syntax.

Property 2.4.4 (Admissibility of substitution)**.** *Every type and term former in the theory except* $\mathbf{v}$ *commutes with substitution: for an operator* θ *accepting* n *arguments, there is a collection of operators on substitutions* $(F_i)_{i<n}$ *such that* $\theta(T_0, \dots)[\gamma] = \theta(T_0[F_0(\gamma)], \dots)$*.*

In our case, admissibility of substitution holds more-or-less by construction. Each time we introduce a type former, we specify an equation of precisely the form taken above.

Unlike canonicity and decidability of conversion, the statement of Property 2.4.4 is somewhat messy and has been subject to several alterations as type theory has developed. Broadly, each time the logical framework presenting the theory has changed, some of the *subjective* metatheorems like Property 2.4.4 become automatic and those which still must be proven manually change form. For instance, some presentations of Martin-Löf type theory did not include a distinguished sort of substitutions. Instead, practitioners proved a sort of *generalized cut* result stating something akin to the following:

$$\frac{\Delta \vdash N : A \qquad \Gamma_0, x : A, \Gamma_1 \vdash M : B}{\Gamma_0, \Delta, \Gamma_1 \vdash M[N/x] : B[N/x]} \tag{2.1}$$

In the above, $M[N/x]$ is an operation on raw terms that performs some sort of capture-avoiding substitution. People eventually passed from individual substitutions like this to simultaneous substitutions between contexts and, from there, to an algebra of substitutions formalized in just the same way as types and terms [Mar92]. Having established substitutions as a formal object in type theory, $M[\gamma]$ became a term constructor like any other and Rule 2.1 became a rule like any other.

Even with this development, the admissibility of substitution did not truly become automatic. The essential content of Rule 2.1 was not merely that we could change the context; it was also crucial that changing the context did not change the shape of a term or type. This latter point is what is captured by Property 2.4.4 and it remains vital. For instance, it enables users to prove lemmas hypothesizing over variables and use them later on.

Remark 2.4.5. Historically, the introduction of formal contexts within logic has been a necessity to avoid contradictions (particularly those around hypothesizing over empty domains) but it has never been representative of actual informal mathematics. Indeed, when working on paper one will not typically find explicit binders or the like. Instead, certain arguments will presuppose the existence of an object to be substantiated by subsequent argument. In particular, there is no conceptual distinction between an argument fully instantiated compared to one which presently lacks a parameter. The role of the admissibilty of substitution is to lend some formal foundation to this informal idea. It ensures that contexts and binding are carefully noted while simultaneously ensuring that no construction is ever affected by the instantiation of a hypothetical. ⋄

Remark 2.4.6. In light of these arguments, a more provocative third distinction may be drawn between admissibilty of substitution and the prior metatheorems. If one is given a type theory which fails to satisfy e.g. canonicity, one might argue that this is a poorly-behaved type theory. I would argue that a theory which does not satisfy the admissibility of substitution or an equivalent is simply not a type theory. The ability to ignore contexts when working in a theory—the fundamental consequence of admissibility of substitution—is a defining characteristic of type theory. ⋄

For a similar historical evolution, one might consider the congruence of definitional equality. Traditional presentations of type theory contained various *congruence* rules stating that e.g., if $M = N$ then $\pi_0(M) = \pi_0(N)$. These rules are entirely absent in our presentation of type theory because they are enforced by the principles of our logical framework; every connective is congruent with respect to equality. Thus, while historically congruences for equality were debated and enforced explicitly, they have since faded into the background of the logical framework and become entirely unremarkable. This is also the case with other metatheorems: subject reduction, presupposition, etc.[4] We emphasize that this does not mean that these are not important properties of a type theory. To the contrary, they are so important that a consensus has been reached to exclude systems that do not satisfy, e.g., subject reduction from our discourse.

Admissibility of substitution is on a similar path: we have replaced the generalized cut principle and admissible substitutions with a cleaner formulation rendering Rule 2.1 automatic. It is then natural to ask why substitution has not been folded into the logical framework entirely so that all connectives are stable under substitution *by fiat*. In fact, modern logical frameworks do enforce this! The logical frameworks considered by Gratzer and Sterling [GS20], Haselwarter and Bauer [HB21], Nordström et al. [NPS90], and Uemura [Uem21] all enforce stability under substitution automatically. In some sense, the presentation of type theory given in this chapter is arguably an antiquated approach.

This choice of approach is deliberate however and not merely an attempt to force the reader to appreciate the advances of modern logical frameworks. As we consider various type theories we will immediately fall outside the scope of all the aforementioned logical frameworks. A key property each of these logical frameworks capitalizes upon to make substitution automatically admissible is the fact that operators in MLTT only modify the context through extension. In particular, each connective may bind an additional variables within its subterms but otherwise Γ is silently carried around. Modal type theories are not like this: the entire purpose of modal type theory is to study connectives that cannot be formulated in this manner. It follows immediately that (1) modal type theories do not fall within the scope of typical logical frameworks and (2) the admissibility of substitution is a subtle and difficult property to guarantee in the modal setting.

As of the writing of this thesis, no general logical framework for modal type theory has emerged. It is not yet clear that modal type theory is sufficiently developed for one to be put forward and so this thesis will work with the more flexible and lower-level

[4] It is a truism of mathematics that the best proof of a theorem is to make it a definition.

logical framework used above. Consequently, we must pay close attention to substitution and ensure that each connective is stable under substitution as we add to our theory. We shall see how the desire to maintain the admissibility of substitution in the presence of modalities is an animating principle of modal type theory and a central contribution of this thesis is to put forward new techniques for achieving it.

3 Category theory and categorical semantics

> Logic now belongs in a *category* whose "objects" are the formulas and whose *morphisms* are the proofs.
>
> Jean-Yves Girard
> *The Blind Spot*

Category theory features prominently in this thesis, both as a language for specifying type theory and as a tool for constructing models. In this chapter, we fix basic notation and conventions observed throughout for categories and survey the classical theory of categorical models of Martin-Löf type theory.

Remark 3.0.1. Despite this chapter including a section on basic category theory, it should not be confused with an introduction to category theory itself. For this, we recommend any of several excellent books e.g., Awodey [Awo10] or Riehl [Rie16]. We make use of some of the machinery of presentable categories which is thoroughly described by the first two chapters of Adámek and Rosický [AR94]. We use some rudimentary 2-category theory as well which is touched on briefly by Riehl [Rie16] and described more thoroughly by e.g., Lack [Lac09]. Ideas from topos theory are frequently used and Mac Lane and Moerdijk [MM92] contains all the necessary information. ◇

3.1 Basic notions in category theory

We begin by fixing the notation and conventions for basic objects of category theory.

Convention 3.1.1. We shall work under the assumption of enough universes throughout this thesis. By convention, we shall fix a pair of Grothendieck universes $\mathcal{V}_0 \in \mathcal{V}_1$ and refer to elements of $\mathcal{V}_0$ as *small* and those of $\mathcal{V}_1$ as large.

Convention 3.1.2. For this thesis, unless specified otherwise a category $\mathcal{C}$ is taken to be locally small and therefore shall have a large set of objects $\mathsf{Ob}(\mathcal{C})$ and small sets $\hom(c_0, c_1)$. We write $\mathsf{Mor}(\mathcal{C})$ for the (large) set of all morphisms.

Example 3.1.3. *The following examples of categories will be used frequently:*

1. *Any preorder P induces a small category P whose set of objects is P itself and such that* $\hom(p_0, p_1) = \{\star \mid p_0 \leq p_1\}$.

2. **Set** *is the category of small sets where* $\hom(X, Y)$ *is the set of functions X to Y.*

3. **Cat** *is the category of small categories and functors between them.*

4. $\hom(\mathcal{C}, \mathcal{D})$ *is the category whose objects are functors from* $\mathcal{C}$ *to* $\mathcal{D}$ *and whose morphisms are natural transformations.*

5. $\mathcal{C}^{\mathsf{op}}$ *is category whose objects are the same as* $\mathcal{C}$ *but whose set of morphisms* $\hom_{\mathcal{C}^{\mathsf{op}}}(c_0, c_1)$ *is defined to be* $\hom_{\mathcal{C}}(c_1, c_0)$*.*

6. *We write* $\mathbf{PSh}(\mathcal{C}) = \hom(\mathcal{C}^{\mathsf{op}}, \mathbf{Set})$ *for the category of presheaves on* $\mathcal{C}$*.*

7. *Given a Grothendieck topology* J *on* $\mathcal{C}$*, we write* $\mathbf{Sh}(\mathcal{C}, J)$ *for the category of sheaves on* $\mathcal{C}$*.*

8. *Given an object* $c : \mathcal{C}$*, we write* $\mathcal{C}/c$ *for the slice category over* c*.*

9. *Given a presheaf* $X : \mathbf{PSh}(\mathcal{C})$*, we write* $\int X$ *for the category of elements of* X*.*

Notation 3.1.4. We fix several pieces of notation for standard categorical concepts:

- We write $\mathsf{lim}_{i:\mathcal{I}}\, c_i$ and $\mathsf{colim}_{i:\mathcal{I}}\, c_i$ for the limit and colimit of a diagram $c : \mathcal{I} \longrightarrow \mathcal{C}$ when such a (co)limit exists.

- We write $\mathbf{0}$ and $\mathbf{1}$ for the initial and terminal objects, respectively. We denote the universal morphism from and to them by ! and !.

- We write $\mathcal{C}^{\rightarrow}$ for the category of arrows and commuting squares within $\mathcal{C}$.

- We write $c_0 \times_{c_1} c_2$ for the pullback of c_0 along $c_2 \longrightarrow c_1$.

- Given $f : c_0 \longrightarrow c_1$ in a category $\mathcal{C}$ with pullbacks, we write $f^* : \mathcal{C}/c_1 \longrightarrow \mathcal{C}/c_0$ for the pullback functor.

- Given a replete subcategory $\mathcal{D} \subseteq \mathcal{C}^{\rightarrow}$, we write $\mathcal{D}^{\mathsf{cart}}$ for the wide subcategory of $\mathcal{D}$ spanned by pullback (i.e., Cartesian) squares.

- Given a functor $f : \mathcal{C} \longrightarrow \mathcal{D}$, we write $f_! \dashv f^* \dashv f_*$ for the adjoint triple between $\mathbf{PSh}(\mathcal{C})$ and $\mathbf{PSh}(\mathcal{D})$. In particular, $f_!$ and f_* are determined by left and right Kan extension while f^* sends X to $X \circ f$.

Convention 3.1.5. We will frequently have occasion to work with (strict) 2-categories, bicategories, pseudofunctors and strict 2-functors. By convention, we refer to the strict variants by default so e.g., 2-category should be interpreted as strict 2-category.

Remark 3.1.6. We shall frequently make use of the fact that any bicategory can be strictified to a 2-category and any pseudofunctor between 2-categories can be strictified to a 2-functor. We refer the reader to Lack [Lac09] for a proof (of a far more general version of this theorem). ◇

Notation 3.1.7. We write $\bullet$ for horizontal composition of 2-cells within a 2-category and $\circ$ for vertical composition. In the special case of horizontally composing α with the identity 2-cell $f \longrightarrow f$ we write $\alpha \star f$ or $f \star \alpha$.

Definition 3.1.8. A pair of morphisms $f : c \longrightarrow d$ and $g : d \longrightarrow c$ within a 2-category $\mathcal{E}$ are said to be adjoint $f \dashv g$ when there exist a pair of 2-cells $\eta : \mathsf{id} \longrightarrow G \circ F$ and $\epsilon : F \circ G \longrightarrow \mathsf{id}$ such that $(\epsilon \star F) \circ (F \star \eta) = \mathsf{id}$ and $(G \star \epsilon) \circ (\eta \circ G) = \mathsf{id}$ (the so-called triangle identities).

Remark 3.1.9. A pair of functors are adjoint in **Cat** just when they are adjoint in the classical sense. ◇

3.2 Universes in categories

We now begin the transition from discussing purely categorical matters to categorical type theory. An important step along this road is the notion of a categorical *universe* inspired by Streicher [Str05].[1] Given the importance that categorical universes play in our theory of semantics of dependent type theory, we afford ourselves a somewhat more in-depth exposition of the theory.

Definition 3.2.1. A universe in a category $\mathcal{C}$ is a (large) set of morphisms $\mathcal{S} \subseteq \mathsf{Mor}(\mathcal{C})$ which is stable under pullback: if $f : c_0 \longrightarrow c_1 \in \mathcal{S}$ and $g : d \longrightarrow c_1$ is an arbitrary morphism, then $d \times_{c_1} c_0$ exists and $g^* f : d \times_{c_1} c_0 \longrightarrow d \in \mathcal{S}$.

Remark 3.2.2. Assuming $\mathcal{C}$ has all pullbacks, we can define a universe more concisely as a subfibration of the cartesian fibration $\mathsf{cod} : \mathcal{C}^{\rightarrow} \longrightarrow \mathcal{C}$. ◇

To motivate this definition, we begin by considering how one might encode a family of objects in category theory. Dependent type theory is designed to handle families of objects well as types themselves are taken over a context. In other words, to define a family of e.g. groups indexed by the natural numbers, one simply considers an ordinary group in a context containing Nat. This is quite distinct from category theory, as objects do not depend on other objects. This mismatch is the quintessential difficulty in the semantics of dependent type theory but, fortunately, well-studied solutions are available.

Let us begin by considering **Set** where a family is relatively easy to describe. Suppose we are given a family of sets $(Y_x)_{x \in X}$ over X. While the family is not itself a set, we can package it into a set by taking the disjoint sum $Y = \sum_{x \in X} Y_x$. There is an evident projection function $Y \longrightarrow X$ sending $\mathsf{in}_x y \mapsto x$. We now note that there is a *bijection* between families of sets over X and *display maps*, morphisms of the form $\bullet \longrightarrow X$. We have already described the assignment of family to a display map and the reverse comes from taking preimages: $\big(f : Y \longrightarrow X\big) \mapsto (f^{-1}(x))_{x \in X}$.

Picking out the set over $x \in X$ can be accomplished by taking the pullback along $x : \mathbf{1} \longrightarrow X$. Given any subset $X' \subseteq X$ more generally, taking the pullback $X' \times_X Y \longrightarrow X'$ corresponds to the *restriction* of the family $(Y)_{x \in X}$ to X'. More generally, pulling back along any morphism $X' \longrightarrow X$ allows us to relabel a family over X to one over X'.

Remark 3.2.3. We can again relate this to type theory: a family is a type $\Gamma \vdash A\,\mathsf{type}$ while a display map becomes the substitution $\Gamma.A \vdash \mathbf{p} : \Gamma$. ◇

[1] Streicher [Str05] differs slightly in what constitutes a universe as Streicher reserves the term universe for a universe in our sense equipped with additional structure.

After switching attention from genuine families to display maps, it becomes straightforward to generalize to arbitrary categories; a family is simply another name for a morphism. We will refer to the codomain X of a family $Y \longrightarrow X$ as the base and the domain Y as the total space. Pulling back this family allows one to change the base and, in particular, pulling back to $\mathbf{1}$ allows one to extract the particular object over some global point of the base.

Remark 3.2.4. The display map definition of a family is commonly used in geometrical settings where one wishes for the index of a family to be some form of space *and* for the family of spaces to vary continuously over the base. This is automatically encoded in the display family definition. For a concrete theorem to this effect, we refer the reader to the equivalence between sheaves on a space X and étale spaces over X. $\diamond$

Having adopted this perspective on families, the definition of a universe of a category is less mysterious: it is isolating some collection of families which is stable under relabeling. The stability under relabeling, in particular, ensures that if a family lies within a universe each of the individual members of that family are likewise contained in the universe.

Some caution is required when making the transition to families and universes in arbitrary categories: in the case of families in **Set** all the information of a display map $Y \longrightarrow X$ was contained in the collection of pullbacks $Y \times_X \mathbf{1}$ for all $\mathbf{1} \longrightarrow X$. This is a rather special feature of **Set**: generally knowing the fibers of $\mathbf{1}$ is neither enough to reconstruct or separate families over a general object X. In a general category $\mathcal{C}$, even if there is a terminal object there is no reason why $\hom(\mathbf{1}, -) : \mathcal{C} \longrightarrow \mathbf{Set}$ should be either full or faithful, and therefore no reason to believe that the information of a display map over $\mathbf{1}$ is generally sufficient.

We are generally interested in working with universes that are closed under various operations, akin to how types are closed under dependent sums, dependent products, etc.. We now review a few of the most essential closure conditions.

Definition 3.2.5. A universe $(\mathcal{C}, \mathcal{S})$ contains the terminal object if $\mathcal{C}$ contains a terminal object and $\mathbf{1} \longrightarrow \mathbf{1}$ lies within $\mathcal{S}$. More generally, a universe is said to contain some object X when $X \longrightarrow \mathbf{1} \in \mathcal{S}$.

Definition 3.2.6. A universe $(\mathcal{C}, \mathcal{S})$ is closed under dependent sums when $\mathcal{S}$ is closed under composition.

This definition is surprising at first glance: why should closure under composition have any bearing on dependent sums. First, let us ensure that this definition corresponds to the expected result in **Set**.

Lemma 3.2.7. *For any universe $\mathcal{S}$ in* **Set** *the following are equivalent*

1. *$\mathcal{S}$ is closed under composition*

2. *If $(Y_x)_{x \in X} \in \mathcal{S}$ and that $(Z_{x,y})_{(x,y) \in Y} \in \mathcal{S}$ then $\left(\sum_{y \in Y_x} Z_{x,y}\right)_{x \in X} \in \mathcal{S}$.*

Proof. Let us first prove (1) implies (2). Fix X, Y, and Z as above. Note that by assumption $Z \longrightarrow Y \in \mathcal{S}$ and by definition, $Z \cong \sum_{x \in X} \sum_{y \in Y_x} Z_{x,y}$. Accordingly, by closure under composition $\pi_1 : Z \longrightarrow X \in \mathcal{S}$ and passing back to a family presentation this is precisely what is required by (2). The proof of that (2) implies (1) is similar. $\square$

A more categorical intuition for the above definition is also available. It is well-known that dependent sum along $f : Y \longrightarrow X$ can be categorically recast as the left adjoint to the pullback functor $f^* : \mathcal{C}/X \longrightarrow \mathcal{C}/Y$ and this adjoint is given by $f \circ -$. The requirement that $\mathcal{S}$ be closed under composition is therefore equivalent to asking that $\mathcal{S}$ is closed under this left adjoint.

This last motivation leads to the next definition. Just as dependent sums along f are the left adjoint to f^*, dependent products are the *right adjoint*. Accordingly, we formulate the following condition for closure under dependent products:

Definition 3.2.8. A universe $(\mathcal{C}, \mathcal{S})$ is closed under dependent products if whenever $f : X \longrightarrow Y$ and $g : Y \longrightarrow Z$ are elements of $\mathcal{S}$ the *pushforward* $g_* f$ exists and is also contained in $\mathcal{S}$.

Definition 3.2.9. We shall refer to a category $\mathcal{C}$ with a universe $\mathcal{S}$ closed under the unit type, dependent sums, and dependent products as a *category with display maps.* In this situation, we shall refer to elements $f \in \mathcal{S}$ as a display map.

The definition of a universe as a class of maps may have been surprising to type theorists for a different reason: within type theory a universe is a single object and maps into the universe classify types. A universe $(\mathcal{C}, \mathcal{S})$ in our sense is nothing like this: there is no single display map $E \longrightarrow B$ codifies $\mathcal{S}$. This choice is deliberate: it turns out that the choice of any such map is highly non-canonical and so, while such a structure is useful in type theory the categorical definition of universe is better behaved if $\mathcal{S}$ is taken as primitive rather than some choice of map which generates it.[2]

That said, the *existence* of such a generic map is an important property to impose upon a universe:

Definition 3.2.10. A generic map $E \longrightarrow B$ for a universe $(\mathcal{C}, \mathcal{S})$ is an element of $\mathcal{S}$ such that every map in $\mathcal{S}$ is a pullback of $E \longrightarrow B$.

Notation 3.2.11. In due course, we shall become sloppy about the distinction between a universe and the generic map generating it. We shall, however, be careful about the distinction for the present.

The properties of a universe can be captured by imposing pieces of structure on a generic map $E \longrightarrow B$. However, there is a switch from mere property to non-unique chosen structure. For instance, consider the closure of a universe under $\mathbf{1}$. This can be rephrased as requiring a chosen pullback square of the following shape:

$$\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & E \arrow[d] \\
\mathbf{1} \arrow[r] & B
\end{tikzcd}$$

However, while asking $\mathbf{1} \longrightarrow \mathbf{1} \in \mathcal{S}$ is a mere property, there can be many different choices of pullback squares of the above form.

[2] Conceptually, the assignment of an object B to collection $\bullet \longrightarrow B \in \mathcal{S}$ lands most naturally in groupoids and not in sets. Consequently, a representation for this functor is “too large” to fit inside $\mathcal{C}$. A generic family is a poor approximation of this functor being represented.

The same process can be carried out for dependent sums and products, though with some additional complexity. For dependent products, for instance, We essentially reduce the problem of showing $g_* f \in \mathcal{S}$ for every suitable $f, g \in \mathcal{S}$ to the requirement that this be the case for a single pair of morphisms and then encode that single requirement through a pullback square. For this, we need the following result:

Definition 3.2.12. Given a morphism $f : c_0 \longrightarrow c_1$ in a category $\mathcal{C}$ such that $c_0^* : \mathcal{C} \longrightarrow \mathcal{C}/c_0$ and $f_* : \mathcal{C}/c_0 \longrightarrow \mathcal{C}/c_1$ exist, we define *polynomial functor* $\mathbf{P}_f = (c_1)_! f_* c_0^*$ [GK13].

Lemma 3.2.13 (Awodey [Awo18])**.** *A universe* $(\mathcal{C}, \mathcal{S})$ *with a generic map* $\tau : E \longrightarrow B$ *is closed under dependent products just when the pushforward* $\mathbf{P}_\tau(\tau) \in \mathcal{S}$

Accordingly, we may replace the property "$(\mathcal{C}, \mathcal{S})$ is closed under dependent products" with the structure of a pullback square $\mathbf{P}_\tau(\tau) \longrightarrow \tau$. We can similarly formulate the existence of dependent sums as a choice of pullback square of a certain shape.

Remark 3.2.14. One way to view this distinction between chosen pullback square or closure conditions is analogous to the distinction between a category with finite limits and a category with *chosen* finite limits. In the presence of choice one can pass back and forth between these two situations, but only the former is invariant under equivalence. As a purely academic matter, passage to ∞-categories offers a potential resolution to this dilemma. In this setting, some categories admit true *object classifiers*: generic maps for whom the choice of classifying pullback square is unique. In such a setting, both closure under dependent products and the choice of code witnessing this fact are mere properties. See Nguyen and Uemura [NU22] for systematic exploitation of this fact. ⋄

We conclude this section with the notion of a hierarchy of universes.

Definition 3.2.15. A hierarchy of universes with generic maps with $\mathcal{C}$ consists of a sequence of universes $\mathcal{S}_i$ such that each has a generic map $E_i \longrightarrow B_i$, for each i there is an inclusion $\mathcal{S}_i \subseteq \mathcal{S}_{i+1}$ and for each i the base of the generic map B_i lies in $\mathcal{S}_{i+1}$.

Suppose that $\mathcal{S}_0 \subseteq \mathcal{S}_1$ and that both universes have generic families $\tau_i : E_i \longrightarrow B_i$. As an immediate consequence of the definition of generic map, we may choose a Cartesian square $u : \tau_0 \longrightarrow \tau_1$. We emphasize, however, that this choice is necessarily arbitrary and it need not commute with the chosen cartesian squares closing τ_0 and τ_1 under e.g., dependent products. When we have chosen all necessary squares and u does commute strictly with these cartesian squares we shall say that the inclusion $\mathcal{S}_0 \subseteq \mathcal{S}_1$ commutes with e.g. dependent products. When this holds for all connectives, we shall simply say that the hierarchy of universes is *strictly cumulative.*

3.3 Examples of categories with universes

We conclude the material on purely categorical topics by constructing several examples of categories with universes that will feature prominently in the semantics of type theory. Of particular importance are those in Grothendieck topoi where we can construct extremely well-behaved generic families. These results are explored at length in the expository paper by Gratzer et al. [GSS22]. Accordingly, we will content ourselves with stating the key theorems and refer the reader to op. cit. for detailed proofs.

All of the universes we will consider in this section are collections of maps in some Grothendieck topos $\mathcal{E}$ satisfying some size constraint on their fibers. Intuitively, this constraint is meant to model a family of small sheaves with a potentially large indexing sheaf. We note that some constraint of this form is necessary if we wish to have a universe with a generic family without running into size issues. To generalize some notion of size to arbitrary Grothendieck topoi, we use the notion of *compact objects*.

Definition 3.3.1 (Definition 1.13, Adámek and Rosický [AR94])**.** In a category $\mathcal{E}$, an object X is said to be κ-compact when $\hom(X,-)$ commutes with κ-filtered colimits.[3]

Example 3.3.2. *A set is κ-compact just when it has cardinality strictly smaller than κ. A presheaf $X : \mathbf{PSh}(\mathcal{C})$ is κ compact if $X(c)$ is κ-compact for each $c : \mathcal{C}$, though the converse need not be true unless $\mathcal{C}$ is κ-small.*

To use κ-compactness as a defining property for a universe, we must a notion of compactness which applies to morphisms.

Definition 3.3.3 (Shulman [Shu19])**.** A map $f : X \longrightarrow Y$ is said to be *relatively κ-compact* when if the pullback $X \times_Y Z$ is κ-compact for any $Z \longrightarrow Y$ where Z is κ-compact.

For our purposes, the details of relative κ-compactness are less important than its closure properties. In particular, for a given Grothendieck topos $\mathcal{E}$ there exists arbitrarily large cardinals κ such the class of relatively κ-compact enjoys excellent closure properties. Moreover, since every map is relatively κ-presentable for sufficiently large κ, a hierarchy of well-behaved universes will be large enough to include any family of interest in $\mathcal{E}$.

Theorem 3.3.4. *Given Grothendieck topos $\mathcal{E}$, there exists a cardinal regular λ such for any inaccessible $\kappa \rhd \lambda$*[4]*, the class of relatively κ-compact forms a universe $\mathcal{S}_\kappa$ enjoying the following properties:*

- *The universe contains the terminal object and indeed contains all monomorphisms.*
- *The subobject classifier and the natural number object are both contained within the universe.*
- *The universe is closed under dependent sums and dependent products.*

Moreover, there are multiple distinct ways of constructing generic families for these universes. Some depend on the specific realization of the topos as a particular left-exact localization of a presheaf topos:

Theorem 3.3.5 (Awodey [Awo22] and Hofmann and Streicher [HS97])**.** *In a presheaf topos $\mathbf{PSh}(\mathcal{C})$, one can define a generic family $E_\kappa \longrightarrow B_\kappa$ for $\mathcal{S}_\kappa$ for any $\kappa \geq |\mathcal{C}|$ where B is defined as follows:*

$$B(c) = \mathbf{PSh}_{\mathcal{V}_\kappa}(\mathcal{C}/c)$$

In the above, $\mathcal{V}_\kappa$ is the Grothendieck universe spanned by sets of rank less than κ.

[3] If the notion of filtered diagrams is unfamiliar, it suffices to consider colimits indexed by a κ-directed partial order.

[4] The relation $\rhd$ is pronounced "sharply larger than". It is a technical necessity in the theory of accessible categories and is discussed by Adámek and Rosický [AR94, Chapter 2].

The essential advantage of the *Hofmann–Streicher generic families* is the direct definition. This enables us to choose well-behaved operators witnessing the closure of the universe under various operations such that, in particular, they organize into a cumulative hierarchy.

Corollary 3.3.6. *For any sequence of inaccessible cardinals* $\cdots > \kappa_i > \kappa_{i-1} > \cdots > |\mathcal{C}|$*, the universes* $\mathcal{S}_{\kappa_i}$ *of relatively* κ_i*-compact morphisms with Hofmann–Streicher generic families organize into a strictly cumulative hierarchy.*

Obtaining well-behaved generic families in arbitrary Grothendieck topoi is more subtle than it may appear. Given a sheaf topos $\mathbf{Sh}(\mathcal{C})$, the most direct approach would be to consider a subobject of the Hofmann–Streicher generic family spanned by those small presheaves $\mathbf{PSh}_{\mathcal{V}_\kappa}(\mathcal{C})$ which happen to be sheaves. While this does induce a map of presheaves generic for relatively κ-compact families of sheaves, it is not itself a sheaf. Essentially, the resulting presheaf satisfies enjoys amalgamation only up to coherent isomorphism [XE16]. While one can pursue this further and develop this construction into a 2-sheaf (a stack) [CMR17],[5] one can also pursue an alternative construction for a generic family of sheaves. For instance, Streicher [Str05] observed that one could *sheafify* the entire family rather than attempting to cut down the Hofmann–Streicher generic map:

Theorem 3.3.7 (Streicher [Str05])**.** *Given a sheaf topos* $\mathbf{Sh}(\mathcal{C}) \subseteq \mathbf{PSh}(\mathcal{C})$*, there exists a* λ *such that the sheafification map applied to the Hofmann-Streicher generic family* $\mathbf{a}(E_\kappa \longrightarrow B_\kappa)$ *is a generic family for the universe of relatively* κ*-compact sheaves* $\mathcal{S}_\kappa$ *for every* $\kappa \rhd \lambda$*.*

Unfortunately, constructing the generic family in this manner loses many of the conveniences of the Hofmann–Streicher construction. In particular, it is no longer clear that the maps witnessing closure under dependent products, dependent products, etc. can be chosen in a cumulative manner.

Corollary 3.3.8. *For any sheaf topos* $\mathbf{Sh}(\mathcal{C})$ *there exists a* λ *such that for any sequence of strongly inaccessible cardinals* $\ldots \kappa_i \rhd \kappa_{i-1} \ldots \rhd \lambda$*, the universes* $\mathcal{S}_{\kappa_i}$ *of relatively* κ_i*-compact maps together with the generic families constructed in Theorem 3.3.7 form a hierarchy of universes.*

While this is perfectly sufficient categorically, in the denotational semantics of type theory strict cumulativity plays an outsized role. We are therefore interested in improving Corollary 3.3.8 to support a strictly cumulative hierarchy of universes.

In certain special cases, a more refined construction for a generic family in a sheaf topos is available. For instance, Zwanziger [Zwa22] shows that the Hofmann–Streicher generic family can be adapted directly to account for sheaf topoi with enough points. We opt for a slightly different approach, choosing instead to focus on a key property of the Hofmann–Streicher generic family which accounts for much of its good behavior.

[5] In fact, this pattern continues until one eventually concludes that sub-presheaf of the ∞-categorical Hofmann-Streicher construction spanned by those small ∞-presheaves which are sheaves is, in fact, a proper ∞-sheaf [Lur09; RSS20].

Definition 3.3.9. A generic family $\tau : E \longrightarrow B$ for a universe $\mathcal{S}$ is said to satisfy *realignment* whenever it is weakly right-orthogonal to all monomorphisms in $\mathcal{S}_{\mathsf{cart}}$, the subcategory of $\mathcal{C}^{\to}$ spanned by elements of $\mathcal{S}$ and cartesian transformations between them.

More explicitly, given a pair of families $f, g \in \mathcal{S}$ along with a cartesian monomorphism $\alpha : f \longrightarrow g$, it is possible to extend a choice of classifying square for f to a classifying square for g:

$$\begin{tikzcd}
f \arrow[r] \arrow[d, tail] & \tau \\
g \arrow[ur, dotted]
\end{tikzcd}$$

Lemma 3.3.10 (Orton and Pitts [OP18])**.** *The Hofmann–Streicher generic family satisfies realignment.*

Realignment has been used in a variety of contexts. It was explicitly introduced by Shulman [Shu15a] to replicate part of the construction of the universal Kan fibration constructed by Kapulkin and Lumsdaine [KL21]. It has also appeared outside of type theory in, e.g., Cisinski [Cis19] and Lurie [Lur22] in the construction of various classifying families. The essence of realignment is to give some control over the choice of classifying square associated to some element of the universe; there is no way to ensure a unique classifying square generally but with realignment we can ensure that the otherwise arbitrary choice satisfies at least some constraints. We shall capitalize frequently on realignment in Chapter 4 but our interest in it for now comes from the following observation:

Theorem 3.3.11 (Gratzer et al. [GSS22] and Shulman [Shu15a])**.** *Given a hierarchy of universes $\mathcal{S}_i$ whose generic families $\tau_i : E_i \longrightarrow B_i$ satisfy realignment such that the cartesian squares $\tau_i \longrightarrow \tau_{i+1}$ are monomorphisms, it is possible to choose codes witnessing closure under all connectives which organize into a strictly cumulative hierarchy.*

In other words, while the sheafified Hofmann–Streicher generic family is too wild to manually construct a strictly cumulative choice of codes, it suffices instead to construct any generic family in $\mathbf{Sh}(\mathcal{C})$ which satisfies realignment. Such a construction is essentially provided by Shulman [Shu15a] through a small-object argument with further details spelled-out by Gratzer et al. [GSS22]:

Theorem 3.3.12. *For any Grothendieck topos $\mathcal{E}$, there exists a λ such that for all $\kappa \rhd \lambda$ there is a generic family satisfying realignment for the universe of relatively κ-compact morphisms.*

Corollary 3.3.13. *For any Grothendieck topos, there exists a regular cardinal λ such that for any sequence of strongly inaccessible cardinals $\dots \kappa_i \rhd \kappa_{i-1} \dots \rhd \lambda$, the universes of relatively κ_i-compact families $\mathcal{S}_{\kappa_i}$ together with the generic families constructed in Theorem 3.3.12 form a strictly cumulative hierarchy of universes.*

Remark 3.3.14. We note that Theorem 3.3.12 does not fully supplant the other constructions of generic families discussed in this section. While the Hofmann–Streicher generic family and those others derived from it is definable within a fairly weak constructive setting, Theorem 3.3.12 relies on choice. In settings where we strive to work purely constructively (e.g., Chapters 4 and 8), the constructively-valid generic families are essential.

We further note that Lemma 3.3.10 has a constructive analog: realignment is valid just when the base of the Cartesian morphism $f \longrightarrow g$ is a level-wise decidable monomorphism. Explicitly, realignment for this family is constructively valid when the characteristic map $\mathsf{cod}(g) \longrightarrow \Omega$ factors through the subobject Ω_{dec} of level-wise decidable propositions. See Orton and Pitts [OP18] for further details. ◇

3.4 Models of dependent type theory

We now transition from discussing pure category theory to type theory. We begin by defining a model of type theory along with a homomorphism of models. As mentioned already in Chapter 2, at this point this process is entirely mechanical; we took a signature in a logical framework as our definition of type theory and an immediate consequence of this is a definition of model. In fact, this process has already been carried out by Dybjer [Dyb96], who repackaged the definition to arrive at the well-known notion of a *category with families (CwF)*. In this section, we will therefore define CwFs and discuss their reformulation into the language of natural models [Awo18]. As a consequence of this process, we shall see that the key difference between category theory and the categorical semantics of type theory is whether one chooses to focus on a universe within a category or on its generic family.

Following Dybjer [Dyb96], we define a model of Martin-Löf type theory as follows:

Definition 3.4.1. A *category with families (CwF)* consists of the following data:

1. A category $\mathcal{C}$,
2. a presheaf $\mathcal{T} : \mathbf{PSh}(\mathcal{C})$,
3. a presheaf $\mathcal{T}^\bullet : \mathbf{PSh}(\int \mathcal{T})$
4. $\mathcal{C}$ has a chosen terminal object
5. For each $c : \mathcal{C}$ and $A \in \mathcal{T}(c)$, there exists a chosen object $d : \mathcal{C}$ equipped with a map $p : d \longrightarrow c$ and an element of $q \in \mathcal{T}^\bullet(d, A \cdot p)$ which together induce a bijection between morphisms $\hom(\bullet, d)$ and pairs $f : \bullet \longrightarrow d$ and $M \in \mathcal{T}^\bullet(\bullet, A \cdot f)$.

Remark 3.4.2. Spelling out the details, objects $\mathcal{C}$ represent contexts while morphisms are substitutions. The presheaf $\mathcal{T}$ sends a context to the set of types in that context, while the functorial action corresponds to the substitution operation on types. The presheaf $\mathcal{T}^\bullet$ sends a context and a type in that context (an element of $\int \mathcal{T}$) to the set of terms of that type. Once again, the functorial action interprets substitution. The two additional properties stipulate the existence of the empty context and context extension, respectively. ◇

Notation 3.4.3. We will write $c.A$ for the chosen context extension of $c : \mathcal{C}$ by $A \in \mathcal{T}(c)$.

Definition 3.4.4. A morphism between $(\mathcal{C}, \mathcal{T}_\mathcal{C}, \mathcal{T}^\bullet_\mathcal{C}, \mathbf{1}_\mathcal{C}, -.-)$ and $(\mathcal{D}, \mathcal{T}_\mathcal{D}, \mathcal{T}^\bullet_\mathcal{D}, \mathbf{1}_\mathcal{D}, -.-)$ consists of a functor $F : \mathcal{C} \longrightarrow \mathcal{D}$ together along with a natural transformations $\alpha_\mathcal{T} : \mathcal{T}_\mathcal{C} \longrightarrow F^*\mathcal{T}_\mathcal{D}$ and $\alpha_{\mathcal{T}^\bullet} : \mathcal{T}^\bullet_\mathcal{C} \longrightarrow (F, \alpha_\mathcal{T})^*\mathcal{T}^\bullet_\mathcal{D}$ which preserve the chosen terminal objects, send $c.A$ to $F(c).\alpha_\mathcal{T}(A)$, etc.

We note that the definition of a CwF is phrased almost entirely in terms of well-known categorical requirements except the last requirement used to model context extension. As it happens, this too can be recast as a more standard construction following [Awo18; Fio12]. In particular, we begin by noting that a presheaf $\mathcal{T}^\bullet : \mathbf{PSh}(\int \mathcal{T})$ is equivalent to a morphism $\tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T}$ via the canonical equivalence between $\mathbf{PSh}(\int \mathcal{T})$ and $\mathbf{PSh}(\mathcal{C})/\mathcal{T}$. In this case, the structure modeling context extension is equivalent to asking τ to be a *representable map* in the following sense:

Definition 3.4.5. A morphism $f : X \longrightarrow Y$ in a presheaf category is said to be representable when each fiber of a representable object is itself representable. That is, for each $\mathbf{y}(c) \longrightarrow Y$, there exists d fitting into the following pullback diagram:

$$\begin{tikzcd}
\mathbf{y}(d) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d] \\
\mathbf{y}(c) \arrow[r] & Y
\end{tikzcd}$$

Remark 3.4.6. There are several useful reformulations of the above definition: f is a representable morphism if and only if f is representable in $\mathbf{PSh}(\mathcal{C})/Y$ if and only if the induced map $\int X \longrightarrow \int Y$ has a right adjoint. $\diamond$

Asking for a choice of objects $c.A$ for each $c : \mathcal{C}$ and $A \in \mathcal{T}(c)$ is precisely equivalent to asking for a choice of objects d representing the fiber over $\lfloor A \rfloor : \mathbf{y}(c) \longrightarrow \mathcal{T}$. Accordingly, we may repackage the definition of a CwF as follows:

Lemma 3.4.7. *A category with family is precisely determined by a category* $\mathcal{C}$ *with a terminal object and a representable morphism* $\tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T} : \mathbf{PSh}(\mathcal{C})$.

Definition 3.4.8. Given a CwF $\tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T}$ on $\mathcal{C}$, there is a universe $[\tau] = \{f^*\tau \mid f \in \hom(\bullet, \mathcal{T})\}$ in $\mathbf{PSh}(\mathcal{C})$ where τ is a generic map.

Thus far we have discussed models of MLTT with no connectives, so it remains only to discuss the closure of a CwF under various connectives. Specifying the closure under connectives like dependent sums, dependent products, booleans, etc. is essentially a piecemeal business. Each connective comes with a few additional pieces of structure which can be added independently of each other. We can also specify this structure either in terms of the classical formulation of CwFs discussed first or phrase it in terms of the natural model. We shall opt for the latter in this introduction as it is generally much more expeditious. We refer the reader to Awodey [Awo18] for a detailed exposition linking the two perspectives.

Definition 3.4.9. A model of type theory $\left(\mathcal{C}, \tau : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}\right)$ is closed under unit types when there exists a map $\mathbf{1} \longrightarrow \mathcal{T}$ fitting into the following pullback diagram:

$$\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^{\bullet} \arrow[d] \\
\mathbf{1} \arrow[r] & \mathcal{T}
\end{tikzcd}$$

Remark 3.4.10. It is helpful to view this definition as a categorical rephrasing of the concise "isomorphism" definition of the unit type given in Chapter 2. The existence of this pullback square specifies (1) a closed type Unit such that (2) there is exactly one element of Unit. From this perspective, the square itself codifies the formation and introduction rules of the unit type. The fact that it is a pullback square yields the elimination principle along with the β and η laws. ⋄

The remaining negative connectives—those with η laws—are specified in much the same way:

Definition 3.4.11. A model of type theory $\left(\mathcal{C}, \tau : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}\right)$ is closed under dependent products when there exists a pullback diagram of the following shape:

$$\begin{tikzcd}
\mathbf{P}_{\tau}(\mathcal{T}^{\bullet}) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^{\bullet} \arrow[d] \\
\mathbf{P}_{\tau}(\mathcal{T}) \arrow[r] & \mathcal{T}
\end{tikzcd}$$

More concisely, there is a cartesian square $\mathbf{P}_{\tau}(\tau) \longrightarrow \tau$.

Definition 3.4.12. A model of type theory $\left(\mathcal{C}, \tau : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}\right)$ is closed under dependent sums when there exists a pullback square $\tau_2 \longrightarrow \tau$ where τ_2 is the map satisfying $\mathbf{P}_{\tau_2} = \mathbf{P}_{\tau} \circ \mathbf{P}_{\tau}$ [GK13].

Definition 3.4.13. A model of type theory $\left(\mathcal{C}, \tau : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}\right)$ is closed under extensional identity types when there is a pullback square $\delta \longrightarrow \tau$ where $\delta : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}^{\bullet} \times_{\mathcal{T}} \mathcal{T}^{\bullet}$ is the diagonal map.

We note that all of these requirements can be phrased entirely in terms of $[\tau]$:

Theorem 3.4.14. *A model of type theory* $(\mathcal{C}, \tau)$ *is closed under dependent products (respectively dependents sums or unit types) if and only if the universe* $[\tau]$ *is closed under dependent products (resp. dependent sums or unit types).*

Unfortunately, this pattern does not continue for types like booleans, the natural numbers, or intensional identity types. Indeed, for the latter, we note that coproducts like booleans or natural number objects in $\mathbf{PSh}(\mathcal{C})$ are never representable, so $\mathbf{2} \longrightarrow \mathbf{1} \in [\tau]$ will never hold.

This parallels our experience with the syntax of type theory, where we saw that defining types like booleans or intensional identity types is a more involved affair. When working purely with type theory, we directly wrote out elimination principles and their β rules and the same process is available to us when working with CwFs. However, doing so requires dropping back down to the lower level of abstraction and switching away from natural models. Accordingly, we must opt for a more complicated definition which states that there exists an object X in $[\tau]$ along with a suitably anodyne map $\mathbf{2} \longrightarrow X$.

Definition 3.4.15. A map $i : A \longrightarrow B$ is said to be anodyne if for each square $i \longrightarrow \tau$ there exists a lift:

$$\begin{tikzcd} A \arrow[r] \arrow[d] & \mathcal{T}^\bullet \arrow[d] \\ B \arrow[r] \arrow[ur, dotted] & \mathcal{T} \end{tikzcd} \tag{3.1}$$

More generally, we shall say i is anodyne for a universe $\mathcal{S}$ if each square $i \longrightarrow f$ has a lift for each $f \in \mathcal{S}$. A map is stably anodyne if each pullback of that map is anodyne.

Asking for such an anodyne map $\mathbf{2} \longrightarrow X$ is equivalent to asking that X and $\mathbf{2}$ be isomorphic *from the perspective of a type* and the resulting lifts can be used to interpret the elimination principle. This is almost enough to capture booleans, but there is a final wrinkle. The elimination rule for booleans is stable with respect to substitution, but if we merely ask for a (stably) anodyne map, there is no reason why the resulting family of lifts will be suitably natural. We will rectify this by introducing the machinery of *lifting structures* to bundle the data of these elimination rules into a more categorical form:

Definition 3.4.16. In a finitely complete cartesian closed category $\mathcal{C}$, a lifting structure $s : m \pitchfork f$ between two maps $m : A \longrightarrow B$ and $f : X \longrightarrow Y$ is a section to the canonical map $X^B \to Y^B \times_{Y^A} X^A$. Diagrammatically, such a section corresponds to a natural family of lifts for diagrams of the following shape:

$$\begin{tikzcd} Z \times A \arrow[r] \arrow[d] & X \arrow[d] \\ Z \times B \arrow[r] \arrow[ur, dotted] & Y \end{tikzcd}$$

Roughly, we will encode an elimination principle as lifting structure $s : m \pitchfork \tau$ for some well-chosen $m : A \longrightarrow B$. As an extremely rough approximation A encodes the hypotheses of the base case—the data of the introduction rules—while B is the general case. The first conjunct of $\mathcal{T}^B \times_{\mathcal{T}^A} \mathcal{T}^{\bullet A}$ then provides the *motive* to the elimination rule while the second provides the base case. The requirement that the lifting structure s be a section corresponds to the β rule of the putative elimination principle. In reality, this simple intuition is complicated by the need to allow the map $A \longrightarrow B$ to depend on additional

Definition 3.4.17. A model of type theory $(\mathcal{C}, \tau)$ is closed under booleans when it is equipped with the following pieces of structure:

1. A commutative diagram of the following form:

$$\begin{tikzcd}
\mathbf{2} \arrow[r] \arrow[d] & \mathcal{T}^{\bullet} \arrow[d] \\
\mathbf{1} \arrow[r, "B"'] & \mathcal{T}
\end{tikzcd} \tag{3.2}$$

2. A lifting structure $s : m \pitchfork \tau$ where $m : \mathbf{2} \longrightarrow \mathbf{1} \times_{\mathcal{T}} \mathcal{T}^{\bullet}$ is the canonical map induced by the above diagram.

Example 3.4.18. *It is worth unfolding the last point in the above definition slightly. Recall that $s : m \pitchfork \tau$ corresponds to a family of lifts for diagrams of the following shape:*

$$\begin{tikzcd}
Z \times \mathbf{2} \arrow[r] \arrow[d] & \mathcal{T}^{\bullet} \arrow[d] \\
Z \times (\mathbf{1} \times_{\mathcal{T}} \mathcal{T}^{\bullet}) \arrow[r] & \mathcal{T}
\end{tikzcd}$$

It suffices to consider the case where $Z = \mathbf{y}(C)$ is representable. In this situation, the bottom map corresponds to a type A in the context C extended by a boolean—we leave it to the reader to convince themselves the relevant diagram is a pullback. Inspecting the top map, we see that it corresponds to two distinct terms: an element of A instantiated with tt *and one with* ff*. A lift of this diagram then corresponds exactly to the output of boolean elimination: it gives us an element of A over an arbitrary element of the booleans which agrees with the two base cases specified by the top map.*

Definition 3.4.19. A model of type theory $(\mathcal{C}, \tau)$ is closed under intensional identity types when it is equipped with the following pieces of structure:

1. A commutative diagram of the following form:

$$\begin{tikzcd}
\mathcal{T}^{\bullet} \arrow[r] \arrow[d] & \mathcal{T}^{\bullet} \arrow[d] \\
\mathcal{T}^{\bullet} \times_{\mathcal{T}} \mathcal{T}^{\bullet} \arrow[r, "B"'] & \mathcal{T}
\end{tikzcd} \tag{3.3}$$

2. A lifting structure $s : m \pitchfork \mathcal{T} \times \tau$ in $\mathbf{PSh}(\mathcal{C})/\mathcal{T}$ where $m : \mathcal{T}^{\bullet} \longrightarrow (\mathcal{T}^{\bullet} \times_{\mathcal{T}} \mathcal{T}^{\bullet}) \times_{\mathcal{T}} \mathcal{T}^{\bullet}$ is the canonical map induced by the above diagram.

Finally, we note that closing a model under a universe is a relatively routine process. There is no elimination rule to speak of, so it amounts to specifying a pair of maps $\mathbf{1} \longrightarrow \mathcal{T}$ and $\mathbf{1} \times_{\mathcal{T}} \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}$ and then choosing (coherent) codes for the display family induced by pulling by τ along the latter morphism.

Definition 3.4.20. We extend Definition 3.4.4 to models of type theory with various connectives by requiring that the natural transformations $\alpha_{\mathcal{T}}$ and $\alpha_{\mathcal{T}^\bullet}$ strictly commute with each type former. We defer to e.g., Castellan et al. [CCD20] for a more in-depth description.

Remark 3.4.21. Often we will be in the position where we are given a model of type theory $(\mathcal{C}, \tau)$ and we wish to use this structure to reason about $\mathcal{C}$. Some of this is immediate. For instance, we can use the more convenient syntax for dependent products offered by type theory to reason about pushforwards in $\mathcal{C}$.

Often, however, we will require some specific objects or principles from $\mathcal{C}$ in order to carry out a construction. To do this, we note that we can extend type theory by any type, term, or equation present in $(\mathcal{C}, \tau)$ and $(\mathcal{C}, \tau)$ will remain a model for this extended theory. This process of extending type theory to better match a particular model is often referred to as *using type theory as an internal language*. One can make this into a precise mathematical process that constructs a certain bi-equivalence between structured categories and certain type theory [CD14; Uem21]. For our purposes, however, the slightly more informal process of extending type theory with constants and equations is sufficient.

To make this more concrete, consider the work by Orton and Pitts [OP18]. Their goal is to use type theory to construct a model of a cubical type theory by using ordinary type theory as a tool for constructing and reasoning about objects in cubical sets. To this end, they extend type theory with a handful of axioms and equations are then interpreted by important objects such as the interval object within cubical sets. $\diamond$

3.5 Promoting a universe to a model

Thus far we have discussed the definition of a model of type theory. We now show two general techniques for promoting a categorical universe to a model of type theory: one which presupposes the universe comes equipped with a generic family and one which does not. The former is often referred to as the *universe* construction [Voe14] and the latter as the *local universes construction* [LW15] (though we will follow the presentation of Awodey [Awo18]).

The goal of both of these procedures is the same: given a category with a universe $(\mathcal{C}, \mathcal{S})$, produce a model of type theory $\tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T} : \mathbf{PSh}(\mathcal{C})$ in which types correspond to elements of $\mathcal{S}$ and terms correspond to sections.

3.5.1 A model of type theory from a generic family

This process can be carried out relatively directly when $(\mathcal{C}, \mathcal{S})$ comes equipped with a generic family $\pi : E \longrightarrow B$. In this case, we will argue that $\tau = \mathbf{y}(\pi)$ gives rise to the desired model of type theory.

Lemma 3.5.1. *τ is representable.*

Proof. We must show that given any map $\mathbf{y}(C) \longrightarrow \mathbf{y}(B)$, the pullback $\mathbf{y}(C) \times_{\mathbf{y}(B)} \mathbf{y}(E)$ is representable. As the Yoneda embedding is fully faithful, the supplied morphism $\mathbf{y}(C) \longrightarrow \mathbf{y}(B)$ is induced by a map $C \longrightarrow B : \mathcal{C}$. Moreover, as $\pi \in \mathcal{S}$, the pullback $C \times_B E$

of π along $C \longrightarrow B$ exists. As the Yoneda embedding preserves limits, $\mathbf{y}(C \times_B E)$ is the necessary representation of $\mathbf{y}(C) \times_{\mathbf{y}(B)} \mathbf{y}(E)$. □

This already shows that τ is a valid model of type theory, but it remains to close τ under all the connectives of type theory. We can essentially link closure under each connective to a specific requirement on $(\mathcal{C}, \mathcal{S})$. For instance, if $\mathcal{S}$ contains the unit type then so does τ, etc.

Lemma 3.5.2. *If* $(\mathcal{C}, \mathcal{S})$ *is closed under the unit type, dependent sums, extensional identity types, or dependent products, then* τ *is closed under the unit type, dependent sums, extensional identity types, or dependent products respectively.*

Proof. We know that $\mathcal{S}$ has a generic family π, so each of the closure conditions can be reformulated in terms of a certain pullback square involving π. Moreover, these pullback squares use only locally cartesian closed operations (pullback, pushforward along π, etc.). Accordingly, they are all preserved by the Yoneda embedding and they immediately give rise to the required structure on $\mathbf{y}(\pi) = \tau$. □

The process of closing τ under booleans is slightly more complex; $\mathbf{2}$ is never preserved by the Yoneda embedding so we cannot argue directly as before. However, it turns out that we can change the data required to close τ under booleans to avoid directly mentioning $\mathbf{2}$.

Lemma 3.5.3. *A model of type theory* $\left(\mathcal{C}, \tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T}\right)$ *supports booleans when it is equipped with a map* $B : \mathbf{1} \longrightarrow \mathcal{T}$*, a pair of maps* $\mathbf{1} \longrightarrow \mathcal{T}^\bullet$ *over* B*, and a section the canonical morphism* $\mathcal{T}^{\bullet\tau[B]} \to \mathcal{T}^{\tau[B]} \times_{\mathcal{T}\times\mathcal{T}} (\mathcal{T}^\bullet \times \mathcal{T}^\bullet)$ *where* $\tau[B]$ *is the fiber of* τ *over* B*.*

Proof. This follows immediately by unfolding the requirements of a boolean and observing that $X^{\mathbf{2}} = X \times X$. □

In particular, we are able to phrase the requirement that τ is closed under booleans using only operations preserved by the Yoneda lemma. In particular, we conclude the following:

Corollary 3.5.4. *If* $(\mathcal{C}, \mathcal{S})$ *has finite products and contains the coproduct* $\mathbf{2} = \mathbf{1} + \mathbf{1}$ *such that* $\mathbf{2}$ *is contained within* $\mathcal{S}$ *then* τ *is closed under booleans.*

Proof. In this situation, we fix a classifying map $B : \mathbf{1} \longrightarrow B$ for $\mathbf{2} \longrightarrow \mathbf{1}$ with respect to the generic map $\pi : E \longrightarrow B$. The two maps $\mathbf{1} \longrightarrow E$ over B follow immediately from this data and the required lifting structure is a consequence of the fact that $E^{\tau[B]} \to B^{\tau[B]} \times_{B \times B} (E \times E)$ is an isomorphism. □

From the above results, we obtain the following theorem:

Theorem 3.5.5. *If* $(\mathcal{C}, \mathcal{S})$ *has finite products and* $\mathcal{S}$ *is closed under all connectives and has a generic map* π *for* $\mathcal{S}$ *then* $\mathbf{y}(\pi)$ *is a model of type theory with all connectives.*

We note that if $\mathcal{C}$ has a strictly cumulative hierarchy of universes, this model can be improved to have a strictly cumulative hierarchy as well.

Corollary 3.5.6. *There is a model of type theory in any Grothendieck topos* $\mathcal{E}$ *with a hierarchy of strictly cumulative universes.*

Proof. This is a consequence of the above and Corollary 3.3.13. □

Remark 3.5.7. A useful consequence of Corollary 3.5.6 is the ability to use type theory to specify structure in a sheaf category. For instance, the specification of closure under dependent sums in a model is somewhat tedious. If we allow ourselves to use type theory to specify the diagram, it becomes far more succinct:

$$\begin{tikzcd}
\sum_{A:\mathcal{T}} \sum_{B:\mathcal{T}^{\tau(A)}} \sum_{a:A} B\,a \arrow[r] \arrow[d] \arrow[dr, phantom, very near start, "\lrcorner"] & \sum_{A:\mathcal{T}} \tau\,A \arrow[d] \\
\sum_{A:\mathcal{T}} B : \mathcal{T}^{\tau(A)} \arrow[r] & \mathcal{T}
\end{tikzcd}$$

Here we have taken advantage of the fact that we can represent $\tau : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}$ as a type $\mathcal{T}$ and a dependent family $A : \mathcal{T} \vdash \tau\,a$. Similar considerations can make the definition of lifting structures or other complex facets of the model simpler. For instance, to realize $s : m \pitchfork \tau$ it suffices to construct an element of the following type:

$$\prod_{C:B\to\mathcal{T}} \prod_{f:(a:A)\to\tau(C(m\,a))} \sum_{l:(b:B)\to\tau(C\,b)} \prod_{a:A} f\,a = l(m\,a)$$

◇

3.5.2 Constructing a model without a generic family

While the above shows that one can leverage a generic family to promote a universe into a model of type theory, the same can be done without such an assumption. Essentially, even though $(\mathcal{C}, \mathcal{S})$ may not have a generic family, $\mathcal{S}$ does induce a universe in $\mathbf{PSh}(\mathcal{C})$ satisfying all the same properties as $\mathcal{S}$ while additionally supporting a generic family. Since a model of type theory really only requires a universe in $\mathbf{PSh}(\mathcal{C})$, this is sufficient to construct a model. This construction was introduced by Lumsdaine and Warren [LW15] as a means of *splitting* a subfibration of the codomain fibration. However, we will follow Awodey [Awo18] and emphasize its role as an operation upon universes.

For the remainder of this subsection, fix $(\mathcal{C}, \mathcal{S})$ to be a locally cartesian closed category with a universe closed under dependent sums, dependent products, extensional identity types, etc. We shall construct a model of type theory $\tau : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}$ on $\mathcal{C}$ such that a type corresponds to an element of $\mathcal{S}$.

The construction of the putative model of type theory $\tau : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}$ is relatively straightforward:

$$\tau = \coprod_{f\in\mathcal{S}} \mathbf{y}(f) : \coprod_{f\in\mathcal{S}} \mathbf{y}(\mathsf{dom}(f)) \longrightarrow \coprod_{f\in\mathcal{S}} \mathbf{y}(\mathsf{cod}(f))$$

Unfolding definitions, a morphism $\mathbf{y}(C) \longrightarrow \mathcal{T}$ corresponds to a map $f : Y \longrightarrow X \in \mathcal{S}$ and a morphism $C \longrightarrow X$. If we are viewing $Y \longrightarrow X$ as a type over a context X, this data determines a type along with a "pending substitution" $C \longrightarrow X$. This pending substitution intuitively is used to ensure that $\mathcal{T}$ determines a presheaf rather than a pseudofunctor of groupoids.

Lemma 3.5.8. *τ is a representable morphism.*

Proof. We must argue that each fiber of τ over a representable is itself representable. To this end, note that any map $\mathbf{y}(C) \longrightarrow \mathcal{T}$ must factor through some $\mathbf{y}(\mathsf{cod}(f)) \longrightarrow \mathcal{T}$ with $f \in \mathcal{S}$. Computing the pullback in two steps using this factorization, we see that it is represented by $C \times_{\mathsf{cod}(f)} \mathsf{dom}(f)$. □

Remark 3.5.9. In fact, if $\mathcal{S} = \mathcal{C}^{\to}$ then τ is the generic map for the class of representable morphisms. More generally, it is the generic map for the universe of representable maps whose fibers land within $\mathcal{S}$. ◇

Closing τ under the connectives of type theory is a more arduous task than when we were given a generic family. The idea is similar in principle: use the relevant structure on $\mathcal{S}$ to construct the desired classifying square on τ. The complication arises from the more intricate structure of τ. We present a few representative cases and defer the remainder to Lumsdaine and Warren [LW15] or Awodey [Awo18].

Lemma 3.5.10. *τ supports dependent products.*

Proof. We must construct a cartesian square $\mathbf{P}_\tau(\tau) \longrightarrow \tau$. We begin by constructing the base of such a square: a map $\mathbf{P}_\tau(\mathcal{T}) \longrightarrow \mathcal{T}$. To this end, let us consider a point $\mathbf{y}(C) \longrightarrow \mathbf{P}_\tau(\mathcal{T})$. After unfolding definitions, this corresponds to two pairs of maps $\big(f_0 : C \longrightarrow X_0, \pi_0 : Y_0 \longrightarrow X_0 \in \mathcal{S}\big)$ and $\big(f_1 : Y_0 \times_{X_0} C \longrightarrow X_1, \pi_1 : Y_1 \longrightarrow X_1 \in \mathcal{S}\big)$. We must construct a pair $\big(f : C \longrightarrow X, \pi : Y \longrightarrow X \in \mathcal{S}\big)$ naturally in C and such that maps $C \longrightarrow Y$ over f correspond to maps $Y_0 \times_{X_0} C \longrightarrow Y_1$ over f_1.

To this end, we begin by construction π using π_0 and π_1 along with the closure of $\mathcal{S}$ under dependent products. We begin by transposing f_1 to obtain $\widehat{f_1} : f_0 \longrightarrow (A_0 \times X_1)^\pi$ within $\mathcal{C}/X_0$. Pulling back π along the structure map $D = (A_0 \times X_1)^\pi \longrightarrow A_0$ we obtain the following diagram:

$$\begin{tikzcd}
& W = Z \times_{X_0} Y_0 \arrow[r] \arrow[d, "\sigma_0"'] \arrow[dr, phantom, "\lrcorner", very near start] & Y_0 \arrow[d] \\
C \arrow[r, "\widehat{f_1}"'] & Z \arrow[r] & X_0
\end{tikzcd}$$

The bottom composite is f_0. Next, we note that there is an evaluation map $W \longrightarrow X_1$ and we thereby obtain $\sigma_1 : Y_1 \times_{X_1} W \longrightarrow W$ by pulling back π_1. We claim that $(f = \widehat{f_1}, \pi = (\sigma_0)_* \sigma_1)$ is the desired pair.

To this end, we begin by noting that $(\sigma_0)_* \sigma_1$ is contained with $\mathcal{S}$ as the latter is assumed to be closed under dependent products. It is also evidently natural in C—this follows directly from the naturality of transposition. That lifts of f along π correspond to lifts of f_1 along π_1 follows by calculating and standard adjoint yoga. □

Lemma 3.5.11. *τ is closed under extensional equality.*

Proof. As before, we must construct a cartesian square $\delta_\tau \longrightarrow \tau$. We begin by constructing a map $\mathcal{T}^\bullet \times_{\mathcal{T}} \mathcal{T}^\bullet \longrightarrow \mathcal{T}$. We will again construct a map between these two presheaves at $C : \mathcal{C}$ and observe it to be natural after the fact. To this end, fix a pair $\big(f : C \longrightarrow X, \pi : Y \longrightarrow X\big)$ along with a pair of lifts $g_0, g_1 : C \longrightarrow Y$ so that it suffices to construct $\big(h : C \longrightarrow Z, \sigma : W \longrightarrow Z \in \mathcal{S}\big)$ naturally in C such that there is a unique lift of h along σ if and only if $g_0 = g_1$. We consider the map $\delta_\pi : Y \longrightarrow Y \times_X Y$ which is contained within $\mathcal{S}$ by assumption. We claim that that $\big(\langle g_0, g_1\rangle : C \longrightarrow Y \times_X Y, \delta_\pi\big)$ satisfies the necessary criteria. We note that the construction is clearly natural in C. A lift of $\langle g_0, g_1\rangle$ along δ_π is unique if one exists, as δ_π is a monomorphism. Moreover, such a lift exists just when $g_0 = g_1$. □

Lemma 3.5.12. *Assume $\mathcal{C}$ contains the coproduct $\mathbf{2} = \mathbf{1} + \mathbf{1}$ and suppose further that there is a factorization of $\mathbf{2} \longrightarrow \mathbf{1}$ into a morphism $b : \mathbf{2} \longrightarrow B$ and $B \longrightarrow \mathbf{1}$ such that $B \longrightarrow \mathbf{1} \in \mathcal{S}$ and b is stably anodyne.*[6] *Under these assumptions, τ is closed under booleans.*

Proof. Unlike the prior examples, we must exhibit more than a certain cartesian square to show that τ is supports booleans. We begin with the required square in $\mathbf{PSh}(\mathcal{C})$:

$$\begin{tikzcd}
\mathbf{2} \arrow[r, "{[\mathsf{tt}, \mathsf{ff}]}"] \arrow[d] & \mathcal{T}^\bullet \arrow[d] \\
\mathbf{1} \arrow[r, "\mathsf{Bool}"'] & \mathcal{T}
\end{tikzcd} \tag{3.4}$$

To construct Bool, it suffices by Yoneda to define a pair $\mathbf{1} \longrightarrow X$ and $f : Y \longrightarrow X$ where $f \in \mathcal{S}$. We choose $\big(\mathsf{id}, B \longrightarrow \mathbf{1}\big)$. To construct $\mathsf{tt}, \mathsf{ff} : \mathbf{1} \longrightarrow \mathcal{T}^\bullet$ and ensure that the diagram commutes, we must produce a pair of sections for this map. We set $\mathsf{tt} = b \circ \mathsf{in}_1$ and $\mathsf{ff} = b \circ \mathsf{in}_2$ and note that these are both trivially sections of $B \longrightarrow \mathbf{1}$ because any map into the terminal object is equal to any other.

We must now construct a lifting structure in $\mathbf{PSh}(\mathcal{C})$ showing that the induced map $\mathbf{2} \longrightarrow \tau[\mathsf{Bool}]$ is internally orthogonal to τ. It is easiest to proceed by unfolding this requirement slightly whereby we may assume that we are given an object $Z : \mathcal{C}$ along with a pair $\big(f : Z \times B \longrightarrow X, \pi : Y \longrightarrow X \in \mathcal{S}\big)$ along with a pair of morphisms $l_{\mathsf{tt}} : f \circ \langle \mathsf{id}, \mathsf{tt}\rangle \longrightarrow \pi$ and $l_{\mathsf{ff}} : f \circ \langle \mathsf{id}, \mathsf{ff}\rangle \longrightarrow \pi$ in $\mathcal{C}/X$. We must construct a map $l : f \longrightarrow \pi$ which extends both l_{tt} and l_{ff} and this must be natural in Z.

To do this, we follow the usual template of choosing a universal lift and then pulling it back to each specific instance to ensure coherence. This is slightly involved, and so we avail ourselves of the internal language of $\mathcal{C}$ and define V as follows:

$$V = \textstyle\sum_{C : X^B} Y(C\,\mathsf{tt}) \times Y(C\,\mathsf{ff})$$

We then consider the pullback $Y \times_X (V \times B) \longrightarrow V \times B$ which, as the pullback of $Y \longrightarrow X$, resides in $\mathcal{S}$. Using the second and third components of V along with the fact that $V \times \mathbf{2} \longrightarrow V \times B$ is anodyne, we obtain a section $s : V \times B \longrightarrow Y \times_X (V \times B)$.

Finally, $l = \pi_1 \circ s \circ (\langle \widehat{f}, l_{\mathsf{tt}}, l_{\mathsf{ff}}\rangle, \mathsf{id})$ yields the desired extension. Its naturality is immediate, as the only choices where made in the construction of s and did not depend on Z, l_{tt}, l_{ff}, or f. □

Gathering these results together, we conclude the following:

[6] In many cases of interest, may be taken to be $b = \mathsf{id}$.

Theorem 3.5.13. *Under the above assumption that* $(\mathcal{C}, \mathcal{S})$ *is a locally cartesian closed category with a universe closed under all connectives,* τ *is a model of type theory with all connectives.*

4 Synthetic Tait computability

> In as much as intuitionists are willing to believe in a formal language, they do believe in the *disjunction property*
>
> J. Lambek and P. J. Scott
> *Introduction to higher order categorical logic*

We conclude Part I with a discussion of *synthetic Tait computability*, a technique which we will use in Chapters 8 and 9. We shall see how this ties together nearly all of what has been discussed in Chapters 2 and 3: we shall capitalize on the idea of syntax as an initial object in a category of models to derive a concrete and useful metatheorem of ordinary type theory. Along the way, we shall have occasion to take advantage of type theory as an internal language of a presheaf topos with its particularly well-behaved generic family as discussed in Chapter 3.

Our goal is to prove the following for type theory with all standard connectives:

Theorem 4.0.1. *If* $\mathbf{1} \vdash M : \mathsf{Bool}$ *then either* $M = \mathsf{tt}$ *or* $M = \mathsf{ff}$*.*

Remark 4.0.2. While all of the machinery used in this proof will seamlessly scale to type theory with all the standard connectives, we shall only give details for dependent products and booleans. This is a compromise to strike a balance between concision in this preliminary material and giving the reader a representative flavor of the proofs. ◇

In Section 4.1 we introduce the concept of *Artin gluing* at a high level and explain how STC can be used to further streamline gluing arguments to prove theorems like the above. In Section 4.2 we introduce the prerequisites of the main construction and in Section 4.3 we carry out the construction of the gluing model. In Section 4.4 we show how to actually extract the desired conclusion from the existence of the gluing model.

Crucially, our techniques extend apply to dependent type theories equipped and can account for the presence of universes. These two features together are the source of much of the complexity in proofs of metatheorems.

Remark 4.0.3. The reader interested in further expository material on synthetic Tait computability is encouraged to consult Sterling [Ste21] or Sterling [Ste22]. The latter in particular presents a far more elementary account of the ideas behind synthetic Tait computability. ◇

4.1 Gluing and synthetic Tait computability

At its heart, the proof of Theorem 4.0.1 is a proof by induction. What, however, the induction ought to scrutinize and what the motive of the induction ought to be are more subtle questions than they might appear at a glance. As those familiar with proofs of termination-like properties might predict, one cannot simply induct on the size of a term when attempting to show it terminates.[1] One will immediately run into the fact that an equation or step like β-reduction will increase the size of a term. In such a situation, one then frequently turns to the idea of a *logical relation*, motivated by the idea that one can proceed by defining a "stronger induction hypothesis" by induction on the size of the type of a term. Unfortunately, this metric is unavailable to us in type theory with universes: any measure on a type will be just as flawed as attempting to induct on the size of a term.

Fortunately, the way out of this situation is offered by the initiality of syntax in the category of models. The force of initiality is precisely an induction principle allowing us to define an operation on syntax by specifying its behavior on each operator. Let us see how this plays out in a far simpler setting.

Lemma 4.1.1. *Every natural number is either even or odd.*

Proof. This is straightforward to prove by induction, but we will take a slightly circuitous route to it. Let us begin by noting that $(\mathsf{Nat}, [\mathsf{z}, \mathsf{suc}])$ is initial in a category of types X equipped with a map $\mathbf{1} + X \longrightarrow X$ (intuitively, objects equipped with zero and successor) and morphisms of carriers commuting with these maps. We will construct a particular object and use initiality to derive the claim.

Consider the following object:

$$X = \textstyle\sum_{n:\mathsf{Nat}} \mathsf{odd}\, x + \mathsf{even}\, x$$

We can then equip X with the necessary structure map:[2]

$$\begin{array}{l}
\alpha : \mathbf{1} + X \to X \\
\alpha(\mathsf{in}_1(\star)) = (\mathsf{z}, \mathsf{zeroEven}) \\
\alpha(\mathsf{in}_2(n, \mathsf{in}_1(p))) = (\mathsf{suc}\, n, \mathsf{in}_2(\mathsf{evenOdd}\, p)) \\
\alpha(\mathsf{in}_2(n, \mathsf{in}_2(p))) = (\mathsf{suc}\, n, \mathsf{in}_1(\mathsf{oddEven}\, p))
\end{array}$$

There is an evident map $\pi_1 : (X, \alpha) \longrightarrow (\mathsf{Nat}, [\mathsf{z}, \mathsf{suc}])$—a routine calculation shows that it commutes appropriately with α. Since Nat is initial in this category, we also have a map $i : (\mathsf{Nat}, [\mathsf{z}, \mathsf{suc}]) \longrightarrow (X, \alpha)$ and the unicity of maps out of Nat ensures that $\pi_1 \circ i = \mathsf{id}$. Accordingly, the $\pi_2 \circ i$ has the type $(n : \mathsf{Nat}) \to \mathsf{even}\, n + \mathsf{odd}\, n$ as required. □

Our proof of canonicity will essentially boil down to this proof, just with a far more sophisticated (generalized) algebraic structure. In total, we consider the following steps:

1. Define a model which equips each object in the syntactic model with some additional data or properties.

[1] In our position, the size of a term is not an entirely well-defined concept as we are only working with equivalences classes of terms up to definitional equality. This could be worked around if the argument based on such a metric was not already hopeless.

[2] We have assumed a number of basic terms governing even and odd here.

2. Show that the projection map forgetting this data is a homomorphism of models.

3. Use initiality to construct a section to the projection and thereby obtain the claim.

Remark 4.1.2. The concept of a model decorating another model with additional structure is often referred to as a *displayed model* or *displayed algebras*. While we do not explore the concept systematically here, others have done so in prior work [KHS19; KKA19]. ◇

The category of contexts in the sought-after displayed model for our proof is constructed using a standard construction of category theory: Artin gluing.

Definition 4.1.3. Given a functor $F : \mathcal{C} \longrightarrow \mathcal{D}$, the Artin gluing $\mathbf{Gl}(F)$ of F is the category whose objects are triples $\big(C : \mathcal{C}, D : \mathcal{D}, f : D \longrightarrow F(C)\big)$. Morphisms in $\mathbf{Gl}(F)$ are given by pairs of maps $(x, y) : (C_0, D_0, f_0) \longrightarrow (C_1, D_1, f_1)$ fitting into a square:

$$\begin{tikzcd}
D_0 \arrow[r, "y"] \arrow[d] & D_1 \arrow[d] \\
F(C_0) \arrow[r, "F(x)"'] & F(C_1)
\end{tikzcd}$$

One then defines a model of type theory atop this category such that the projection automatically induced by Artin gluing constitutes a homomorphism of models. Thus, this approach to proving metatheorems is often referred to simply as *gluing*.

Remark 4.1.4. The idea of using Artin gluing to construct models to prove metatheorems for logic predates type theory. The earliest applications [Fre78] used gluing to analyze higher-order logic (viewed as the internal logic of an elementary topos). A textbook account of this application of gluing is given by Lambek and Scott [LS88]. ◇

Gluing has been extensively used in type theory for decades [AHS95; AK16; Coq19; Fio02; MS93; Str98], largely following the above proof-sketch. A distinguishing feature of our approach here compared to these prior works is the method by which we build the model of type theory atop the results of gluing. Generally once one has finished using Artin gluing to obtain a category of contexts, it becomes necessary to roll up one's sleeves and manually construct the rest of the structure. This can be quite taxing, especially in the case of complex type theories like those analyzed in this thesis.

In order to simplify this portion of the construction, we utilize *synthetic Tait computability (STC)*, a technique developed primarily by Sterling in collaboration with Carlo Anguili, Robert Harper, the author, and others [Gra22; GB22; Niu+22; Ste21; SA21; SH21; SH22]. The role of STC is to reduce the tedious construction of the model of type theory atop the glued category of contexts into a series of programming exercises in the internal language of a particular presheaf category. As the internal language of a presheaf category, we have access to a rich extensional type theory and, in particular, all the necessary tools to make the construction of the model simple and uniform. Working internally, we avoid some of the difficulties endemic to prior gluing proofs: the need to carry around substitutions, check complex naturality requirements, etc.

4.2 Preliminary constructions

We begin the actual proof of Theorem 4.0.1 by constructing the relevant categories and functors for this situation.

4.2.1 Structured categories and CwFs

We begin with an issue particular to this thesis and modal type theory: since we will primarily be concerned with applying STC to modal type theories, we must deal with a logical framework not well-adapted for working purely categorically. Phrased differently, while type theory is modeled by a category with families, we would rather regard syntax not as an initial CwF, but as an initial structured category. If we were only concerned with Martin-Löf type theory, we could opt for a more convenient logical framework that would make this automatic. For the sake of consistency of what is to come, we define an ad-hoc notion of structured category to work with and then use initiality of syntax among CwFs to show that syntax still occupies a privileged position among such categories.

Definition 4.2.1. A Martin-Löf structured category $\mathcal{E}$ is a locally Cartesian closed category equipped with the following structure:

1. A map $\tau : \mathcal{T}^{\bullet} \longrightarrow \mathcal{T}$ equipped with codes closing the resulting universe under dependent sums, dependent products, and a unit type. We will write e.g., Sig and Prod for the classifying maps associated with the first two conditions.

2. τ is also equipped with the commuting squares and lifting structures closing it under booleans and intensional identity types as described in Section 3.4.

3. There is a morphism $\mathsf{Uni} : \mathbf{1} \longrightarrow \mathcal{T}$ and a map $\mathsf{El} : \mathcal{U} = \mathbf{1} \times_{\mathcal{T}} \tau \longrightarrow \mathcal{T}$ such that the pullback υ of τ along the latter is weakly closed under all connectives.

Remark 4.2.2. Some prior work has referred to similar structures as *higher-order models* of type theory [BKS23]. These structures are explored from the perspective of 2-monad theory by Gratzer and Sterling [GS20]. ◇

Remark 4.2.3. We illustrate the last requirement explicitly for booleans and dependent products. We require a map $\mathbf{1} \longrightarrow \mathcal{U}$ such that the composite map $\mathbf{1} \longrightarrow \mathcal{T}$ classifies the same family as the map code for booleans $\mathsf{Bool} : \mathbf{1} \longrightarrow \mathcal{T}$. We do not, however, require that they are the same map.

For dependent products, we require a map $\widehat{\mathsf{Prod}} : \mathbf{P}_{\upsilon}(\mathcal{U}) \longrightarrow \mathcal{U}$ such that the composite $\mathsf{El} \circ \widehat{\mathsf{Prod}}$ classifies the same family as composite of $\mathbf{P}_{\upsilon}(\mathcal{U}) \longrightarrow \mathbf{P}_{\tau}(\mathcal{T})$ and the code for dependent products.

We note that we do not require υ to be strictly cumulative within τ in the sense of Section 3.2. This is more convenient technically, but the distinction is not essential. ◇

In essence, a Martin-Löf structured category is a model of type theory à la natural models, but where one neglects to require that (1) τ is a morphism of presheaves and (2) that τ is representable. In particular, since all the structure defined on τ uses only the operations of a locally Cartesian closed category, one can define what it means for τ to be equipped with connectives in any LCCC.

Every model of type theory induces an ML category $(\mathcal{C}, \tau)$ precisely by “forgetting” $\mathcal{C}$ and considering $(\mathcal{E} = \mathbf{PSh}(\mathcal{C}), \tau)$. In particular, the syntactic model of type induces the following:

Definition 4.2.4. We denote by $(\mathcal{S}, \tau_{\mathcal{S}})$ the ML category induced by the initial model of type theory constructed from syntax.

Definition 4.2.5. A morphism of ML categories $F : \mathcal{E} \longrightarrow \mathcal{F}$ is a locally Cartesian closed functor which strictly sends $\tau_{\mathcal{E}}$ to $\tau_{\mathcal{F}}$ and strictly preserves all relevant codes.

Theorem 4.2.6 (Quasi-projectivity)**.** *Given a morphism $F : (\mathcal{E}, \tau_{\mathcal{E}}) \longrightarrow (\mathcal{S}, \tau_{\mathcal{S}})$ of ML categories, the following conditions hold:*

1. *Given any representable $\mathbf{y}(\Gamma) : \mathcal{S}$, there exists $[\![\Gamma]\!] : \mathcal{E}$ along with a canonical isomorphism $\alpha : \mathbf{y}(\Gamma) \cong F([\![\Gamma]\!])$.*
2. *Given any morphism $\lfloor A \rfloor : \mathbf{y}(\Gamma) \longrightarrow \mathcal{T}_{\mathcal{S}}$ or $\lfloor M \rfloor : \mathbf{y}(\Gamma) \longrightarrow \mathcal{T}^{\bullet}_{\mathcal{S}}$, there exists $[\![A]\!] : [\![\Gamma]\!] \longrightarrow \mathcal{T}_{\mathcal{E}}$ or $[\![M]\!] : [\![\Gamma]\!] \longrightarrow \mathcal{T}_{\mathcal{E}}$ such that $F([\![A]\!]) \circ \alpha = \lfloor A \rfloor$ or $F([\![M]\!]) \circ \alpha = \lfloor M \rfloor$*

Proof. The proof is a routine exercise in constructing a displayed model. We define a model whose category of contexts is given by triples $(\Gamma : \mathsf{Cx}, E : \mathcal{S}, \alpha : \mathbf{y}(\Gamma) \cong F(E))$ and morphisms are pairs of a substitution and a morphism in $\mathcal{E}$ fitting into the expected commuting square. Likewise, the presheaf of types is defined by sending (Γ, E, α) to the set of pairs $\left(A \in \mathcal{T}_{\mathcal{S}}(\Gamma), f : E \longrightarrow \mathcal{T}_{\mathcal{E}}\right)$ such that $F(f) \circ \alpha = \lfloor A \rfloor$. The definition of terms is analogous.

We close this model under the usual connectives of type theory using the fact that $\mathcal{E}$ is an ML category. The forgetful functor projecting out Γ from (Γ, E, α), A from (A, f), etc. is then easily seen to be a morphism of CwFs.

The conclusion then follows from the initiality of the syntactic model. □

With all of this machinery to hand, we can revise our goal slightly: we will no longer aim to construct a displayed model, but instead a displayed ML category. We will then replace our application of initiality to construct a section to projection with Theorem 4.2.6 which provides a certain “local” section.

4.2.2 Artin gluing and global sections

We now return to the main story and consider constructing the category of contexts for the displayed ML category. As previously mentioned, this is done through Artin gluing the $\mathcal{S}$—presheaves on the syntactic category of contexts—along a certain functor. The exact functor chosen depends on what metatheorem is being proven and choosing the correct one is the main source of creativity in a proof done in this style. In essence, each metatheorem characterizes terms in a particular subset of contexts and the characterization is stable under some subset of substitutions. These two considerations induce a (non-full) subcategory of the category of contexts $\hom_i(\mathcal{C}, \mathsf{Cx})$, and we will glue along the induced functor $i^* : \mathcal{S} \longrightarrow \mathbf{PSh}(\mathcal{C})$. Intuitively, this gluing category allows us to speak about (proof-relevant) predicates on terms and types in contexts drawn from $\mathsf{Ob}\,\mathcal{C}$ provided those predicates are stable under the morphisms of $\mathcal{C}$.

In this case, we hope to prove a result about *closed* terms. Thus the subcategory under consideration is generated by $\mathbf{1}$ and so it is the terminal category. Unfolding the associated nerve functor, we will glue along the global sections functor $\Gamma = \hom(\mathbf{1}, -)$.

Definition 4.2.7. We define $\mathcal{G} = \mathbf{Gl}(\Gamma : \mathcal{S} \longrightarrow \mathbf{Set})$.

Lemma 4.2.8. *The projection map $\pi : \mathcal{G} \longrightarrow \mathcal{S}$ is a logical functor with both left and right adjoints.*

We immediately reap some rewards from choosing to glue along a continuous functor out of $\mathcal{S}$—a presheaf category—rather than manipulating the category of contexts directly; the resulting category is extremely well-behaved.

Theorem 4.2.9 (Carboni and Johnstone [CJ95])**.** *$\mathcal{G}$ is a presheaf category.*

We isolate one particular object in $\mathcal{G}$ for future use:

Lemma 4.2.10. *There is a subterminal object $U \hookrightarrow \mathbf{1}$ in $\mathcal{G}$ such that π is equivalent to the pullback functor $\mathcal{G} \longrightarrow \mathcal{G}/U$.*

Proof. The subterminal in question is $(\mathbf{1}_{\mathcal{S}}, \mathbf{0}_{\mathbf{Set}}, !)$. That restricting by U has the desired effect follows directly from calculation. □

4.2.3 The language of synthetic Tait computability

In light of Theorem 4.2.9, only one task remains in our construction of a displayed ML category: we must construct a morphism $\tau_{\mathcal{G}}$ in $\mathcal{G}$ which is sent to $\tau_{\mathcal{S}}$ by π and close $\tau_{\mathcal{G}}$ under all the connectives of type theory in a manner compatible with $\tau_{\mathcal{S}}$. Rather than constructing these externally, we will take advantage of the fact that $\mathcal{G}$ is a presheaf topos and therefore admits a rich internal language as a result of Corollary 3.3.6.

Theorem 4.2.11. *$\mathcal{G}$ admits a model of extensional type theory with a strict universe of propositions Ω and a hierarchy of cumulative universes.*

We will now set about extending this language with various constants to better shape it into an internal language for $\mathcal{G}$. We begin by internalizing $U \hookrightarrow \mathbf{1}$.

Extension 4.2.1. *We postulate a proposition* $\mathbf{syn} : \Omega$ *internalizing $U \hookrightarrow \mathbf{1}$.*

This proposition proves to be tremendously useful in light of the equivalence between $\mathcal{G} \longrightarrow \mathcal{G}/U$ and $\mathcal{G} \longrightarrow \mathcal{S}$. In particular, by assuming $\mathbf{syn}$ within this type theory we can reason within $\mathcal{S}$. Moreover, weakening by $\mathbf{syn}$ corresponds precisely to π so that we can describe the behavior of objects in $\mathcal{G}$ under π by analyzing their properties with respect to the following *open modality* [Joh02; RSS20]:

$$\bigcirc A = \mathbf{syn} \to A$$

Remark 4.2.12. It is illuminating to compute a more explicit form of the functor $X \mapsto X^U$ based on the definition of $\mathcal{G}$ given in Definition 4.1.3. In this case, we may represent X as a morphism $S \longrightarrow \hom(\mathbf{1}, X)$ for some $X : \mathcal{S}$. Noting that $X^{\mathbf{1}} \cong X$ and $S^{\mathbf{0}} \cong \mathbf{1}$, we see that X^U is represented by the map $\mathsf{id} : \hom(\mathbf{1}, X) \longrightarrow \hom(\mathbf{1}, X)$.

In other words, the effect of hypothesizing $\mathbf{syn}$ is to erase the portion of X which comes from $\mathbf{Set}$ and leave only the $\mathcal{S}$ component. ◇

Notation 4.2.13. We will often work with an element of $\bigcirc\mathcal{U}$ (recall that $\mathcal{U}$ is the universe of small types) and will need a convenient syntax for the dependent modality $\hat{\bigcirc} : \bigcirc\mathcal{U} \to \mathcal{U}$. We will take advantage of the fact that this particular dependent modality can be realized by an ordinary dependent product and write $\bigcirc_z A\, z$ as shorthand for $(z : \mathbf{syn}) \to A\, z$.

Capitalizing on the idea that terms and types in a context with $z : \mathbf{syn}$ correspond to terms and types in $\mathcal{S}$, we import a suite of contexts internalizing the ML structure of $(\mathcal{S}, \tau_{\mathcal{S}})$ for use within our type theory. For the sake of concision, we shall specify only dependent products and booleans.

Extension 4.2.2. *Under the assumption $z : \mathbf{syn}$, there exist types* $\mathsf{Ty} : \mathcal{U}$ *and* $\mathsf{Tm} : \mathsf{Ty} \to \mathcal{U}$. *We further assume constants structure* $(\mathsf{Ty}, \mathsf{Tm})$ *as a universe closed under various connectives:*

$$\begin{aligned}
&\mathsf{Prod} : (A : \mathsf{Ty}) \to (\mathsf{Tm}(A) \to \mathsf{Ty}) \to \mathsf{Ty}\\
&\alpha_{\mathsf{Prod}} : (A : \mathsf{Ty})(B : \mathsf{Tm}(A) \to \mathsf{Ty}) \to ((a : \mathsf{Tm}\, A) \to \mathsf{Tm}(B\, a)) \cong \mathsf{Tm}(\mathsf{Prod}\, A\, B)\\
&\mathsf{Bool} : \mathsf{Ty}\\
&\mathsf{true}, \mathsf{false} : \mathsf{Tm}(\mathsf{Bool})\\
&\mathsf{if} : (B : \mathsf{Tm}\,\mathsf{Bool} \to \mathsf{Ty}) \to \mathsf{Tm}(B\,\mathsf{true}) \to \mathsf{Tm}(B\,\mathsf{false}) \to (b : \mathsf{Tm}\,\mathsf{Bool}) \to \mathsf{Tm}(B\, b)\\
&\_ : (B : \mathsf{Tm}\,\mathsf{Bool} \to \mathsf{Ty})(N_0 : \mathsf{Tm}(B\,\mathsf{true}))(N_1 : \mathsf{Tm}(B\,\mathsf{false})) \to \mathsf{if}\, B\, N_0\, N_1\, \mathsf{true} = N_0\\
&\_ : (B : \mathsf{Tm}\,\mathsf{Bool} \to \mathsf{Ty})(N_0 : \mathsf{Tm}(B\,\mathsf{true}))(N_1 : \mathsf{Tm}(B\,\mathsf{false})) \to \mathsf{if}\, B\, N_0\, N_1\, \mathsf{false} = N_1
\end{aligned}$$

Suppose we are given a type A (corresponding to some object $X : \mathcal{G}$) and an element $N : \mathbf{syn} \to A$ (correspond to a morphism f into $\pi(X)$). Many of our tasks can be rephrased as needing to *lift* f along π. We can concisely capture this within type theory by requesting an element $M : A$ such that $(z : \mathbf{syn}) \to M = N(z)$. In fact, statements of this form are so common we will introduce notation for them:

$$\{A \mid z : \mathbf{syn} \mapsto N\} = \left(\textstyle\sum_{M:A} (z : \mathbf{syn}) \to M = N(z)\right)$$

Notation 4.2.14. The notation $\{A \mid z : \mathbf{syn} \mapsto N\}$ comes from Cohen et al. [Coh+17] where they are used to specify the boundaries of (hyper)cubes. Following op. cit., we will refer to the constraint $(z : \mathbf{syn}) \to M = N(z)$ as a *boundary constraint.*

We will also have occasion to use the *closed* modality $\bullet$ associated with $\bigcirc$:

Extension 4.2.3. *We postulate an operator* $\bullet : \mathcal{U} \to \mathcal{U}$ *such that* $\bullet A$ *satisfies the mapping out property of the pushout* $A \coprod_{A \times \mathbf{syn}} \mathbf{syn}$. *Explicitly, to construct* $f : \bullet A \to B$ *it suffices to construct a map* $f_0 : A \to B$ *and* $f_1 : \mathbf{syn} \to B$ *such that whenever we are given* $a : A$ *and* $z : \mathbf{syn}$, *we have* $f_0\, a = f_1\, z$.

We could have equivalently characterized $\bullet$ as the unique idempotent lex monad such that $\bigcirc\bullet A = \mathsf{Unit}$ and $A = \bigcirc A \times_{\bullet\bigcirc A} \bullet A$. However, the mapping out property as a pushout will be of some use.

We also will require an internalized form of *realignment* (Lemma 3.3.10), which allows us to replace an up-to-isomorphism solution to an extension problem with a strict solution:

Extension 4.2.4. *Each universe satisfies an internal form of realignment [OP18] so that, in particular, the following map has a section* re *for any* $B : \mathcal{U}$*:*

$$\left(\sum_{A:\mathcal{U}} A \cong B\right) \to \left(\sum_{A:\mathbf{syn}\to\mathcal{U}} (z : \mathbf{syn}) \to A(z) \cong B\right)$$

Remark 4.2.15. Explicitly, re allows us to take a type B and a partially defined type $A_0 : \bigcirc\mathcal{U}$ which are isomorphic when both are defined via $\alpha_0 : (z : \mathbf{syn}) \to A_0\, z \cong B$ and obtain $A : \mathcal{U}$ and $\alpha : A \cong B$ such that $(z : \mathbf{syn}) \to A_0\, z = A$ and $(z : \mathbf{syn}) \to \alpha_0\, z = \alpha$.

In practice, realignment is used in a situation where we are confronted by the problem of extending the partially defined A_0 to some total type. We break this process into two stages by first constructing an extension *up to isomorphism*—B and α_0—and then use realignment to parlay this up-to-isomorphism solution into a genuine extension. ◇

We shall refer to extensional type theory supplemented with all of these extensions as *the language of synthetic Tait computability.* We may now phrase the core of the construction of the displayed ML category purely in terms of this language (once again limiting ourselves to dependent products and booleans for the sake of concision):

Lemma 4.2.16 (Fundamental lemma)**.** *There exists* $\mathsf{Ty}^* : \{\mathcal{U} \mid z : \mathbf{syn} \mapsto \mathsf{Ty}\, z\}$ *and* $\mathsf{Tm}^* : \{\mathsf{Ty}^* \to \mathcal{U} \mid z : \mathbf{syn} \mapsto \mathsf{Tm}\, z\}$ *along with the following constants:*

$$\begin{aligned}
&\mathsf{Prod}^* : (A : \mathsf{Ty}^*) \to (B : \mathsf{Tm}^*(A) \to \mathsf{Ty}^*) \to \{\mathsf{Ty}^* \mid z : \mathbf{syn} \mapsto \mathsf{Prod}\, z\, A\, B\}\\
&\alpha_{\mathsf{Prod}^*} : (A : \mathsf{Ty}^*)(B : \mathsf{Tm}^*(A) \to \mathsf{Ty}^*)\\
&\quad \to \{((a : \mathsf{Tm}^*\, A) \to \mathsf{Tm}^*(B\, a)) \cong \mathsf{Tm}^*(\mathsf{Prod}^*\, A\, B) \mid z : \mathbf{syn} \mapsto \alpha_{\mathsf{Prod}}\, z\, A\, B\}\\
&\mathsf{Bool}^* : \mathsf{Ty}^*\\
&\mathsf{true}^*, \mathsf{false}^* : \mathsf{Tm}^*(\mathsf{Bool}^*)\\
&\mathsf{if}^* : (B : \mathsf{Tm}^*\, \mathsf{Bool}^* \to \mathsf{Ty}^*)(N_0 : \mathsf{Tm}^*(B\, \mathsf{true}^*))(N_1 : \mathsf{Tm}^*(B\, \mathsf{false}^*))(M : \mathsf{Tm}^*\, \mathsf{Bool}^*)\\
&\quad \to \{\mathsf{Tm}^*(B\, M) \mid z : \mathbf{syn} \mapsto \mathsf{if}\, z\, B\, N_0\, N_1\, M\}\\
&\_ : (B : \mathsf{Tm}^*\, \mathsf{Bool}^* \to \mathsf{Ty}^*)(N_0 : \mathsf{Tm}^*(B\, \mathsf{true}^*))(N_1 : \mathsf{Tm}^*(B\, \mathsf{false}^*))\\
&\quad \to \mathsf{if}^*\, B\, N_0\, N_1\, \mathsf{true}^* = N_0\\
&\_ : (B : \mathsf{Tm}^*\, \mathsf{Bool}^* \to \mathsf{Ty}^*)(N_0 : \mathsf{Tm}^*(B\, \mathsf{true}^*))(N_1 : \mathsf{Tm}^*(B\, \mathsf{false}^*))\\
&\quad \to \mathsf{if}^*\, B\, N_0\, N_1\, \mathsf{false}^* = N_1
\end{aligned}$$

Remark 4.2.17. Note that the boundary conditions on some constants are necessary to make those of other constants well-typed. For instance, without the boundary conditions on Tm^* and Prod^* the boundary of α_{Prod^*} would be ill-typed. ◇

In essence, the existence of these constants upgrades $\mathcal{G}$ to an ML category while the requirement that each constant match its corresponding version from $\mathcal{S}$ ensures that π is a morphism of ML categories. Thus, this lemma plays the same role as the *fundamental lemma of logical relations* and constitutes the heart of our proof.

4.3 The canonicity model

We now set about proving Lemma 4.2.16. To this end, we begin by constructing Ty^*. Recall our earlier intuition all the way back in Lemma 4.1.1: a type in our displayed model should consist of an element $A : \mathsf{Ty}$ along with a type whose elements are pairs of $M : \mathsf{Tm}\,A$ together with a proof that M is canonical. Informally, the latter type is the *total space* of the predicate isolating canonical elements of A.

More formally, we will begin with the following putative definition:

$$\begin{array}{l}\textbf{record}\ \Phi : \mathcal{U}_2\ \textbf{where}\\ \quad \mathsf{tp} : \bigcirc_z \mathsf{Ty}\,z\\ \quad \mathsf{pred} : \{\mathcal{U}_1 \mid z : \textbf{syn} \mapsto \mathsf{Tm}\,z\,A\}\end{array}$$

Notation 4.3.1. Here and elsewhere we will avail ourselves of `Agda`-style records. We will use the standard "dot" notation to project out fields e.g., $\phi.\mathsf{tp}$ and use copattern notation to construct elements of Φ.

Now the strong constraints we have placed upon pred ensures that after assuming $z : \textbf{syn}$ we have the following chain of isomorphisms:

$$\begin{aligned}&\Phi\\ &\qquad \textit{(Passing from a record to a dependent sum.)}\\ &\cong \textstyle\sum_{\mathsf{tp}:\bigcirc_z \mathsf{Ty}\,z}\{\mathcal{U}_1 \mid z : \textbf{syn} \mapsto \mathsf{Tm}\,z\,A\}\\ &\qquad \textit{(Taking advantage of the fact that } \bigcirc \textit{ is idempotent.)}\\ &\cong \textstyle\sum_{\mathsf{tp}:\mathsf{Ty}\,z}\{\mathcal{U}_1 \mid z : \textbf{syn} \mapsto \mathsf{Tm}\,z\,A\}\\ &\qquad \textit{(Having assumed } z : \textbf{syn} \textit{ the second component is fixed.)}\\ &\cong \textstyle\sum_{\mathsf{tp}:\mathsf{Ty}\,z} \mathsf{Unit}\\ &\cong \mathsf{Ty}\,z\end{aligned}$$

We denote the composite of these isomorphisms ϕ.

Revisiting the statement of Lemma 4.2.16 however, this is not enough. Whatever we choose for Ty^* must satisfy $(z : \textbf{syn}) \to \mathsf{Ty}^* = \mathsf{Ty}\,z$ and Φ satisfies this only up to isomorphism. Fortunately, this can be rectified *automatically* using realignment. Accordingly, we define Ty^* as follows:

$$(\mathsf{Ty}^*, \phi^*) = \mathsf{re}(\Phi, \phi) \tag{4.1}$$

By definition, Ty^* comes equipped with an isomorphism $\phi^* : \Phi \cong \mathsf{Ty}^*$ which extends ϕ. We will take advantage of this isomorphism to construct elements of Ty^* using record-style copattern notation (specifying tp and pred) and project out these fields using dot notation. The condition that ϕ^* extends ϕ amounts to the following (where $A : \mathsf{Ty}^*$):

$$(z : \textbf{syn}) \to A = A.\mathsf{tp}\,z \tag{4.2}$$

In fact, our construction of $\mathcal{T}*$ leads us nearly immediately to the correct definition of $\mathsf{Tm}^* : \mathsf{Ty}^* \to \mathcal{U}$. This must assign each $A : \mathsf{Ty}^*$ to an element of $\mathcal{U}$ subject to the constraint that if $z : \textbf{syn}$ then $\mathsf{Tm}^*\,A = \mathsf{Tm}\,z\,A$. Inspecting the definition of Ty^*, we see that A comes equipped with just such a type:

$$\mathsf{Tm}^*\,A = A.\mathsf{pred} \tag{4.3}$$

Two tasks remain: we must close $(\mathsf{Ty}^*, \mathsf{Tm}^*)$ under dependent products and booleans. Somewhat counter-intuitively, dependent products are simpler so we begin with them.

Lemma 4.3.2. $(\mathsf{Ty}^*, \mathsf{Tm}^*)$ *is closed under dependent products.*

Proof. Unfolding the statement of this lemma, we must construct terms of the following types:

$$\begin{aligned}
&\mathsf{Prod}^* : (A : \mathsf{Ty}^*) \to (B : \mathsf{Tm}^*(A) \to \mathsf{Ty}^*) \to \{\mathsf{Ty}^* \mid z : \mathbf{syn} \mapsto \mathsf{Prod}\, z\, A\, B\}\\
&\alpha_{\mathsf{Prod}^*} : (A : \mathsf{Ty}^*)(B : \mathsf{Tm}^*(A) \to \mathsf{Ty}^*)\\
&\qquad \to \{((a : \mathsf{Tm}^*\, A) \to \mathsf{Tm}^*(B\, a)) \cong \mathsf{Tm}^*(\mathsf{Prod}^*\, A\, B) \mid z : \mathbf{syn} \mapsto \alpha_{\mathsf{Prod}}\, z\, A\, B\}
\end{aligned}$$

We will begin with Prod^* and therefore fix $A : \mathsf{Ty}^*$ and $B : \mathsf{Tm}^*\, A \to \mathsf{Ty}^*$. As mentioned above, we it suffices to specify $(\mathsf{Prod}^*\, A\, B).\mathsf{tp}$ and $(\mathsf{Prod}^*\, A\, B).\mathsf{pred}$. Inspecting the constraints on Prod^* along with Eq. (4.2), we notice that tp is already fully determined: we must set $(\mathsf{Prod}^*\, A\, B).\mathsf{tp} = \lambda z.\, \mathsf{Prod}\, z\, A\, B$ to satisfy the relevant boundary constraint. It remains to define pred whose type is now refined to the following:

$$(\mathsf{Prod}^*\, A\, B).\mathsf{pred} : \{\mathfrak{U} \mid z : \mathbf{syn} \mapsto \mathsf{Tm}\, z\, (\mathsf{Prod}\, A\, B)\}$$

Just as with $\mathfrak{T}*$, we will begin by defining a type Φ which satisfies the boundary condition only up to isomorphism and then use realignment to rectify the situation. With an eye towards defining α_{Prod^*}, we know that $(\mathsf{Prod}^*\, A\, B).\mathsf{pred}$ must be isomorphic to $\Phi = (a : \mathsf{Tm}^*\, A) \to \mathsf{Tm}^*(B\, a)$. Let us now observe that α_{Prod} is precisely the required isomorphism $\Phi \cong \mathsf{Tm}\, z(\mathsf{Prod}\, A\, B)$ after utilizing the boundary conditions on Tm^*.

All told, we make the following definition:

$$((\mathsf{Prod}^*\, A\, B).\mathsf{pred}, \alpha_{\mathsf{Prod}^*}) = \mathsf{re}((a : \mathsf{Tm}^*\, A) \to \mathsf{Tm}^*(B\, a), \alpha_{\mathsf{Prod}}) \tag{4.4}$$

Remarkably, the isomorphism generated by re satisfies exactly the boundary condition for α_{Prod^*} and so no additional work is required to define it. □

In a certain sense, what made the construction of dependent products so efficient was the presence of the η law. This enabled us to bundle up the introduction, elimination, β, and η rules into a single object. Moreover, such an isomorphism fully constrained the definition o f Prod^*—at least up to isomorphism. The lack of an η law for Bool^*, by contrast, will result in both some additional manual labor and creativity.

Lemma 4.3.3. $(\mathsf{Ty}^*, \mathsf{Tm}^*)$ *is closed under booleans.*

Proof. Unfolding, we have the following list of constants to define:

$$\begin{aligned}
&\mathsf{Bool}^* : \mathsf{Ty}^*\\
&\mathsf{true}^*, \mathsf{false}^* : \mathsf{Tm}^*(\mathsf{Bool}^*)\\
&\mathsf{if}^* : (B : \mathsf{Tm}^*\, \mathsf{Bool}^* \to \mathsf{Ty}^*)(N_0 : \mathsf{Tm}^*(B\, \mathsf{true}^*))(N_1 : \mathsf{Tm}^*(B\, \mathsf{false}^*))(M : \mathsf{Tm}^*\, \mathsf{Bool}^*)\\
&\qquad \to \{\mathsf{Tm}^*(B\, M) \mid z : \mathbf{syn} \mapsto \mathsf{if}\, z\, B\, N_0\, N_1\, M\}\\
&\_ : (B : \mathsf{Tm}^*\, \mathsf{Bool}^* \to \mathsf{Ty}^*)(N_0 : \mathsf{Tm}^*(B\, \mathsf{true}^*))(N_1 : \mathsf{Tm}^*(B\, \mathsf{false}^*))\\
&\qquad \to \mathsf{if}^*\, B\, N_0\, N_1\, \mathsf{true}^* = N_0\\
&\_ : (B : \mathsf{Tm}^*\, \mathsf{Bool}^* \to \mathsf{Ty}^*)(N_0 : \mathsf{Tm}^*(B\, \mathsf{true}^*))(N_1 : \mathsf{Tm}^*(B\, \mathsf{false}^*))\\
&\qquad \to \mathsf{if}^*\, B\, N_0\, N_1\, \mathsf{false}^* = N_1
\end{aligned}$$

We begin with Bool^*. As before, we have two fields to define—$.\mathsf{tp}$ and $.\mathsf{pred}$—and the first is fully constrained by the boundary constraint:

$$\mathsf{Bool}^*.\mathsf{tp} = \mathsf{Bool}$$

Unlike dependent products, there is no isomorphism fully constraining $\mathsf{Bool}^*.\mathsf{pred}$ so we have the freedom to choose several different definitions. At this point we return to our earlier intuition for $.\mathsf{pred}$: it should be a type consisting of pairs of a type $M : \mathsf{Tm}\,\mathsf{Bool}$ along with a proof that M is canonical. From this and with an eye towards the boundary constraint, we are led to the following intermediate definition:

$$\Phi = \textstyle\sum_{b : \bigcirc_z \mathsf{Tm}\, z\, \mathsf{Bool}} \bullet(b = \mathsf{true} + b = \mathsf{false})$$

Notice the appearance of the closed modality here. It is necessary to ensure the existence of the following isomorphism under the assumption $z : \mathbf{syn}$:

$$\begin{aligned}
\psi : \Phi &\cong \textstyle\sum_{b : \bigcirc_z \mathsf{Tm}\, z\, \mathsf{Bool}} \bullet(b = \mathsf{true} + b = \mathsf{false}) \\
&\cong \textstyle\sum_{b : \mathsf{Tm}\, z\, \mathsf{Bool}} \bullet(b = \mathsf{true} + b = \mathsf{false}) \\
&\cong \textstyle\sum_{b : \mathsf{Tm}\, z\, \mathsf{Bool}} \mathsf{Unit} && \text{Recall } \bigcirc\bullet A \cong \mathsf{Unit} \\
&\cong \mathsf{Tm}\, z\, \mathsf{Bool}
\end{aligned}$$

This isomorphism is precisely what we required as input for realignment:

$$(\mathsf{Bool}^*.\mathsf{pred}, \alpha) = \mathsf{re}(\Phi, \psi) \tag{4.5}$$

The definition of $\mathsf{Bool}^*.\mathsf{pred}$ ensures that it satisfies the relevant boundary condition. It remains to handle the introduction and elimination rules. For the introduction rule we use the isomorphism α to construct the required elements of $\mathsf{Bool}^*.\mathsf{pred}$:

$$\mathsf{true}^* = \alpha(\mathsf{true}, \eta(\mathsf{in}_1(\star))) \tag{4.6}$$

$$\mathsf{false}^* = \alpha(\mathsf{false}, \eta(\mathsf{in}_2(\star))) \tag{4.7}$$

The boundary conditions e.g., $(z : \mathbf{syn}) \to \mathsf{true}^* = \mathsf{true}\, z$ follow from the assumption that α extends ψ. We leave the details to the reader.

Finally, it remains to define if^*. This is an exercise in programming with the elimination principle for $\bullet X = X \coprod_{X \times \mathbf{syn}} \mathbf{syn}$:

$$\mathsf{if}^*\, B\, N_0\, N_1\, M = \begin{cases} \mathsf{if}\, z\, B\, N_0\, N_1\, M_0 & M = \alpha(M_0, \mathsf{in}_2(z)) \\ N_0 & M = \alpha(M_0, \mathsf{in}_1(\mathsf{in}_1(*))) \\ N_1 & M = \alpha(M_0, \mathsf{in}_1(\mathsf{in}_2(*))) \end{cases} \tag{4.8}$$

Informally, the three cases split on which of the three states $\bullet(b = \mathsf{true} + \mathsf{false})$ occupies: either $\mathbf{syn}$ holds, b is a true, or b is false. As $\bullet\ldots$ is a pushout, we must check that the results of each of the three branches match as they are supposed to, but this is an immediate consequence of boundary conditions along with the computation rules for if. A similarly direct computation shows that if^* satisfies the two required equations, completing the proof. □

Proof of Lemma 4.2.16. The fundamental lemma is now an immediate consequence of the results of this section. □

Corollary 4.3.4. *Write* $\tau_{\mathcal{G}} : \mathcal{T}^{\bullet}_{\mathcal{G}} \longrightarrow \mathcal{T}_{\mathcal{G}}$ *for the family in* $\mathcal{G}$ *denoted by* Tm^**, then* $(\mathcal{G}, \tau_{\mathcal{G}})$ *is an ML category and* π *is a morphism of ML categories.*

4.4 The canonicity theorem

We are now, finally, in a position to prove the main result of this chapter.

Proof of Theorem 4.0.1. Fix $\mathbf{1} \vdash M : \mathsf{Bool}$. We wish to show that either $M = \mathsf{tt}$ or $M = \mathsf{ff}$. To this end, let us consider $\lfloor M \rfloor : \mathbf{1} = \mathbf{y}(\mathbf{1}) \longrightarrow \mathcal{T}^{\bullet}_{\mathcal{S}}$ the interpretation of M into the syntactic model $\mathcal{S}$. Applying Theorem 4.2.6 and Corollary 4.3.4, we conclude that there exists a morphism $M^* : \mathbf{1} \longrightarrow \mathcal{T}^{\bullet}_{\mathcal{G}}$ such that $\pi(M^*) = M$.

Inspecting the proof of Theorem 4.2.6, we observe that $\tau_{\mathcal{G}} \circ M^* : \mathbf{1} \longrightarrow \mathcal{T}_{\mathcal{G}}$ is the interpretation of Bool^*. We may unfold the interpretation of Bool^* to conclude that a map $\mathbf{1} \longrightarrow [\![\mathsf{Bool}^*]\!]^*\mathcal{T}^{\bullet}_{\mathcal{G}}$ is determined by a pair of (1) a map $N : \mathbf{1} \longrightarrow [\![\mathsf{Bool}]\!]^*\mathcal{T}^{\bullet}_{\mathcal{S}}$ in $\mathcal{S}$ and (2) a proof that either $N = \mathsf{true}$ or $N = \mathsf{false}$ (more properly, a map $\mathbf{1} \longrightarrow N = \mathsf{true} + N = \mathsf{false}$ in $\mathbf{Set}$). Combining this with the fact that $\pi(M^*) = M$, we see that that M^* determines a proof $M = \mathsf{true}$ or $M = \mathsf{false}$ exactly as required. □

Remark 4.4.1. While we have not belabored the point, all of the machinery here (the various categories, the models of type theory, etc.) is constructively valid. Accordingly, this proof of canonicity can be carried out in e.g., the effective topos where it would yield an evaluator for closed booleans. ◇

Part II

Modalities in type theory

5 Towards multimodal type theory

Nothing is more arbitrary than a modal logic: "I am done with this logic, may I have another one?" seems to be the motto of modal logicians.

Jean-Yves Girard
The Blind Spot

5.1 A motivating example

In Chapters 2 and 3 we introduced type theories as formal systems comprised of type and term operators invariant under substitution. It is this invariance property that gives type theory its particular flavor but, unfortunately, there are many useful type constructors which are not stable under substitution.

Consider, for instance, the work by Orton and Pitts [OP18] analyzing the semantics of cubical type theory in a presheaf topos $\mathbf{PSh}(\mathcal{C})$ using the rich internal language available in $\mathbf{PSh}(\mathcal{C})$. The aim of this work was to simplify the task of constructing a model of cubical type theory and, in particular, to make it easier to handle the most complex portion of cubical type theory: its novel handling of the identity type. In cubical type theory, each type comes equipped with a set of operations for manipulating the identity type (the Kan composition operations) and these can be quite complex. Any model must equip each type with a Kan composition structure, a highly technical task.

The internal approach in op. cit., however, allows these Kan operations to be restructured as programming exercises that can be specified concisely. In fact, a major reason for this advantage is the fiberwise nature of the internal language; when working externally the necessity of working and passing between various fibers obscures the essence of the composition operators. While this approach to semantics is quite appealing, Orton and Pitts stop short of defining a univalent universe equipped with a Kan composition structure. Indeed, defining such a universe proved to be a substantial undertaking and was accomplished in subsequent work by Licata et al. [Lic+18].

In order to define a universe, Licata et al. [Lic+18] took advantage of a certain *amazing right adjoint* within the category $\mathbf{PSh}(\mathcal{C})$. Exponentiation by the interval object $\mathbb{I}$ was a left adjoint with $(-)^{\mathbb{I}} \dashv (-)_{\mathbb{I}}$. Clever application of $(-)_{\mathbb{I}}$ allowed Licata et al. to define their universe.

One might imagine that all that is required is to extend the extensional type theory serving as the internal language of $\mathbf{PSh}(\mathcal{C})$ with a type-constructor representing this right adjoint. Unfortunately, this is not possible:

Theorem 5.1.1 (Theorem 5.1 [Lic+18])**.** *Given a type constructor* $(-)_{\mathbb{I}} : \mathcal{U} \to \mathcal{U}$ *and a natural isomorphism* $A^{\mathbb{I}} \to B \cong A \to B_{\mathbb{I}}$*, we must have* $\mathbb{I} \cong \mathbf{1}$*.*

This result may seem surprising; semantically $-^{\mathbb{I}}$ does have a right adjoint and $\mathbb{I}$ is certainly not isomorphic to $\mathbf{1}$. What has gone wrong? The issue lies within the hidden strength of asking the right adjoint $(-)_{\mathbb{I}}$ to form a type operator $\mathcal{U} \to \mathcal{U}$. Unraveling the semantics of type theory within $\mathbf{PSh}(\mathcal{C})$, this entails having not only a functor sending small objects to small objects but having such a functor on each slice in a manner compatible with pullback. Examining $\mathbf{PSh}(\mathcal{C})$, it is apparent that $-_{\mathbb{I}}$ cannot be defined coherently on each slice and the bijection $\hom(A^{\mathbb{I}}, B) \cong \hom(A, B_{\mathbb{I}})$ does not extend to an isomorphism of exponential objects. To summarize, we cannot axiomatize $(-)_{\mathbb{I}}$ nor its property as a right adjoint within type theory because neither respects substitution. This is particularly unfortunate within the context of Licata et al. [Lic+18]; the thesis of the work was that working within the internal language was an effective way to construct models of cubical type theory. This leads to an impasse: can one maintain the convenience of working internally while still having access to the amazing right adjoint?

To resolve this dilemma and carry out the construction of universes, Licata et al. [Lic+18] modify the underlying type theory to include a *modality*: a mapping of types to types which need not be stable under substitution.

Let us revisit the the external adjunction $(-)^{\mathbb{I}} \dashv (-)_{\mathbb{I}}$. We cannot extend the bijection of hom-sets to an isomorphism of exponentials but we can (tautologically) rephrase it as a bijection between the global points of the exponential objects $X^{Y^{\mathbb{I}}}$ and $(X_{\mathbb{I}})^{Y}$. Within any presheaf topos, we can organize the global points of a presheaf X into a presheaf themselves. We have an adjunction $\Delta \dashv \Gamma : \mathbf{PSh}(\mathcal{C}) \longrightarrow \mathbf{Set}$ and the comonad $\square = \Delta \circ \Gamma$ induced by this adjunction sends a presheaf to the (discrete) presheaf of its global points $\square X$. We may therefore actually phrase our bijection as an isomorphism:

$$\alpha : \square X^{Y^{\mathbb{I}}} \cong \square (X_{\mathbb{I}})^{Y}$$

Similarly, while $(-)_{\mathbb{I}}$ does not induce a map $\mathcal{U} \to \mathcal{U}$, it does induce a map sending the global points of $\mathcal{U}$ and the global points of $\mathcal{U}$:[1]

$$(-)_{\mathbb{I}} : \square \mathcal{U}^{\square \mathcal{U}}$$

If we could internalize $\square$, therefore, we could add these constants to our theory to circumvent Theorem 5.1.1. Unsurprisingly, $\square$ suffers the same issues as $(-)_{\mathbb{I}}$ when it comes to internalizing it within type theory; it does not extend to each slice category in a coherent manner. In order to internalize it, we therefore must modify the type theory to permit such type operators.

Concretely, Licata et al. [Lic+18] work within *crisp type theory*, a variant of Martin-Löf type theory equipped with two classes of variables: normal and crisp. Crisp variables may be used wherever normal variable are used but the reverse is not true. The type theory features a new type constructor $\square -$ and elements of $\square A$ consist of elements of

[1] As a closed type, $\mathcal{U}$ is interpreted by an object of $\mathbf{PSh}(\mathcal{C})$

A constructed using only crisp variables. In order to ensure that this is well-behaved, various careful modifications must be made to the structures of contexts and substitutions (allowing e.g., only crisp variables to replace crisp variables).

One can construct a model of crisp type theory within $\mathbf{PSh}(\mathcal{C})$ which realizes $[\![\Box A]\!]$ as essentially $\Box[\![A]\!]$ and therefore safely add the postulates described above. Crucially, pivoting to crisp type theory permitted Licata et al. a sufficiently flexible internal language to describe $(-)_{\mathbb{I}}$. In particular, it was necessary to have an internal language which violated the core principle of invariance under substitution but which retained much of the character of type theory.

5.1.1 Multiple interacting modalities

Let us consider an entirely separate motivation for the $\Box$ modality: guarded recursion and guarded domain theory. A centerpiece of programming language theory is the study of solutions to so-called domain equations: fixed points to functors $F : \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \longrightarrow \mathcal{C}$. Many programming languages can be modeled only in categories permitting solutions a certain domain equation such as $F(X, Y) = Y^X$ and the existence of solutions to a large class of domain equations has been established within various categories of domains [SP82]. Unfortunately, some of the domain equations which arise in practice (particularly around higher-order store) do not admit solutions in classical domain theory. This has spawned the subfield of guarded domain theory [Bir+12], centering around $\mathbf{PSh}(\omega)$ and the *later* functor given by the following formula:

$$(\blacktriangleright X)(0) = \mathbf{1} \qquad (\blacktriangleright X)(n+1) = X(n)$$

We note that there exists a natural transformation $\mathsf{next} : \mathsf{id} \longrightarrow \blacktriangleright$.

Intuitively, $\blacktriangleright X$ serves as a version of X *delayed by a step* such that its value cannot be used unless it is to construct something else under the $\blacktriangleright$ functor. In ultrametric realizations of domain theory [AR89], $\blacktriangleright$ models the contraction functor $\frac{1}{2} \cdot -$. The force of the later modality is captured by the *Löb induction* principle:

$$\mathsf{loeb} : X^{\blacktriangleright X} \longrightarrow X \qquad \mathsf{loeb} \circ f = \epsilon \circ \langle f, \mathsf{loeb} \circ \mathsf{next} \circ f \rangle$$

Moreover, many of the domain equations which could not be solved over classical domains admit solutions within *guarded domains*.

Definition 5.1.2. A guarded domain is a presheaf $X : \mathbf{PSh}(\omega)$ equipped with a structure map $\theta_X : \blacktriangleright X \longrightarrow X$.

Working with guarded domains over traditional domains also satisfies an important desideratum of synthetic domain theory generally: $\mathbf{PSh}(\omega)$ is a topos. It therefore hosts a rich internal language that we may use to reason about guarded domains. For instance, the structure map θ_X combines with Löb induction to give a fixed-point combinator on X, expressed as follows within the internal language:

$$\mathsf{fix}\, f = \mathsf{loeb}(\lambda x.\, f(\theta_X\, x))$$

Another remarkable fact is that the universe of small types in $\mathbf{PSh}(\omega)$ is itself a guarded domain. This allows for much of the theory of domain equations to be replaced by the simpler theory of term-level fixed points.

In order to work effectively with guarded domains in the internal language, it is necessary to internalize $\blacktriangleright$. Fortunately, this is a far easier task than internalizing $\square$. In particular, there is a natural transformation $\mathsf{next} : \mathsf{id} \longrightarrow \blacktriangleright$ and we therefore extend $\blacktriangleright$ to slice categories of $\mathbf{PSh}(\omega)$ as follows:

$$\begin{tikzcd}
\blacktriangleright_Y X \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \blacktriangleright X \arrow[d] \\
Y \arrow[r] & \blacktriangleright Y
\end{tikzcd} \tag{5.1}$$

This functor gives a type operator $\blacktriangleright : \mathcal{U} \longrightarrow \mathcal{U}$ within the internal language along with a natural transformation $\mathsf{next} : A \to \blacktriangleright A$. The structure map witnessing that the universe is a guarded domain is realized by the existence of a *dependent* $\blacktriangleright$: modality:

$$\begin{tikzcd}
 & \blacktriangleright(\sum_{A:\mathcal{U}} \mathsf{El}(A)) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \sum_{A:\mathcal{U}} \mathsf{El}(A) \arrow[d] \\
\mathcal{U} \arrow[r, "\mathsf{next}"] \arrow[rr, bend right, "\blacktriangleright"'] & \blacktriangleright\mathcal{U} \arrow[r, "\hat{\blacktriangleright}"] & \mathcal{U}
\end{tikzcd}$$

This situation is near the ideal; it is quite possible to work internally to $\mathbf{PSh}(\omega)$ to give concise arguments in guarded domain theory and denotational semantics based upon it [BBM14; KMV22; MP16; MV19; PMB15]. We refer to type theory extended with $\blacktriangleright$, $\hat{\blacktriangleright}$, and loeb as *guarded type theory*.

Certain properties in guarded domain theory, however, cannot be internalized within this framework. A crucial difference between guarded domain theory and classical domain theory is the apparent *intensionality* of the former and frequently the 'denotational semantics' in guarded domain theory do not satisfy the expected equations on the nose. The source of failure can typically be traced to the inequality of id and $\theta_X \circ \mathsf{next}$. To correct for this mismatch, it is common to define a notion of *weak bisimilarity* in guarded domain theory [MP16; PMB15] which essentially equates elements up to θ_X. While one can define a notion of weak bisimilarity internally, it is frequently necessary to consider it externally in order to show e.g., that two terms are not weakly bisimilar. More generally, external reasoning is frequently necessary to even state certain properties e.g. termination, productivity, or other expressions of *adequacy*. In these circumstances, a richer internal language is required.

It is for this purpose that various flavors of guarded type theory are extended with a second modality: the familiar $\square$ [Biz+16; Clo+15; KMV22]. We draw attention to an important isomorphism available within $\mathbf{PSh}(\omega)$:

$$\square X \cong \square\blacktriangleright X \tag{5.2}$$

Adding $\square$ to guarded type theory together with the isomorphism $\square\blacktriangleright A \cong \square A$ enables a plethora of new constructions. Aside from merely being able to reason about global behavior, it becomes possible to define coinductive data structures from their guarded counterpart. Various extensions to this system have culminated in clocked type theory [AM13; BGM17; BM15].

As previously discussed, $\square$ cannot be internalized as easily as $\blacktriangleright$. A priori, it seems plausible that crisp type theory could be stretched to accommodate $\blacktriangleright$ but this ad-hoc solution does not provide a satisfactory account of Eq. (5.2). Indeed, crisp type theory works well only when considering one modality and when that particular modality happens to be an idempotent comonad.

While idempotent comonads may arise with some frequency, they are far from the only modality worth considering. Similar but subtlety different modalities are considered in the context of distributive programming [Mur08], staged programming [DP01; HPS22], information flow [Kav19], axiomatic cohesion [SS12; Shu18], normalization-by-gluing [BKS21], relational parametricity [Cav21; Nuy20], and many others. Many of these situations demand certain laws like Eq. (5.2) forcing modalities to interact in specific ways.

In this chapter, we survey the current landscape of type theories incorporating one or more modalities to motivate and situate our proposed general modal type theory: MTT [Gra+20a].

5.2 Fibered modalities

Before discussing various approaches for integrating modalities which do not internalize easily, we begin by discussing the best possible situation for modalities: those which do commute with substitution and therefore require no contortions to include within type theory. These are the modalities which we can restrict to act *fiberwise* and we refer to them as *fibered modalities*.

Syntactically, a fibered modality admits a simple description:

Definition 5.2.1. Within type theory, a fibered modality is a type constructor $\bigcirc$ with the following formation rule:

$$\frac{\Gamma \vdash A\ \mathsf{type}}{\Gamma \vdash \bigcirc A\ \mathsf{type}} \tag{5.3}$$

In order to broaden type theory to include non-fibered modalities, it is helpful to consider what properties fibered modalities have semantically. To this end, consider a category with display maps $(\mathcal{C}, \mathscr{D})$. The class of well-behaved (specifically, left exact) fibered modalities in internal language of $\mathcal{C}$ most commonly arise from functors F equipped with a point $\eta : \mathsf{id} \longrightarrow F$ which preserves display maps, the terminal object, and cartesian squares of display maps. Define $F_{\mathscr{D}} : \mathscr{D}^{\mathsf{cart}} \longrightarrow \mathscr{D}^{\mathsf{cart}}$ as follows:

$$\begin{tikzcd}
F(X) \times_{F(Y)} Y \arrow[r] \arrow[d, "F_{\mathscr{D}}(f)"'] \arrow[dr, phantom, very near start, "\lrcorner"] & F(X) \arrow[d, "F(f)"] \\
Y \arrow[r, "\eta_Y"'] & F(Y)
\end{tikzcd}$$

The action on morphisms is canonically induced by the pullback lemma.

Lemma 5.2.2. *In the local universes model $\tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T}$ induced by $(\mathcal{C}, \mathscr{D})$ (Section 3.5.2), the map $F_{\mathscr{D}}$ induces a type connective $[\![\bigcirc]\!] : \mathcal{T} \longrightarrow \mathcal{T}$ modeling the above formation rule. Furthermore, $\tau^*[\![\bigcirc]\!]$ is given by restricting $F_!(\tau)$ along $\eta : \mathsf{id} \longrightarrow F_!$ (explicitly, $\coprod_{f:\mathscr{D}} \mathbf{y}(F_{\mathscr{D}}(f))$).*

Proof. Recall the definition of the presheaf of types:

$$\mathcal{T} = \textstyle\coprod_{f:\mathscr{D}} \mathbf{y}(\mathsf{cod}(f))$$

To model Rule 5.3, we must define a map $[\![\bigcirc]\!] : \mathcal{T} \longrightarrow \mathcal{T}$. We do so by sending f-component of $\mathcal{T}$ to the $F_{\mathscr{D}}(f)$-component (observe that $\mathsf{cod}(f) = \mathsf{cod}(F_{\mathscr{D}}(f))$).

By universality and disjointness of coproducts within $\mathbf{PSh}(\mathcal{C})$, we observe that $\bigcirc^*\tau$ is realized by the following morphism:

$$\begin{tikzcd}
\coprod_{f:\mathscr{D}} \mathbf{y}(\mathsf{dom}(F_{\mathscr{D}}(f))) \arrow[d] \\
\coprod_{f:\mathscr{D}} \mathbf{y}(\mathsf{cod}(F_{\mathscr{D}}(f)))
\end{tikzcd}$$

This is seen to be equivalent to $\eta^* F_!(\tau)$ by recalling that $F_!$ is cocontinuous and sends $\mathbf{y}(c)$ to $\mathbf{y}(F(c))$. □

Remark 5.2.3. The choice of $[\![\bigcirc]\!]$ above is far from the only code classifying this family. For instance, one could equivalently define $[\![\bigcirc]\!]$ to send $\mathsf{in}_f(\alpha)$ to $\mathsf{in}_{F(f)}(\eta \circ \alpha)$. ◇

Lemma 5.2.4. *The point $\eta : \mathsf{id} \longrightarrow F$ induces a term $\Gamma.A \vdash \eta : \bigcirc A[\mathbf{p}]$ within this model.*

Lemma 5.2.5. *Fix a pair of types $\Gamma \vdash A\,\mathsf{type}$ and $\Gamma.\bigcirc A \vdash B\,\mathsf{type}$, in this model there is a canonical isomorphism between $\bigcirc \sum_A B[\mathbf{p}.\eta]$ and $\sum_{\bigcirc A} \bigcirc B$*

Proof. Unfolding the interpretations of A and B within our model, we obtain a pair of display maps $p_A : E_A \longrightarrow U_A$ and $p_B : E_B \longrightarrow U_B$ along with a map $\sigma_A : [\![\Gamma]\!] \longrightarrow U_A$ and a map $\sigma_B : [\![\Gamma]\!] \times_{F(U_A)} F(E_A) \longrightarrow U_B$.

It suffices to argue that these two types classify isomorphic families over $[\![\Gamma]\!]$. Accordingly, we may pull back this data along σ_A and σ_B assume $[\![\Gamma]\!] = U_A$ and $U_B = U_A \times_{F(U_A)} F(E_A)$. We observe that the projection map $U_B \longrightarrow U_A$ is precisely $F_{\mathscr{D}}(p_A)$.

Unfolding constructions given in Awodey [Awo18, Proposition 28], $\sum_A B[\mathbf{p}.\eta]$ classifies the following family over $[\![\Gamma]\!]$:

$$E_B \times_{U_B} E_A \to E_A \longrightarrow U_A$$

Next, let us observe that

$$\begin{aligned}
&F(E_B \times_{U_B} E_A) \times_{F(U_A)} U_A \\
&\cong F(E_B) \times_{F(U_B)} (F(E_A) \times_{F(U_A)} U_A) \\
&\cong F(E_B) \times_{F(U_B)} U_B
\end{aligned}$$

Accordingly, $\bigcirc \sum_A B[\mathbf{p}.\eta]$ classifies the following family $F_{\mathscr{D}}(p_A) \circ F_{\mathscr{D}}(p_B)$. This is precisely what $\sum_{\bigcirc A} \bigcirc B$ classifies, completing the proof. □

These handful of results characterize the typical situation for a fibered modality within type theory. Briefly, it is a pointed (Lemma 5.2.4) lex (Lemma 5.2.5) fibered modality. Further properties of the underlying functor F are also reflected into the type theory. For instance, if F is a monad then one can internalize a fiberwise version of the join operator. Better still, adding a fibered modality requires no changes to the judgmental structure of type theory; we may simply add constants internalizing arbitrarily many fibered modalities to the theory without disrupting substitution.

This construction also illuminates why the later modality from Section 5.1 was so much more amenable to internalization than $\Box$. While both are left exact, the later modality has a point while the $\Box$ modality does not. Both $\Box$ and $\blacktriangleright$, however, satisfy the other properties required by Lemma 5.2.2. Accordingly, we will focus our attention on internalizing modalities which preserve display maps and cartesian squares between them, but do not necessarily admit a point.

5.3 One non-fibered modality

In Section 5.2, we began with particular formation rule (Rule 5.3) and sketched a semantic interpretation for a model of type theory $(\mathcal{C}, \mathscr{D})$ supplemented with a functor F which validated it. A key step in the latter was the existence of a point $\mathbf{1} \longrightarrow F$. It was this point that we used to extend the action of F to the slices of $\mathcal{C}$ and thereby obtain a type-theoretic connective. Unfortunately, however, not every important functor comes equipped with a point. In this section, we reverse perspective slightly. Let us fix a finitely complete category with display maps $(\mathcal{C}, \mathscr{D})$ along with a lex functor F which preserves display maps and consider possible rules for modeling F within the internal language of $\mathcal{C}$.

Fix a display map $p : X \longrightarrow A$ in $\mathcal{C}$. Applying F to p yields another display map over the base $F(A)$. Observing the correspondence between display maps over A and types in context A, we might render this situation syntactically as the following rule:

$$\frac{\Gamma \vdash A \,\mathsf{type}}{F(\Gamma) \vdash F(A) \,\mathsf{type}} \tag{5.4}$$

Using the standard approach for local universe models, we can further arrange that this operation satisfies a limited form of substitution invariance:

$$\frac{\Gamma \vdash \delta : \Delta \qquad \Delta \vdash A \,\mathsf{type}}{F(\Gamma) \vdash F(A)[F(\delta)] = F(A[\delta]) \,\mathsf{type}} \tag{5.5}$$

Finally, using the fact that F is lex, we obtain a canonical isomorphism:

$$\alpha_F : F(\Gamma.A) \cong F(\Gamma).F(A) \tag{5.6}$$

Unfortunately, without further assumptions on F it is difficult to obtain any further reasoning principles. Moreover, adding only these three rules seriously disrupts existing type theory. In particular, we have no general way to reduce $F(A)[\delta]$ so the *substitution property* of type theory (Property 2.4.4) fails; it is not possible to push a substitution through a term and resolve it at variables.

To illustrate this last point, suppose we are given a substitution $\Gamma \vdash \delta : F(\Delta)$ and a type $\Delta \vdash A\,\mathsf{type}$, we note that there no obvious reduction for $F(A)[\delta]$ as F may not even be well-formed in context Γ. What would it take to ensure that we could commute δ past $F(A)$? Examining the rules available, there is no possibility that such a reduction can exist unless $\Gamma = F(\Gamma_0)$; the conclusion of Rule 5.4 insists that the context be of this form. Returning to our semantics, this is akin to asking that if we are given a morphism $X \longrightarrow F(Y)$ then X must lie within the image of F. Recall, however, that we have assumed F to be lex. Therefore, every object admits a map onto $F(\mathbf{1})$, so this requirement is far too strong.

It is possible to ignore the absence of a general substitution rule and simply work with Rules 5.4 to 5.6 as-is. The result is essentially a calculus with *delayed substitutions* [Bd00]. For instance, we may use Rule 5.6 to recover the introduction rule familiar to such systems:

$$\frac{\Gamma \vdash M : A}{F(\Gamma) \vdash \mathsf{mod}(M) \triangleq \mathbf{v}[\alpha_F \circ F(\mathsf{id}.M)] : F(A)}$$

The results, however, are not satisfactory. It is unlikely that such a type theory will satisfy a normalization theorem and the type-checking problem will therefore be undecidable.[2]

Setting aside the issue of normalization, a type theory built around delayed substitutions loses much of the appeal of working within type theory. In a simply-typed setting, one may be able to make ad-hoc simplifications [Clo+15]. In a dependently typed setting, however, it becomes necessary to specify the equational theory of such a system [Biz+16]. Even when the intended models are well-behaved, the resulting system forces the user to write and reason about substitutions in order to work with modal types. In a situation where modal types are used sparingly and with careful abstraction, this may be tenable. If, on the other hand, the modalities are used frequently, if multiple modalities are used simultaneously, or if one must reason about the equality of modal operations, the results are only a small improvement over directly working with the model without an internal language.

We have now arrived at the main question of modal type theory generally and this thesis specifically: can we restrict F or modify the type theory in such a way that Rules 5.4 to 5.6 yield a usable type theory?

Remark 5.3.1. *Useful* is left intentionally vague. In practice, we will ask that our type theory satisfies the familiar metatheorems of Martin-Löf type theory (canonicity, normalization, decidable type-checking) and that the syntax is convenient enough to be used to carry out case studies. ◇

5.4 *Dual-context type theories*

We now turn to our first approach to building a usable theory while incorporating a non-fibered modalities: dual-context type theories. Contemplating Rule 5.4, the issue stems from the fact that not every context Γ can be uniquely written as $F(\Delta)$. As

[2] Note that there is no a priori reason why $F(\delta_0) = F(\delta_1)$ should imply $\delta_0 = \delta_1$ so Rule 5.5 could potentially be used to equate wildly different terms.

discussed above, it is infeasible to force this to be the case, but we can resolve this issue in a different way. Rather than forcing there to be a unique Δ such that $\Gamma = F(\Delta)$, we can hope that there is at least a *chosen* Δ for each each context Γ along with a substitution $\uparrow_\Gamma \colon \Gamma \longrightarrow F(\Delta)$. If we further require that every substitution commute appropriately with $\uparrow_\Gamma$—that is, $\uparrow_{-}$ is natural—we can define a restricted version of Rule 5.4 which respects substitution.

Let us write $\Delta;\Gamma$ to signify that the chosen F-component of Γ is Δ. As semantic intuition, a dual context $\Delta;\Gamma$ represents a pair of an object $[\![\Delta]\!]$ in our category of contexts and a family $[\![\Gamma]\!] : \bullet \longrightarrow F([\![\Delta]\!])$.[3] The new formation rule is defined as follows:

$$\frac{\Delta; \mathbf{1} \vdash A\,\mathsf{type}}{\Delta;\Gamma \vdash \Box A\,\mathsf{type}}$$

There are a variety of different ways to elaborate this idea [dR15; Kav17; PD01; Shu18; Zwa19]. We will follow Zwanziger [Zwa19] and define our dual-context system to *extend* the standard judgments of type theory with copies working over dual-contexts. In particular, we define new forms of judgments $\vdash \Delta;\Gamma\,\mathsf{dcx}$, $\Delta;\Gamma \vdash A\,\mathsf{type}$, $\Delta;\Gamma \vdash M : A$.

5.4.1 Dual-contexts and dual-substitutions

The judgment for dual contexts themselves $\vdash \Delta;\Gamma\,\mathsf{dcx}$ is given by the following rules:

$$\frac{\vdash \Delta\,\mathsf{cx}}{\vdash \Delta;\mathbf{1}\,\mathsf{dcx}} \qquad \frac{\vdash \Delta\,\mathsf{cx} \qquad \vdash \Delta;\Gamma\,\mathsf{dcx} \qquad \Delta;\Gamma \vdash A\,\mathsf{type}}{\vdash \Delta;\Gamma.A\,\mathsf{dcx}}$$

$$\frac{\vdash \Delta\,\mathsf{cx} \qquad \Delta \vdash A\,\mathsf{type} \qquad \vdash \Delta;\Gamma\,\mathsf{dcx}}{\vdash \Delta.A;\Gamma[\mathbf{p}]\,\mathsf{dcx}}$$

The last rule deserves particular scrutiny as it is not typically present in the informal syntax for dual-context type theories. It allows us to take a dual-context $\Delta;\Gamma$ and e.g., add new variables to Δ and weaken Γ to form $\Delta.A;\Gamma[\mathbf{p}]$. Such a weakening must be explicitly noted; returning to our informal semantics Γ represents a family over Δ and it must be pulled-back to live over $\Delta.A$.

As alluded to above, a crucial point of point of dual contexts is that substitutions between them (*dual substitutions*) respect the division between the two contexts. Informally, this means that given a substitution $\Delta_0;\Gamma_0 \longrightarrow \Delta_1;\Gamma_1$ only the elements of Δ_0 are used to implement Δ_1. We will enforce this by defining a projection map from dual substitutions to substitutions simultaneously with the specification of dual substitutions:

$$\frac{\Delta_0;\Gamma_0 \vdash \gamma : \Delta_1;\Gamma_1}{\Delta_0 \vdash \pi(\gamma) : \Delta_1}$$

The calculus of dual substitutions can be presented in a variety of ways, and we will optimize for a tighter connection to the categorical semantics over usability; prior work

[3] This order of dependence is not forced. For an arbitrary F, there is no reason to prefer having Γ depend on $F(\Delta)$ instead of the reverse or some more complex alternative. In practice, however, F is a comonad and this configuration is most advantageous.

has already explored more ergonomic presentations thoroughly [dR15; Kav17; PD01; Shu18; Zwa19].

Many of the rules governing dual substitutions closely mirror those for ordinary substitutions:

$$\frac{\vdash \Delta;\Gamma\ \mathsf{dcx}}{\Delta;\Gamma \vdash \mathsf{id} : \Delta;\Gamma \qquad \pi(\mathsf{id}) = \mathsf{id}} \qquad\qquad \frac{\vdash \Delta_0;\Gamma\ \mathsf{dcx} \qquad \vdash \Delta_1\ \mathsf{cx} \qquad \Delta_0 \vdash \delta : \Delta_1}{\Delta_0;\Gamma \vdash {!_\delta} : \Delta_1;\mathbf{1} \qquad \pi(!_\delta) = \mathsf{id}}$$

$$\frac{\vdash \Delta;\Gamma\ \mathsf{dcx} \qquad \Delta;\Gamma \vdash A\ \mathsf{type}}{\Delta;\Gamma.A \vdash \mathbf{p} : \Delta;\Gamma \qquad \pi(\mathbf{p}) = \mathsf{id}}$$

$$\frac{\begin{array}{c}\vdash \Delta_0;\Gamma_0\ \mathsf{dcx} \qquad \vdash \Delta_1;\Gamma_1\ \mathsf{dcx} \\ \Delta_1;\Gamma_1 \vdash A\ \mathsf{type} \qquad \Delta_0;\Gamma_0 \vdash \gamma : \Delta_1;\Gamma_1 \qquad \Delta_0;\Gamma_0 \vdash M : A[\gamma]\end{array}}{\Delta_0;\Gamma_0 \vdash \gamma.M : \Delta_1;\Gamma_1.A \qquad \pi(\gamma.M) = \pi(\gamma)}$$

$$\frac{\begin{array}{c}\vdash \Delta_0;\Gamma_0\ \mathsf{dcx} \qquad \vdash \Delta_1;\Gamma_1\ \mathsf{dcx} \qquad \vdash \Delta_2;\Gamma_2\ \mathsf{dcx} \\ \Delta_0;\Gamma_0 \vdash \gamma_1 : \Delta_1;\Gamma_1 \qquad \Delta_1;\Gamma_1 \vdash \gamma_2 : \Delta_2;\Gamma_2\end{array}}{\Delta_0;\Gamma_0 \vdash \gamma_2 \circ \gamma_1 : \Delta_2;\Gamma_2 \qquad \pi(\gamma_2 \circ \gamma_1) = \pi(\gamma_2) \circ \pi(\gamma_1)}$$

Note, however, that these rules only allow us to produce dual-substitutions which are sent by π to the identity (*vertical substitutions*). The only way to produce non-vertical dual-substitutions is through the maps governing $\Delta;\Gamma[\mathbf{p}]$:

$$\frac{\Delta \vdash A\ \mathsf{type} \qquad \vdash \Delta;\Gamma\ \mathsf{dcx}}{\Delta.A;\Gamma[\mathbf{p}] \vdash \mathsf{lift}(\mathbf{p}) : \Delta;\Gamma \qquad \pi(\mathsf{lift}(\mathbf{p})) = \mathbf{p}} \tag{5.7}$$

$$\frac{\Delta_1 \vdash A\ \mathsf{type} \qquad \Delta_0 \vdash \delta : \Delta_1 \qquad \Delta_0 \vdash M : A[\delta] \qquad \Delta_0;\Gamma_0 \vdash \gamma : \Delta;\Gamma_1 \qquad \pi(\gamma) = \delta}{\Delta_0;\Gamma_0 \vdash \mathsf{factor}(\gamma;M) : \Delta_1.A;\Gamma_1[\mathbf{p}] \qquad \pi(\mathsf{factor}(\gamma;\delta_1)) = \delta.M} \tag{5.8}$$

$$\frac{\Delta_1 \vdash A\ \mathsf{type} \qquad \Delta_0 \vdash \delta : \Delta_1 \qquad \Delta_0 \vdash M : A[\delta] \qquad \Delta_0;\Gamma_0 \vdash \gamma : \Delta_1;\Gamma_1 \qquad \pi(\gamma) = \delta}{\Delta_0;\Gamma_0 \vdash \mathsf{lift}(\mathbf{p}) \circ \mathsf{factor}(\gamma;M) = \gamma : \Delta_1;\Gamma_1} \tag{5.9}$$

$$\frac{\begin{array}{c}\Delta_1 \vdash A\ \mathsf{type} \qquad \Delta_0 \vdash \delta : \Delta_1 \qquad \Delta_0 \vdash M : A[\delta] \qquad \Delta_0;\Gamma_0 \vdash \gamma : \Delta_1;\Gamma_1 \\ \pi(\gamma) = \delta \qquad \Delta_0;\Gamma_0 \vdash \gamma' : \Delta_1.A;\Gamma_1[\mathbf{p}] \qquad \Delta_0;\Gamma_0 \vdash \mathsf{lift}(\mathbf{p}) \circ \gamma' = \gamma : \Delta_1;\Gamma_1\end{array}}{\Delta_0;\Gamma_0 \vdash \gamma' = \mathsf{factor}(\gamma;M) : \Delta_1.A;\Gamma_1[\mathbf{p}]} \tag{5.10}$$

Rule 5.7 relates $\Delta_1;\Gamma$ and $\Delta_0;\Gamma[\delta]$ by providing a *lifted* version of the weakening substitution: a dual substitution $\Delta.A;\Gamma[\mathbf{p}] \longrightarrow \Delta;\Gamma$ whose image under π is $\mathbf{p}$.

While this rule allows us to map *out* of $\Delta.A;\Gamma[\mathbf{p}]$, the other novel form of substitution Rule 5.8 allows us to map *into* $\Delta.A;\Gamma[\mathbf{p}]$. In fact, it gives $\Delta.A;\Gamma[\mathbf{p}]$ and $\mathsf{lift}(\mathbf{p})$ a universal property exactly like that of $(\Delta;\Gamma.A), \mathbf{p}$. It states that to define a substitution into $\Delta.A;\Gamma[\mathbf{p}]$, it suffices to provide a substitution into $\Delta;\Gamma$ and an extension of its image under π through $\Delta.A$. In fact, taken in conjunction with the two equations specified above (Rules 5.9 and 5.10), Rule 5.8 is the *unique* factorization of γ through $\mathsf{lift}(\mathbf{p})$.

Remark 5.4.1. While the novel rules governing dual-substitutions are quite complex, they admit a short semantic description; these rules ensure the existence of a cartesian lift for each $\Delta.A \vdash \mathbf{p} : \Delta$ along π. ◇

5.4.2 Modal types in dual-context type theory

Having sweated the details around dual-contexts, we are now in a position to quickly and easily specify the modal types in this theory. While many dual-context type theories work with a single comonad, we will decompose this comonad into a pair of *adjoint* modalities witnessing an adjunction between types in normal contexts and types in dual-contexts. The comonad typically taken as primitive then arises as a composite of these adjoints.

We begin with the left adjoint, which promotes a type in a single context to one in a dual-context:

$$\frac{\Delta \vdash A\,\mathsf{type}}{\Delta;\Gamma \vdash \mathcal{L}\,A\,\mathsf{type}} \tag{5.11}$$

After all the effort expended defining dual-contexts and substitutions, we can specify a substitution principle for this formation rule:

$$\frac{\Delta_0;\Gamma_1 \vdash \gamma : \Delta_1;\Gamma_1 \qquad \Delta_1 \vdash A\,\mathsf{type}}{\Delta_0;\Gamma_0 \vdash (\mathcal{L}\,A)[\gamma] = \mathcal{L}(A[\pi\gamma])\,\mathsf{type}}$$

It is here—and the substitution rule for the introduction rule—where the projection from dual-substitutions to substitutions is necessary. Without such an operation, there would be no way to recover a substitution $\Delta_0 \longrightarrow \Delta_1$ from the given dual-substitution $\Delta_0;\Gamma_0 \longrightarrow \Delta_1;\Gamma_1$ and the same difficulties from Section 5.3 would reassert themselves.

Remark 5.4.2. Intuitively, in our semantics Δ will be interpreted as a portion of $[\![\Gamma]\!]$ in the image of F—the functor we are attempting to internalize. This rule will then be interpreted by a combination of weakening (to pass from $[\![\Delta;\Gamma]\!]$ to $F([\![\Delta]\!])$) and application of F. ◇

The introduction rule and its substitution principle for $\mathcal{L}\,A$ directly parallel the formation rule:

$$\frac{\Delta \vdash M : A}{\Delta;\Gamma \vdash \mathsf{mod}_{\mathcal{L}}(M) : \mathcal{L}\,A} \tag{5.12}$$

$$\frac{\Delta_0;\Gamma_0 \vdash \gamma : \Delta_1;\Gamma_1 \qquad \Delta_1 \vdash M : A}{\Delta_0;\Gamma_0 \vdash \mathsf{mod}_{\mathcal{L}}(M)[\gamma] = \mathsf{mod}_{\mathcal{L}}(M[\pi\gamma]) : \mathcal{L}\,A} \tag{5.13}$$

The elimination principle for $\mathcal{L}\,A$ is more complex. While one can see it as a form of pattern-matching similar to the elimination rule for coproducts, a slightly different perspective is more informative. Note that the introduction rule for $\mathcal{L}\,A$ induces the following dual-substitution:

$$\Delta.A;\Gamma[\mathbf{p}] \vdash \mathsf{lift}(\mathbf{p}).\mathsf{mod}_{\mathcal{L}}(\mathbf{v}) : \Delta;\Gamma.\,\mathcal{L}\,A \tag{5.14}$$

This substitution shifts an element of Δ to an element of Γ modulating by the left adjoint modality. While both contexts intuitively represent the same object, this substitution is not generally invertible. The role of the elimination principle is to force this substitution to be weakly orthogonal to display maps. Phrased informally, the elimination principle ensures that when constructing a term these two contexts are equivalent.

$$\frac{\Delta;\Gamma.\,\mathcal{L}\,A \vdash B\,\mathsf{type} \qquad \Delta;\Gamma \vdash M_0 : \mathcal{L}\,A \qquad \Delta.A;\Gamma[\mathbf{p}] \vdash M_1 : B[\mathsf{lift}(\mathbf{p}).\mathsf{mod}_{\mathcal{L}}(\mathbf{v})]}{\Delta;\Gamma \vdash \mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(-) \leftarrow M_0\ \mathsf{in}\ M_1 : B[\mathsf{id}.M_0]} \tag{5.15}$$

$$\frac{\Delta;\Gamma.\,\mathcal{L}\,A \vdash B\,\mathsf{type} \qquad \Delta \vdash M_0 : A \qquad \Delta.A;\Gamma[\mathbf{p}] \vdash M_1 : B[\mathsf{lift}(\mathbf{p}).\mathsf{mod}_{\mathcal{L}}(\mathbf{v})]}{\Delta;\Gamma \vdash \mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(-) \leftarrow \mathsf{mod}_{\mathcal{L}}(M_0)\ \mathsf{in}\ M_1 = M_1[\mathsf{id}.M_0] : B[\mathsf{id}.M_0]} \tag{5.16}$$

The substitution rule for $\mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(-) \leftarrow -\ \mathsf{in}\ -$ is unsurprising, but slightly involved:

$$(\mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(-) \leftarrow M_0\ \mathsf{in}\ M_1)[\gamma] = \mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(-) \leftarrow M_0[\gamma]\ \mathsf{in}\ M_1[\mathsf{factor}(\gamma;\mathbf{v})]$$

Remark 5.4.3. The substitution on M_1, in essence, capitalizes on the fact that cartesian reindexing is functorial. ◇

To relate this rule more directly to the weak orthogonality of Eq. (5.14), observe that we may use Rule 5.15 to solve the necessary lifting problem:

$$\begin{tikzcd}
\Delta.A;\Gamma[\mathbf{p}] \arrow[r, "\mathsf{id}.M"] \arrow[d] & \Delta;\Gamma.\,\mathcal{L}\,A.B \arrow[d, "\mathbf{p}"] \\
\Delta;\Gamma.\,\mathcal{L}\,A \arrow[r, "\mathsf{id}"'] \arrow[ur, dashed, "\mathsf{id}.(\mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(-) \leftarrow \mathbf{v}\ \mathsf{in}\ M)"] & \Delta;\Gamma.\,\mathcal{L}\,A
\end{tikzcd}$$

We now present the rules for the right adjoint modality $\mathcal{R}$:

$$\frac{\Delta;\mathbf{1} \vdash A\,\mathsf{type}}{\Delta \vdash \mathcal{R}\,A\,\mathsf{type}} \qquad \frac{\Delta;\mathbf{1} \vdash M : A}{\Delta \vdash \mathsf{mod}_{\mathcal{R}}(M) : \mathcal{R}\,A} \qquad \frac{\Delta \vdash M : \mathcal{R}\,A}{\Delta;\Gamma \vdash \mathsf{unmod}_{\mathcal{R}}(M) : A}$$

$$\frac{\Delta;\mathbf{1} \vdash M : A}{\Delta;\mathbf{1} \vdash \mathsf{unmod}_{\mathcal{R}}(\mathsf{mod}_{\mathcal{R}}(M)) = M : A} \qquad \frac{\Delta \vdash M : \mathcal{R}\,A}{\Delta \vdash \mathsf{mod}_{\mathcal{R}}(\mathsf{unmod}_{\mathcal{R}}(M)) = M : \mathcal{R}\,A}$$

The substitution principles for these type and term formers follow straightforwardly from π and $\mathsf{lift}(-)$. We leave the details to the reader. Notice that, unlike $\mathcal{L}$, the elimination rule for $\mathcal{R}\,A$ simply inverts the introduction rule. This option is not available for $\mathcal{L}\,A$; without Rule 5.15 it would be impossible to use a variable of type $\mathcal{L}\,A$. As a result, we are able to equip $\mathcal{R}\,A$ with both a β and η principle. To do the same for $\mathcal{L}\,A$ would require the introduction of complex commuting conversions [Kav17].

Remark 5.4.4. While we have only discussed the modalities, all the standard connectives of dependent type theory can be added to this system. We do note, however, that all connectives must be duplicated: one version for dual-contexts and one version for single contexts. While we will not discuss their semantics, we will therefore avail ourselves of dependent sums, dependent products, identity types, etc., in examples. ◇

5.4.3 Programming with dual-contexts

Prior to discussing the semantics of dual-context type theories, we will take a moment to work through the construction of the unit and counit combinators. The purpose here is two-fold. First, we want to show that these two modalities are actually adjoint in some sense. Second, and in some ways more importantly, the purpose of dual-context type theory is to provide a theory more usable than working directly with delayed substitutions or working externally. Small exercises like these provide at least some data on the usability of the theory.

Remark 5.4.5. In order to facilitate these small examples, we will use a more informal syntax with named variables rather than De Bruijn indices. We leave it to the reader to translate these terms back into the core calculus previously presented. ◇

We begin by defining the unit $\eta : A \longrightarrow \mathcal{R}\,\mathcal{L}\,A$ for a type $\Delta \vdash A\,\mathsf{type}$:

$$\begin{aligned}&\eta : A \to \mathcal{R}\,\mathcal{L}\,A\\&\eta = \lambda x.\,\mathsf{mod}_{\mathcal{R}}(\mathsf{mod}_{\mathcal{L}}(x))\end{aligned}$$

In this case, the calculation is very direct: we begin with a term $\Delta \vdash a : A$ and after applying $\mathsf{mod}_{\mathcal{R}}(-)$ and $\mathsf{mod}_{\mathcal{L}}(-)$ our goal is to produce exactly this. In particular, the actions of $\mathsf{mod}_{\mathcal{R}}(-)$ and $\mathsf{mod}_{\mathcal{L}}(-)$ on the context are inverse to each other: sending Δ to $\Delta;\mathbf{1}$ and then back to Δ.

The counit is slightly more involved:

$$\begin{aligned}&\epsilon : \mathcal{L}\,\mathcal{R}\,A \to A\\&\epsilon = \lambda x.\,\mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(y) \leftarrow x\ \mathsf{in}\ \mathsf{unmod}_{\mathcal{R}}(y)\end{aligned}$$

Let us trace through this expression step-by-step. First, we bind $x : \mathcal{L}\,\mathcal{R}\,A$ such that our context is now $\mathbf{1}; x : \mathcal{L}\,\mathcal{R}\,A$. The application of the elimination rule for $\mathcal{L}$ allows us to shift x to an entry within the Δ-portion of the dual-context. We therefore obtain $y : \mathcal{R}\,A; x : \mathcal{L}\,\mathcal{R}\,A$. Finally, we apply the $\mathsf{unmod}_{\mathcal{R}}(-)$ rule which forces us to drop the Γ portion of the context, but we are then able to use $y : \mathcal{R}\,A$ to finish the term.

In addition to merely defining the unit and the counit, we must show that they actually assemble into an internal adjunction i.e., that they satisfy the triangle identities:

$$\begin{aligned}(x : \mathcal{R}\,A) &\to x = \mathsf{mod}_{\mathcal{R}}(\epsilon\,\mathsf{unmod}_{\mathcal{R}}(\eta(x)))\\(x : \mathcal{L}\,A) &\to x = \mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(y) \leftarrow x\ \mathsf{in}\ \epsilon\,\mathsf{mod}_{\mathcal{L}}(\eta(y))\end{aligned}$$

The first law follows by reflexivity; the two terms are definitionally equal:

$$\begin{aligned}&\mathsf{mod}_{\mathcal{R}}(\epsilon\,\mathsf{unmod}_{\mathcal{R}}(\eta(x)))\\&\ = \mathsf{mod}_{\mathcal{R}}(\epsilon\,\mathsf{unmod}_{\mathcal{R}}(\mathsf{mod}_{\mathcal{R}}(\mathsf{mod}_{\mathcal{L}}(x))))\end{aligned}$$

$$
\begin{aligned}
&= \mathsf{mod}_{\mathcal{R}}(\epsilon\,\mathsf{mod}_{\mathcal{L}}(x)) \\
&= \mathsf{mod}_{\mathcal{R}}(\mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(y) \leftarrow \mathsf{mod}_{\mathcal{L}}(x)\ \mathsf{in}\ \mathsf{unmod}_{\mathcal{R}}(y)) \\
&= \mathsf{mod}_{\mathcal{R}}(\mathsf{unmod}_{\mathcal{R}}(x)) \\
&= x
\end{aligned}
$$

The second equation is not definitional, but it does follow from the elimination principle of $\mathcal{L}\,A$. Fix $x : \mathcal{L}\,A$. Through the elimination principle for $\mathcal{L}\,A$, it suffices to show $x = \mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(y) \leftarrow x\ \mathsf{in}\ \epsilon\,\mathsf{mod}_{\mathcal{L}}(\eta(y))$ by constructing such an identification after replacing x by $\mathsf{mod}_{\mathcal{L}}(y)$. After this replacement, the identification is again definitional:

$$
\begin{aligned}
&\mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(y) \leftarrow \mathsf{mod}_{\mathcal{L}}(y)\ \mathsf{in}\ \epsilon\,\mathsf{mod}_{\mathcal{L}}(\eta(y)) \\
&\ = \mathsf{let}\ \mathsf{mod}_{\mathcal{L}}(z) \leftarrow \mathsf{mod}_{\mathcal{L}}(\eta(y))\ \mathsf{in}\ \mathsf{unmod}_{\mathcal{R}}(z) \\
&\ = \mathsf{unmod}_{\mathcal{R}}(\eta(y)) \\
&\ = \mathsf{mod}_{\mathcal{L}}(y)
\end{aligned}
$$

We may therefore summarize the results of this section with the following theorem:

Theorem 5.4.6. *The quadruple* $(\mathcal{L}, \mathcal{R}, \eta, \epsilon)$ *forms an adjunction.*

5.4.4 The semantics of dual-context type theory

While we have not stressed the point thus far, dual-context type theory can be presented as a generalized algebraic theory and therefore gives rise to a category of models. We reformulate this definition in more familiar terms prior to defining the standard model of dual-context type theory in Section 5.4.5.

As an extension of standard type theory, a model of dual-context type theory includes a category of contexts $\mathcal{C}$ equipped with a terminal object modeling ordinary contexts substitutions between them. The structure modeling of dual-contexts and dual-substitutions also arranges into a category $\mathcal{D}$—dual-substitutions include both composition and identity—and the operation π defined on dual-substitutions defines a functor $\pi : \mathcal{D} \longrightarrow \mathcal{C}$.

In order to model types and terms for both single and dual-contexts, we require a pair of representable natural transformations $\tau_{\mathcal{C}} : \mathbf{PSh}(\mathcal{C})$ and $\tau_{\mathcal{D}} : \mathbf{PSh}(\mathcal{D})$. These are also necessary to state the semantic reformulation of $\mathsf{lift}(\mathbf{p})$ and $\mathsf{factor}(-;-)$. In particular, define a morphism $f : \mathcal{C}$ to be a *display map* if $\mathbf{y}(f)$ is a pullback of $\tau_{\mathcal{C}}$. The existence of $\mathsf{lift}(\mathbf{p})$ and $\mathsf{factor}(-;-)$ amount to the requirement that π has cartesian lifts for all display maps.

Finally, the contexts $\Delta; \mathbf{1}$ are terminal objects in the fiber over Δ and the rule for $!_\delta$ ensures that a map into this object is precisely determined by a map into Δ. In total, we have the following:

Definition 5.4.7. A model of dual-context type theory without modal types is given by the following data:

1. categories $\mathcal{C}$ and $\mathcal{D}$ and a functor $\pi : \mathcal{D} \longrightarrow \mathcal{C}$ such that $\mathcal{C}$ has a terminal object,
2. a pair of representable natural transformations $\tau_{\mathcal{C}} : \mathbf{PSh}(\mathcal{C})$ and $\tau_{\mathcal{D}} : \mathbf{PSh}(\mathcal{D})$,

3. such that each display map in $\mathcal{C}$ has a cartesian lift along π,
4. and the functor π has a right adjoint T.

There is another form of context extension in dual-context type theory: extending $\Delta; \Gamma$ to $\Delta.A; \Gamma[\mathbf{p}]$. No further structure is needed to model this, however, as it follows from the representability of $\tau_\mathcal{C}$ and the fact that π is enough cartesian lifts:

Lemma 5.4.8. $\pi^*\tau_\mathcal{C}$ *is a representable natural transformation.*

Proof. Fix $A : \mathbf{y}(D) \longrightarrow \pi^*\mathcal{T}_\mathcal{C}$. We wish to find a representation for $\mathbf{y}(d) \times_{\pi^*\mathcal{T}_\mathcal{C}} \pi^*\mathcal{T}^\bullet_\mathcal{C}$.

First, observe that A is determined by a map $A_0 : \mathbf{y}(\pi D) \longrightarrow \mathcal{T}_\mathcal{C}$ fitting into the following diagram by assumption:

$$\begin{tikzcd}
\mathbf{y}(C) \arrow[r, "p_1"] \arrow[d, "\mathbf{y}(p_0)"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^\bullet_\mathcal{C} \arrow[d] \\
\mathbf{y}(\pi D) \arrow[r] & \mathcal{T}_\mathcal{C}
\end{tikzcd}$$

As a cartesian fibration, we are able to lift p_0 to a morphism $p_0^\dagger : p_0^* D \longrightarrow D$. We claim that $p_0^* D$ is the desired representation. To this end, fix the following:

$$\begin{tikzcd}
\mathbf{y}(D') \arrow[drr, bend left, "g"] \arrow[ddr, bend right, "f"'] & & \\
& \mathbf{y}(p^* D) \arrow[r, "\widehat{p_1}"] \arrow[d, "\mathbf{y}(p_0^\dagger)"] & \pi^*\mathcal{T}^\bullet_\mathcal{C} \arrow[d] \\
& \mathbf{y}(D) \arrow[r] & \pi^*\mathcal{T}_\mathcal{C}
\end{tikzcd}$$

We wish to find a unique map $D' \longrightarrow p^* D$ fitting in to this diagram. Transposing along $\pi_! \dashv \pi^*$, we obtain a unique map $b : \pi(D') \longrightarrow C$ such that $\pi(d_0) = p \circ b$:

$$\begin{tikzcd}
\mathbf{y}(\pi D') \arrow[drr, bend left, "\widehat{g}"] \arrow[ddr, bend right, "\pi f"'] \arrow[dr, dotted, "h"] & & \\
& \mathbf{y}(C) \arrow[r, "p_1"] \arrow[d, "\mathbf{y}(p_0)"] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^\bullet_\mathcal{C} \arrow[d] \\
& \mathbf{y}(\pi D) \arrow[r] & \mathcal{T}_\mathcal{C}
\end{tikzcd}$$

The universal property of the cartesian lift $p_0^\dagger$ then ensures that there is a unique map $\bar{h}$ laying over h such that $p^\dagger \circ \bar{h} = d_0$. Moreover, $\widehat{p_1} \circ \bar{h} = g$ by naturality of transposition. Accordingly, $\bar{h}$ is the required 'gap map' and its unicity follows from the unicity of h and the universal property of $p^\dagger$. □

It remains to explain how the existence of $\mathcal{L}\,A$ and $\mathcal{R}\,A$ are reflected in a model. Prior to this, we require a few preliminaries on the functors $\pi^* : \mathbf{PSh}(\mathcal{C}) \longrightarrow \mathbf{PSh}(\mathcal{D})$ and $T^* : \mathbf{PSh}(\mathcal{C}) \longrightarrow \mathbf{PSh}(\mathcal{D})$ induced by precomposition.

Consider the syntactic model of dual-context type theory. In this model, the presheaf $\mathcal{T}_{\mathcal{D}}$ sends $\Delta;\Gamma$ to the set of types $\{A \mid \Delta;\Gamma \vdash A\,\mathsf{type}\}$. Similarly, $\mathcal{T}_{\mathcal{C}}$ sends Δ to the set of types $\{A \mid \Delta \vdash A\,\mathsf{type}\}$. In this case, $\pi^*\mathcal{T}_{\mathcal{C}}$ sends a dual-context $\Delta;\Gamma$ to the set of types $\{A \mid \Delta \vdash A\,\mathsf{type}\}$ and $T^*\mathcal{T}_{\mathcal{D}}$ sends Δ to types in context $\Delta;\mathbf{1}$.

Recall the formation rule for $\mathcal{R}\,A$, which takes a type context $\Delta;\mathbf{1}$ and produces a type in context Δ. This formation rule, along with its substitution principle, is then captured by a single morphism:

$$T^*\mathcal{T}_{\mathcal{D}} \longrightarrow \mathcal{T}_{\mathcal{C}}$$

More generally, we may use π^* and T^* to model premises with contexts modified by adding or dropping the Γ-zone of a dual-context. Just as we used exponentiation by $\tau[A]$ to model extending a context by a variable of type A, we will use these functors to specify the modal types.

We will begin with $\mathcal{R}$ because the presence of an η law allows us to give a particularly elegant description. Notice that the elimination rule for $\mathcal{R}\,A$ ensures that the set of terms $\Delta \vdash M : \mathcal{R}\,A$ is naturally isomorphic to the set of terms $\Delta;\mathbf{1} \vdash M : A$. In a general model of dual-context type theory, all the rules of $\mathcal{R}$ are then captured by the following pullback square:

$$\begin{tikzcd}
T^*\mathcal{T}_{\mathcal{D}}^{\bullet} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}_{\mathcal{C}}^{\bullet} \arrow[d] \\
T^*\mathcal{T}_{\mathcal{D}} \arrow[r] & \mathcal{T}_{\mathcal{C}}
\end{tikzcd} \tag{5.17}$$

The bottom and top arrows of this diagram model the formation and introduction rules of the right adjoint modality. The gap map induced by the universal property of the pullback square models the elimination rule. The required β and η properties follow from the properties of the gap map.

The left adjoint is more involved because, like other types without an η law, we cannot simply bundle all the information of the connective into one pullback square. Instead, we must specify the formation and introduction rules and only afterwards the elimination rule. The formation and introduction rules, at least, mirror those of $\mathcal{R}$ and are captured by the existence of a the following commuting square:

$$\begin{tikzcd}
\pi^*\mathcal{T}_{\mathcal{C}}^{\bullet} \arrow[r] \arrow[d] & \mathcal{T}_{\mathcal{D}}^{\bullet} \arrow[d] \\
\pi^*\mathcal{T}_{\mathcal{C}} \arrow[r] & \mathcal{T}_{\mathcal{D}}
\end{tikzcd} \tag{5.18}$$

Recall prior our explanation of the elimination principle for $\mathcal{L}\,A$ in Section 5.4.2: the elimination rule forces the canonical substitution $\Delta.A;\Gamma[\mathbf{p}] \longrightarrow \Delta;\Gamma.\,\mathcal{L}\,A$ to be anodyne. Just as with intensional identity types or coproducts, this elimination rule can be modeled by a left lifting structure as described in Section 3.4.

The essence of Lemma 5.4.8 is that we can model $\Delta.A;\Gamma[\mathbf{p}]$ in a model by considering a fiber of $\pi^*\tau_{\mathcal{C}}$. Accordingly, the morphism we need to lift against is canonical map $\pi^*\mathcal{T}_{\mathcal{C}}^\bullet \longrightarrow \pi^*\mathcal{T}_{\mathcal{C}} \times_{\mathcal{T}_{\mathcal{D}}} \mathcal{T}_{\mathcal{D}}^\bullet$ in the slice category $\mathbf{PSh}(\mathcal{D})/\pi^*\mathcal{T}_{\mathcal{C}}$. We wish for this to be orthogonal to projections $\Delta;\Gamma.B \longrightarrow \Delta;\Gamma$. It therefore suffices to require the following lifting structure in $\mathbf{PSh}(\mathcal{D})/\pi^*\mathcal{T}_{\mathcal{C}}$:

$$\left(\pi^*\mathcal{T}_{\mathcal{C}}^\bullet \longrightarrow \pi^*\mathcal{T}_{\mathcal{C}} \times_{\mathcal{T}_{\mathcal{D}}} \mathcal{T}_{\mathcal{D}}^\bullet\right) \pitchfork \left(\pi^*\mathcal{T}_{\mathcal{C}} \times \tau_{\mathcal{D}}\right) \tag{5.19}$$

Definition 5.4.9. A model of dual-context type theory consists of a model without modal types (Definition 5.4.7) supplemented with Structures 5.17 to 5.19.

5.4.5 The standard model

While it is not strictly necessary, we devote some attention to the "standard model" of dual-context type theory as it provides simpler examples of several phenomenon that will become important in later chapters. This material may, however, be skipped upon first reading.

This model of dual-context type theory is induced by an adjunction between a pair categories equipped with classes of display maps $F : (\mathcal{E}, \mathscr{D}_{\mathcal{E}}) \rightleftarrows (\mathcal{F}, \mathscr{D}_{\mathcal{F}}) : G$ and is designed so that the modal types internalize these two functors. We will assume that both $\mathcal{E}$ and $\mathcal{F}$ have terminal objects and pullbacks along display maps and that F preserves them. We additionally require that both F and G preserve display maps.

Our presentation of this model is heavily inspired by Zwanziger [Zwa22]. We offer a modest improvement on op. cit. by not requiring $\mathcal{E}$ and $\mathcal{F}$ come equipped with chosen CwF structures which the adjunctions respect. Instead, we adapt Shulman's general coherence result based on local universes [LW15; Shu23] to strictify $\mathcal{E}$ and $\mathcal{F}$.

Remark 5.4.10. While we have discussed F thus far as if its a endofunctor on some category $\mathcal{C}$, it adds no complexity to allow F to be a functor between two different categories. In fact, in many examples we will be primarily interested in the comonad $\square\bigcirc A$, but it is most natural to break realize $\square$ and $\bigcirc$ as functors between different categories. ◇

Recall our earlier intuition about dual-contexts: a dual context $\Delta;\Gamma$ corresponds to a family $\Gamma \longrightarrow F(\Delta)$. We can codify such families into a category. In fact we have already encountered this category: it is the Artin gluing $\mathbf{Gl}(F)$. We recall the network of adjoints between $\mathbf{Gl}(F)$ and $\mathcal{E}$ and $\mathcal{F}$:

$$\begin{tikzcd}
\mathcal{E} \arrow[r, "\mathbf{j}_*" description] & \mathbf{Gl}(F) \arrow[l, shift right=3, "\mathbf{j}^*" description] \arrow[l, shift left=3, "\mathbf{j}^!" description] \arrow[r, shift left=2, "\mathbf{i}^*" description] & \mathcal{F} \arrow[l, shift left=2, "\mathbf{i}_*" description]
\end{tikzcd}$$

Remark 5.4.11. The existence of $\mathbf{j}^!$ is not automatic; its existence is a consequence of F being a left adjoint. ◇

We further recall that $\mathbf{j}^*$ is a cartesian fibration, and $\mathbf{i}_*$ is an LCC functor.

Some care is required to produce the representable natural transformations of terms and types. We will take define the universe of types over $\mathcal{E}$ to be the standard local universes construction $\tau_{\mathcal{E}}$. In particular, $\mathcal{T}_{\mathcal{E}}(A)$ consists of sets of diagrams of the following shape in $\mathcal{E}$ where $p \in \mathscr{D}_{\mathcal{E}}$:

$$\begin{tikzcd}
 & U \arrow[d, "p"] \\
A \arrow[r] & V
\end{tikzcd}$$

We require a slightly more complex construction for the universe of types over dual-contexts. Essentially, we pull the universe of types in $\mathcal{F}$ along $\mathbf{i}^*$. Accordingly, $\mathcal{T}_{\mathbf{Gl}(F)}\big(X_1 \longrightarrow F(X_0)\big)$ consists of sets of diagrams of the following shape in $\mathcal{F}$ with $p \in \mathscr{D}_{\mathcal{F}}$:

$$\begin{tikzcd}
 & U \arrow[d, "p"] \\
X_1 \arrow[r] & V
\end{tikzcd}$$

Lemma 5.4.12. $(\mathcal{E}, \mathbf{Gl}(F), \tau_{\mathcal{E}}, \tau_{\mathbf{Gl}(F)})$ *is a model of dual-context type theory without modal types.*

Proof. Recalling that F preserves display maps and $\mathcal{F}$ has pullbacks along display maps, we may argue that $\mathbf{j}^*$ has cartesian lifts of maps in $\mathcal{D}_{\mathcal{F}}$. We have already noted that $\mathbf{j}^*$ has a right adjoint.

We have already shown that $\tau_{\mathcal{E}}$ is representable. It remains only to show that $\tau_{\mathbf{Gl}(F)} = (\mathbf{i}^*)^*\tau_{\mathcal{F}}$ is representable. This end, fix $A : \mathbf{y}(X) \longrightarrow \mathcal{T}_{\mathbf{Gl}(F)}$.

By definition of left Kan extension, $(\mathbf{i}^*)_!\mathbf{y}(Z) \cong \mathbf{y}(\mathbf{i}^*Z)$ and by manipulating adjoints, we further observe that $(\mathbf{i}^*)^*\mathbf{y}(Z) \cong \mathbf{y}(\mathbf{i}_*Z)$. Taking these two facts into consideration, we may factor A through the unit of the adjunction $(\mathbf{i}^*)_! \dashv (\mathbf{i}^*)^*$. Applying the pullback lemma, we are reduced to showing that the left-hand square is a pullback in the following diagram:

$$\begin{tikzcd}
Z \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & W \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & (\mathbf{i}^*)^*\mathcal{T}^\bullet_{\mathcal{F}} \arrow[d] \\
\mathbf{y}(X) \arrow[r] & \mathbf{y}(\mathbf{i}_*\mathbf{i}^*X) \arrow[r] & (\mathbf{i}^*)^*\mathcal{T}_{\mathcal{F}}
\end{tikzcd}$$

To this end, let us first observe that $W \cong \mathbf{y}(\mathbf{i}_*Y)$ for some $p : Y \longrightarrow \mathbf{i}^*X \in \mathscr{D}_{\mathcal{F}}$ as a fiber of $\tau_{\mathcal{F}}$ over a representable object. It therefore suffices to argue that $\mathbf{i}_*Y \times_{\mathbf{i}_*\mathbf{i}^*X} X$ exists in $\mathbf{Gl}(F)$. Unfolding, we are able to realize this limit with $Y \longrightarrow F(\mathbf{j}^*X)$. □

Remark 5.4.13. While we have chosen to define context extensions over $\mathbf{Gl}(F)$ abstractly, one could also give a direct definition sending a context $X \longrightarrow F(A)$ and a type $\big(X \longrightarrow V, U \longrightarrow V\big)$ to $X \times_V U \longrightarrow F(A)$. The reader may find it instructive to verify that the more abstract construction given above yields the same result. ◇

It remains only to show that this model supports modal types. We will begin with the right adjoint modality as it is easier to construct.

Lemma 5.4.14. $(\mathcal{E}, \mathbf{Gl}(F), \tau_{\mathcal{E}}, \tau_{\mathbf{Gl}(F)})$ *supports right adjoint modalities.*

Proof. We must construct a diagram in the shape of Structure 5.17:

$$\begin{tikzcd}
(\mathbf{j}_*)^*\mathcal{T}^\bullet_{\mathbf{Gl}(F)} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^\bullet_{\mathcal{E}} \arrow[d] \\
(\mathbf{j}_*)^*\mathcal{T}_{\mathbf{Gl}(F)} \arrow[r] & \mathcal{T}_{\mathcal{E}}
\end{tikzcd}$$

To this end, recall that $\tau_{\mathbf{Gl}(F)} \cong (\mathbf{i}_*)_!\tau_{\mathcal{F}}$ and $(\mathbf{j}_*)^* = (\mathbf{j}^!)_!$. Accordingly, we may replace the left-hand vertical map in this diagram with $(\mathbf{j}^! \circ \mathbf{i}_*)_!\tau_{\mathcal{F}} \cong G_!\tau_{\mathcal{F}} \cong F^*\tau_{\mathcal{F}}$.

We now directly define a natural transformation $\alpha : F^*\tau_{\mathcal{F}} \longrightarrow \tau_{\mathcal{E}}$. The component at C sends a type to its transpose along $F \dashv G$:

$$\alpha\big(F(X) \longrightarrow V, U \longrightarrow V\big) = \big(X \longrightarrow G(V), G(U) \longrightarrow G(V)\big)$$

Note that G preserves display maps, so α sends a local universe to a valid local universe. The naturality of transposition ensures that this assignment is natural in C.

We can directly check that the elements of these two types are naturally isomorphic—again by transposition—whereby we obtain the required pullback:

$$\begin{tikzcd}
F^*\mathcal{T}^\bullet_{\mathcal{F}} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^\bullet_{\mathcal{E}} \arrow[d] \\
F^*\mathcal{T}_{\mathcal{F}} \arrow[r, "\alpha"'] & \mathcal{T}_{\mathcal{E}}
\end{tikzcd}$$

□

Lemma 5.4.15. $(\mathcal{E}, \mathbf{Gl}(F), \tau_{\mathcal{E}}, \tau_{\mathbf{Gl}(F)})$ *supports left adjoint modalities.*

Proof. We must exhibit Structures 5.18 and 5.19. We begin with Structure 5.18. We must construct a diagram of the following shape:

$$\begin{tikzcd}
(\mathbf{j}^*)^*\mathcal{T}^\bullet_{\mathcal{E}} \arrow[r] \arrow[d] & \mathcal{T}^\bullet_{\mathbf{Gl}(F)} \arrow[d] \\
(\mathbf{j}^*)^*\mathcal{T}_{\mathcal{E}} \arrow[r] & \mathcal{T}_{\mathbf{Gl}(F)}
\end{tikzcd}$$

By rearranging and transposing with the various adjoints involved, it suffices to produce the following:

$$\begin{tikzcd}
\mathcal{T}^\bullet_{\mathcal{E}} \arrow[r] \arrow[d] & (\mathbf{i}^* \circ \mathbf{j}_*)^*\mathcal{T}^\bullet_{\mathcal{F}} \arrow[d] \\
\mathcal{T}_{\mathcal{E}} \arrow[r] & (\mathbf{i}^* \circ \mathbf{j}_*)^*\mathcal{T}_{\mathcal{F}}
\end{tikzcd}$$

However, $\mathbf{i}^* \circ \mathbf{j}_* = F$. So we may once again simplify this diagram to

$$\begin{tikzcd}
\mathcal{T}_{\mathcal{E}}^{\bullet} \arrow[r] \arrow[d] & F^*\mathcal{T}_{\mathcal{F}}^{\bullet} \arrow[d] \\
\mathcal{T}_{\mathcal{E}} \arrow[r] & F^*\mathcal{T}_{\mathcal{F}}
\end{tikzcd}$$

We define directly natural transformations $\alpha : \mathcal{T}_{\mathcal{E}} \longrightarrow F^*\mathcal{T}_{\mathcal{F}}$ and $\alpha^{\bullet} : \mathcal{T}_{\mathcal{E}}^{\bullet} \longrightarrow F^*\mathcal{T}_{\mathcal{F}}^{\bullet}$. For instance, we define α as follows:

$$\alpha\big(A \longrightarrow V, U \longrightarrow V\big) = \big(F(A) \longrightarrow F(V), F(U) \longrightarrow F(V)\big)$$

Recall that F sends display maps to display maps, so this definition is valid. The definition of $\alpha^{\bullet}$ is similarly defined to use F.

It remains to construct Structure 5.19. We begin by unraveling this structure slightly. It suffices to find a family of lifts to diagrams of the following shape, natural in $A : \mathbf{y}(X) \longrightarrow (\mathbf{j}^*)^*\mathcal{T}_{\mathcal{E}}$:

$$\begin{tikzcd}
\mathbf{y}(X) \times_{(\mathbf{j}^*)^*\mathcal{T}_{\mathcal{E}}} (\mathbf{j}^*)^*\mathcal{T}_{\mathcal{E}}^{\bullet} \arrow[r] \arrow[d] & (\mathbf{i}^*)^*\mathcal{T}_{\mathcal{F}}^{\bullet} \arrow[d] \\
\mathbf{y}(X) \times_{(\mathbf{j}^*)^*\mathcal{T}_{\mathcal{E}}} (\mathbf{j}^*)^*\mathcal{T}_{\mathcal{E}} \times_{\mathcal{T}_{\mathbf{Gl}(F)}} \mathcal{T}_{\mathbf{Gl}(F)}^{\bullet} \arrow[r] & (\mathbf{i}^*)^*\mathcal{T}_{\mathcal{F}}
\end{tikzcd}$$

Let us observe that both the upper and lower left-hand corners are representable. The first by Lemma 5.4.8 and the second by the representability of $\tau_{\mathbf{Gl}(F)}$. We may calculate the representations for both of these objects. To this end, let us unfold $X = X_1 \longrightarrow F(X_0)$ and note that A is determined by a pair of maps $\big(X_0 \longrightarrow V, U \longrightarrow V\big)$ in $\mathcal{E}$.

For the upper-left corner, it follows from the proof of Lemma 5.4.8 that it is the cartesian reindexing of X over $X_0 \times_V U$. Inspecting $\mathbf{j}^*$, this is $X_1 \times_{F(V)} F(U) \longrightarrow F(X_0 \times_V U)$. For the lower-left corner, the proof of Lemma 5.4.12 reveals that this limit is represented by $X_1 \times_{F(V)} F(U) \longrightarrow F(X_0)$.

Returning to our original diagram, we transpose along $(\mathbf{i}^*)_! \dashv (\mathbf{i}^*)^*$ and observe that it suffices to find a family of natural lifts for the following diagrams:

$$\begin{tikzcd}
X_1 \times_{F(V)} F(U) \arrow[r] \arrow[d] & \mathcal{T}_{\mathcal{F}}^{\bullet} \arrow[d] \\
X_1 \times_{F(V)} F(U) \arrow[r] & \mathcal{T}_{\mathcal{F}}
\end{tikzcd}$$

As the left-hand map is an identity, this conclusion follows immediately. □

Theorem 5.4.16. $(\mathcal{E}, \mathbf{Gl}(F), \tau_{\mathcal{E}}, \tau_{\mathbf{Gl}(F)})$ *is a model of dual-context type theory.*

5.4.6 Conclusions

Dual-context type theory allows us to extend type theory with a pair of adjoint modalities and the resulting theory is quite usable. This is apparent not just from the small examples carried out in Section 5.4.3, but from the larger examples carried out in related theories in the literature [Lic+18; Shu18]. Indeed, returning to our original motivation from Section 5.1, Licata et al. [Lic+18] formalized their construction of a univalent universe in an extension of Agda with a modality presented in a similar style [Vez18]. Concretely, the $\flat$ modality used in this formalization can be encoded in dual-context type theory as follows:

$$\flat A = \Box \bigcirc A$$

Moreover, while we are not able to interpret the theory into exactly a pair of categories linked by an adjunction, the model given in Section 5.4.5 gives a good approximation. Importantly, it is at least theoretically to unfold statements in dual-context type theory to obtain actual constructions in a pair of categories, even if this process may be complex in some cases.

Finally, while we have not examined the metatheory of this system, it seems likely many of the desirable properties one might expect from type theory (normalization, canonicity, decidable type-checking, etc.) will hold.

The limitations of dual-context type theory, however, becomes apparent when we consider extending the theory with more than just an adjoint pair of modality. The key insight of dual-context type theory was to rectify Rule 5.4 by ensuring that each context could be decomposed into an F-portion Δ and a non-F portion Γ. If we begin to add more modalities, the complexity of the context structure explodes. If we have to consider F_0 and F_1, we will require different contexts allowing us to interpret Rule 5.4 for F_0, F_1, $F_0 \circ F_1$, $F_1 \circ F_0$, etc. While this is mathematically possible—it underlays Shulman [Shu23]—such a complex structure does not generate a usable syntax. Even in relatively concrete cases, this is problematic: while dual-context type theories suffice for $\Box$, it is unclear how one might scale them to $\Box$ and $\blacktriangleright$ in such a way as to support $\Box \circ \blacktriangleright \simeq \Box$.

5.5 Fitch-style/Kripke-style type theories

Since the problem with scaling dual-context type theories was an explosion in the structure necessary for each context, we might hope to regain control over contexts by ensuring that as many choices as possible are forced or at least canonical. In particular, the dual-context type theory ensures that each dual-context Γ is associated to some Δ by a map $p : \Gamma \longrightarrow F(\Delta)$. We have previously argued its impractical for this map to be uniquely determined, but we could argue that it is at least *initial* among such maps. That is, for any alternative Δ' and any map $p' : \Gamma \longrightarrow F(\Delta')$ there is a unique map $f : \Delta \longrightarrow \Delta'$ fitting into the following triangle:

$$\begin{tikzcd} & \Gamma \arrow[dl] \arrow[dr] & \\ F(\Delta) \arrow[rr, "F(f)"'] & & F(\Delta') \end{tikzcd} \tag{5.20}$$

The crucial point is that there is a unique-up-to-isomorphism choice of initial objects, whereby it becomes unnecessary to record Δ as part of the structure of Γ. The existence of sufficiently many such maps can be rephrased directly in terms of F:

Lemma 5.5.1. *Fix $F : \mathcal{C} \longrightarrow \mathcal{D}$. There exists an initial $C \longrightarrow F(D)$ for each $C : \mathcal{C}$ if and only if F is a right adjoint.*

Proof. As previously argued, given C there is a contractible choice of objects D and maps $\eta_C : C \longrightarrow F(D)$ initial among such maps. Set $L(C) = D$ and define the unit of the purported adjunction at C to be η_C. Calculation shows that L is functorial and η is the unit of an adjunction. □

Accordingly, we arrive at the intuition that we may add more modalities to our theory without the contexts becoming more complex provided we can ensure that each modality behaves like a right adjoint.

Why have we opted for the initial such map rather than e.g., the terminal map? Indeed such a terminal map always exists: $\Gamma \longrightarrow \mathbf{1} \cong F(\mathbf{1})$. This map, however, is considerably less useful; it corresponds to the following rule:

$$\frac{\mathbf{1} \vdash A\,\mathsf{type}}{\Gamma \vdash F(A)\,\mathsf{type}}$$

Indeed, if we return to Rule 5.4, the existence of an initial map $\Gamma \longrightarrow F(\Delta)$ corresponds exactly to having a 'best' delayed substitution to chose as a premise of this rule. Rule 5.5 together with Diagram 5.20 mean that any other choice of delayed substitution can be recovered if we choose $L(\Gamma)$ and η. Indeed, suppose we are given Δ and $\delta : \Gamma \longrightarrow F(\Delta)$, then there exists a *unique* u such that

$$F(A[u])[\eta] = F(A)[\delta]$$

To a first approximation, therefore, the left adjoint to F gives a best possible choice of delayed substitution—and the right adjoint gives the worst.

5.5.1 From right adjoints to dependent right adjoints

We now wish to shape this intuition sketched above into a workable syntax for type theory. In particular, we will follow Birkedal et al. [Bir+20] and present a dependent type theory extended with a single modality $\Box$. The discussion above indicated that we should view $\Box$ as a right adjoint, so we will force this to be the case by extending contexts with a left adjoint action $L(-)$.

In the dual-context case, the left adjoint existed—as the map $\Delta; \Gamma \mapsto \Delta$—but we did not work with this characterization explicitly. In this calculus, we will not attempt to specify the left adjoint as an operation mapping a telescope of variables to a telescope of variables. Instead, we will directly add a new context former to represent the left adjoint and require sufficient structure to treat it as such. In particular, we will entirely disregard how this left adjoint might 'compute' in specific examples:

$$\frac{\vdash \Gamma\,\mathsf{cx}}{\vdash L(\Gamma)\,\mathsf{cx}} \qquad \frac{\vdash \Gamma, \Delta\,\mathsf{cx} \qquad \Gamma \vdash \delta : \Delta}{L(\Delta) \vdash L(\delta) : L(\Delta)} \qquad \frac{\vdash \Gamma\,\mathsf{cx}}{L(\Gamma) \vdash L(\mathsf{id}) = \mathsf{id} : L(\Gamma)}$$

$$\frac{\vdash \Gamma_0, \Gamma_1, \Gamma_2\,\mathsf{cx} \qquad \Gamma_0 \vdash \gamma_1 : \Gamma_1 \qquad \Gamma_1 \vdash \gamma_2 : \Gamma_2}{L(\Gamma_0) \vdash L(\gamma_2 \circ \gamma_1) = L(\gamma_2) \circ L(\gamma_1) : L(\Gamma_2)}$$

These rules can be quickly summarized by stating that we have "added a new functorial context former L".

The rules for modalities follow the pattern indicated above with the introduction and elimination rules for $\Box$ becoming 'transposition' of a sort:

$$\frac{L(\Gamma) \vdash A\,\mathsf{type}}{\Gamma \vdash \Box A\,\mathsf{type}} \qquad \frac{L(\Gamma) \vdash M : A}{\Gamma \vdash \mathsf{mod}(A) : \Box A} \qquad \frac{\Gamma \vdash M : \Box A}{L(\Gamma) \vdash \mathsf{unmod}(M) : A}$$

$$\frac{L(\Gamma) \vdash M : A}{L(\Gamma) \vdash \mathsf{unmod}(\mathsf{mod}(M)) = M : A} \qquad \frac{\Gamma \vdash M : \Box A}{\Gamma \vdash \mathsf{mod}(\mathsf{unmod}(M)) = M : A}$$

$$(\Box A)[\delta] = \Box(A[L(\delta)]) \qquad \mathsf{mod}(M)[\delta] = \mathsf{mod}(M[L(\delta)])$$

$$\mathsf{unmod}(M)[L(\delta)] = \mathsf{unmod}(M[\delta])$$

The substitution rules use the functorial action of $L(-)$ to commute a substitution past the formation and introduction rules. One small oddity is the formation rule which ensures states that $\Box A$ is a type in context Γ just when A is a type in context $L(\Gamma)$. That is, $\Box$ is not quite a "functor" on the category of types. This, however, is to be expected in a situation where $\Box$ is not assumed to be fibered and therefore cannot be presented as a functor on the universe.

Remark 5.5.2. In fact, all of these rules are derivable in the calculus of Section 5.3 with the additional assumption of an initial map $\eta : \Gamma \longrightarrow F(L(\Gamma))$ for each Γ. For instance, we *define* $\Box A = F(A)[\eta]$ and the above substitution rule follows from the naturality of η:

$$(\Box A)[\delta] = F(A)[\eta \circ \delta] = F(A)[F(L(\delta)) \circ \eta] = F(A[\delta])[\eta] = \Box(A[\delta])$$

The other connectives and equations are derivable from similar considerations. ◇

The modality itself does not act on contexts, only on types. This was also the case for dual-context type theory, but this restriction prevents us from simply defining $L(-)$ as a left adjoint: what would it be a left adjoint to? We therefore arrive at the notion of a *dependent adjunction*.

Definition 5.5.3. A dependent adjunction consists of three components: a functor F on contexts, a type former G mapping types $F(\Gamma) \vdash A\,\mathsf{type}$ to a type $\Gamma \vdash G(A)\,\mathsf{type}$, and a natural bijection of terms taken up to definitional equality:

$$\{M \mid F(\Gamma) \vdash M : A\} \cong \{N \mid \Gamma \vdash N : G(A)\}$$

We may concisely summarize the rules above by saying that they freely extend type theory with a dependent adjunction.

Dependent adjunctions are nearly as old as dual-contexts. They first appeared in Davies and Pfenning [DP01] but without the recognition of the universal property they enjoyed. They have experienced renewed interest, however, with the work on Clocked Type Theory [BGM17] and parallel work by Clouston [Clo18]. Both the term "dependent right adjoint" and the full generalization to dependent types are due to Birkedal et al. [Bir+20]. Clouston [Clo18] refers to modalities presented in this style as *Fitch-style modalities*, while other work [HP23] has referred to them as *Kripke-style modalities*.

The main advantage of using dependent adjunctions to structure modalities is the flexibility. In order to work fluidly with multiple modalities, one need only add correspondingly many dependent adjunctions. There is, however, a significant flaw in working with the calculus presented above. Consider again the elimination rule for $\Box$:

$$\frac{\Gamma \vdash M : \Box A}{L(\Gamma) \vdash \mathsf{unmod}(M) : A} \tag{5.21}$$

Unfortunately, this rule exhibits exactly the same deficiencies as Rule 5.4. There is no way to equip this rule with a general substitution principle. We have a limited form of substitution—$\mathsf{unmod}(M)[L(\delta)] = \mathsf{unmod}(M[\delta])$—but no more. The result is nearly the same as with Section 5.3, though we have shifted the problem from the formation and introduction rules to elimination rules.

Two possible solutions present themselves if we are to continue using Rule 5.21:

1. We can attempt to make an argument based on the special properties of $L(-)$ in the category of syntactic contexts and substitutions that all substitutions $\Delta \longrightarrow L(\Gamma)$ can be factored in some canonical way.

2. One could carry out the analysis indicated above but then axiomatize these properties directly to ensure that the additional structure is always present.

Historically, the first work on dependent Fitch-style type theories opted for the first approach [BGM17; Bir+20; GSB19a]. Examining the syntactic category of contexts and substitutions, one may directly characterize the substitutions $\Delta \longrightarrow L(\Gamma)$ and concoct a special substitution principle based on this characterization. Birkedal et al. [Bir+20], for instance, show that in their calculus the only such substitutions are generated by a series of weakenings followed by $L(\gamma)$ for some unique γ. In order to ensure that $\mathsf{unmod}(M)$ admits a substitution principle, it therefore suffices to manually close Rule 5.21 under weakening:

$$\frac{\Gamma \vdash M : \Box A}{L(\Gamma).B_0.\ldots.B_n \vdash \mathsf{unmod}(M) : A[\mathbf{p}^n]}$$

This process yields a calculus which can be implemented, but the process by which it was derived was highly ad-hoc and syntactic. Consequently, the resulting calculus is ill-adapted to work as an internal language. Just because every syntactic substitution $\Delta \longrightarrow L(\Gamma)$ takes the shape described above does not mean that this holds in all models. In fact, such a result holds in hardly any models. More pressingly, this analysis is rather intricate and must be redone as the modalities are altered. The complexity also increases quickly: no rule similar to the above is known for a system which includes both ▶ and □.

We will later explore what hidden structures permits this argument in the syntactic model in order to generalize these arguments in Section 5.6. Before doing so, however, we will discuss the semantics of the rules outlined above and ignore the shortcomings of the actual syntax. We refer to the theory due to Birkedal et al. [Bir+20] extending type theory with a dependent adjunction as DRA.

5.5.2 Models of Fitch-style type theory

A major appeal of Fitch-style type theories is their clean expression in semantics. The additional structure atop of the standard requirements of a model of type theory breaks into two separate components: a functor on the category of contexts to model $L(-)$ and a type former to model modal types.

In order to package up these components, we begin by giving a categorical reformulation of Definition 5.5.3. As it adds no real additional complexity, we generalize to a dependent adjunction linking two models of type theory.

Definition 5.5.4. Given two models of type theory $(\mathcal{C}, \tau_{\mathcal{C}})$ and $(\mathcal{D}, \tau_{\mathcal{D}})$ a dependent adjunction from $\mathcal{C}$ to $\mathcal{D}$ consists of the following data:

1. a functor $L : \mathcal{D} \longrightarrow \mathcal{C}$,
2. and maps $R : F^*\mathcal{T}_{\mathcal{C}} \longrightarrow \mathcal{T}_{\mathcal{D}}$ and $r : F^*\mathcal{T}^\bullet_{\mathcal{C}} \longrightarrow \mathcal{T}^\bullet_{\mathcal{D}}$ assembling into the following:

$$\begin{tikzcd}
L^*\mathcal{T}^\bullet_{\mathcal{C}} \arrow[r, "r"] \arrow[d, "F^*\tau_{\mathcal{C}}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^\bullet_{\mathcal{D}} \arrow[d, "\tau_{\mathcal{D}}"] \\
L^*\mathcal{T}_{\mathcal{C}} \arrow[r, "R"'] & \mathcal{T}_{\mathcal{D}}
\end{tikzcd}$$

Remark 5.5.5. If one unfolds the above definition into the standard language of CwFs, the existence of the above pullback square enforces the existence of a natural bijection between $\mathcal{T}^\bullet(D, R\,A)$ and $\mathcal{T}^\bullet(L(D), A)$ for all $D : \mathcal{D}$ and $A : \mathcal{T}_{\mathcal{C}}(L(D))$ ⋄

Example 5.5.6. *The structure necessary to model the right adjoint in Section 5.4.4 is equivalent to asking for a dependent adjunction from dual-contexts to contexts.*

With this definition in place, we can easily formulate a model of the Fitch-style type theory described above:

Definition 5.5.7. A model of Fitch-style type theory consists of a model of type theory $(\mathcal{C}, \tau)$ together with a dependent adjunction from $(\mathcal{C}, \tau)$ to itself.

5.5.3 The standard model

While a dependent adjunction does not explicitly require an adjunction of contexts, most examples of dependent adjunctions do arise in this manner. We outline here what conditions on a category with display maps are necessary to obtain a dependent adjunction from an ordinary endoadjunction. Accordingly, fix a category with display maps $(\mathcal{C}, \mathscr{D})$ and an adjunction $L \dashv R$ with $L, R : \mathcal{C} \longrightarrow \mathcal{C}$. Our main result is the following:

Theorem 5.5.8. *The local universes construction on* $(\mathcal{C}, \mathscr{D})$ *induces a model of Fitch-style type theory if* R *sends display maps to display maps.*

Remark 5.5.9. This result is a modest improvement over the local universes construction presented by Birkedal et al. [Bir+20]. In particular, our result allows for models in which not all morphisms are considered display maps. The substance of the proof is, however, essentially the same. ◇

In order to establish this result, it suffices to show the following:

Lemma 5.5.10. *The ordinary adjunction* $L \dashv R$ *induces a dependent adjunction if* R *preserves display maps.*

Proof. We use L for the functor on contexts $\mathcal{C} \longrightarrow \mathcal{C}$. It therefore suffices to produce maps R and r fitting into the following diagram:

$$\begin{tikzcd}
L^*\mathcal{T}^\bullet \arrow[r, "r"] \arrow[d, "L^*\tau"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^\bullet \arrow[d, "\tau"] \\
L^*\mathcal{T} \arrow[r, "R"'] & \mathcal{T}
\end{tikzcd}$$

In the above, $\tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T}$ is the representable map induced by the local universes construction.

We define the component of R at $C : \mathcal{C}$ as follows:

$$R_C\big(L(C) \longrightarrow B, E \longrightarrow B\big) = \big(C \longrightarrow R(B), R(E) \longrightarrow R(B)\big)$$

This operation is easily seen to be natural in C as transposition is natural. The definition of r is similar:

$$r_C\big(L(C) \longrightarrow E, E \longrightarrow B\big) = \big(C \longrightarrow R(E), R(E) \longrightarrow R(B)\big)$$

Direct calculation shows that R and r fit into a diagram of the required shape so it remains only to check that this diagram is a pullback. To this end, let us observe that the pullback $(L^*\mathcal{T} \times_{\mathcal{T}} \mathcal{T}^\bullet)(C)$ is comprised of triples:

$$\big\{\big(f : L(C) \longrightarrow B, p : E \longrightarrow B, g : C \longrightarrow R(E)\big) \mid p \circ \widehat{g} = f\big\}$$

The unique gap map induced by r then sends the pair $\big(f : L(C) \longrightarrow E, p : E \longrightarrow B\big)$ to $(p \circ f, p, \widehat{f})$. As this defines a bijection on each component, the gap map is invertible and the diagram is a pullback. □

5.5.4 Conclusions

In fact, all of the modalities mentioned in Section 5.1 form dependent right adjoints. They—along with several others—are considered as motivating examples by Birkedal et al. [Bir+20]. Accordingly, so long as we wish to consider only one modality at a time, Fitch-style type theory can be used effectively with the standard model providing a connection to the intended semantics.

One can also consider extending type theory with a pair of dependent right adjoints to host a pair of modalities. Semantically, there is no issue with this approach. The standard model fluidly adapts to interpret an arbitrary number of dependent adjunctions from $\mathcal{C}$ to itself. While this offers an improvement over dual-context type theories where both the syntax and the semantics where tied to working with a single modality, it does not completely solve the problem.

In particular, the problems posed by the elimination rule cannot be ignored forever. While with only one modality it is possible to find an adhoc adaptation suitable for obtaining a substitution law, it seems a practical impossibility to do this with a pair of modalities. The difficulties only compound if one wishes to allow the modalities to interact as in the case of $\Box$ and $\blacktriangleright$.

Our final step in this chapter, therefore, is to consider the additional structure required to rectify Rule 5.21.

5.6 Parametric adjoints

In this section we describe a solution to the issues presented by Rule 5.21 proposed by Gratzer et al. [Gra+22]. Let us recall the problematic rule:

$$\frac{\Gamma \vdash M : \Box A}{L(\Gamma) \vdash \mathsf{unmod}(M) : A}$$

If we wanted to describe a substitution principle for this rule as-is, we would have to somehow show that an arbitrary substitution $\Delta \longrightarrow L(\Gamma)$ can be presented uniquely as $L(\Delta') \longrightarrow L(\Gamma)$. This issue closely mirrors the original issues discussed in Section 5.3 and, once again, there is no reason to suppose such a presentation exists in general. Unlike Rule 5.4 however, the situation is nearly ideal in the syntactic category for DRA where there was a universal factorization of each substitution into one in the image of $L(-)$. Specifically, given $\Delta \vdash \gamma : L(\Gamma)$ we claimed that γ could be decomposed into a series of weakening operations combined with a substitution $L(\gamma')$ for some unique γ'. In some sense, the paucity of maps into $L(-)$ meant that this problem could be circumvented.

Stepping back from the particular details involving weakening, the core of this result was that given a substitution $\Delta \vdash \gamma : L(\Gamma)$ there existed a context $U(\Delta, \gamma)$ together with a substitution $\Delta \longrightarrow L(U(\Delta, \gamma))$ initial among substitutions $\Delta \longrightarrow L(-)$. This condition looks suspiciously like a requirement that $L(-)$ form a right adjoint, but the universal object $U(\Delta, \gamma)$ is allowed to depend on γ in addition to Δ. This slightly weaker structure is called a parametric right adjoint:

Definition 5.6.1. Given two categories $\mathcal{C}$ and $\mathcal{D}$ such that $\mathcal{C}$ has a terminal object, a parametric right adjoint $G : \mathcal{C} \longrightarrow \mathcal{D}$ is a functor such that the induced functor $\mathcal{C} \longrightarrow \mathcal{D}/G(\mathbf{1})$ is a right adjoint.

Example 5.6.2. *In a category with finite products, $A \times -$ is a parametric right adjoint (see Section 5.6.1).*

In other words, a parametric adjunction $\mathcal{C} \longrightarrow \mathcal{D}$ consists of a normal functor $G : \mathcal{C} \longrightarrow \mathcal{D}$ together with a left adjoint which applies not to $\mathcal{D}$, but to $\mathcal{D}/G(\mathbf{1})$. Returning to our earlier examination of $L(-)$, the observations regarding $L(-)$ in the syntactic category can be summarized as follows:

Theorem 5.6.3. *In the category of (syntactic) contexts and substitutions for* DRA*, $L(-)$ is a parametric right adjoint.*

Remark 5.6.4. There is a small leap from our original discussion to the requirements of a parametric right adjoint; a PRA requires a substitution $\Delta \longrightarrow L(\mathbf{1})$ whereas originally we considered a map $\Delta \longrightarrow L(\Gamma)$ for some Γ. In fact, the coherences required on $U(\Delta, \gamma)$ will allow us to reduce to the case where $\Gamma = \mathbf{1}$. ◇

The proof is carried out by careful induction and analysis of the rules generating substitutions in DRA. A version of Theorem 5.6.3 appears in all discussions for Fitch-style type theories [BGM17; Bir+20; GSB19a]. In fact, all of the careful syntactic analysis carried out in these theories to recover a form of the substitution lemma can be packaged into proving a version of this theorem.

Phrased differently, having a PRA is sufficient to present a tamer version of the elimination rule. We can extend Fitch-style type theory such that $L(-)$ forms the right half of a parametric adjunction and thereby recover a well-behaved calculus. Unlike in the concrete cases above, we will not require its left adjoint ($U(\Delta, \gamma)$) to have any additional properties. In this sense, the passage from DRA to this calculus mirrors the passage from AdjTT to DRA; we are axiomatizing an emergent property of the category of contexts to reduce a syntactic theorem to a definition.

Let us now demonstrate that freely adding a parametric adjunction is sufficient. Taking advantage of the new structure, we reformulate the elimination rule as follows:

$$\frac{U(\Gamma, \rho) \vdash M : \Box A \qquad \Gamma \vdash \rho : L(\mathbf{1})}{\Gamma \vdash \mathsf{unmod}(M, \rho) : A[\eta]} \tag{5.22}$$

$$\frac{\Delta \vdash \gamma : \Gamma \qquad U(\Gamma, \rho) \vdash M : \Box A \qquad \Gamma \vdash \rho : L(\mathbf{1})}{\Delta \vdash \mathsf{unmod}(M, \rho)[\gamma] = \mathsf{unmod}(M[U(\gamma, \rho)], \rho \circ \gamma) : A[L(U(\gamma, r)) \circ \eta]} \tag{5.23}$$

In the conclusion of the first rule, η is the unit of the parametric adjunction. Note that, while not present in our syntax, the unit depends on an object of contexts sliced over $L(\mathbf{1})$. In particular, η depends on both Γ and ρ.

The payoff of this reformulation is in the second rule. The substitution principle now flows directly from the functoriality of $U(-, -)$. The situation is essentially the same as Section 5.5, but now repeated for the elimination rule rather than the introduction rule. We have ensured the existence of universal factorizations and are able to produce a substitution rule from them.

Fortunately, unlike the situation in Section 5.5 the addition of a PRA is sufficient; adapting the elimination rule with this additional structure does not cause problems for any other portions of the theory. So, while a type theory extended with just a dependent adjunction still suffers from a poorly behaved elimination rule, adding an additional parametric adjunction suffices to rectify the theory.

We can rephrase the β and η rules associated with $\Box A$ in terms of this new elimination principle:

$$\frac{\Gamma \vdash \rho : L(\mathbf{1}) \qquad L(U(\Gamma, \rho)) \vdash M : A}{\Gamma \vdash \mathsf{unmod}(\mathsf{mod}(M), \rho) = M[\eta] : A[\eta]} \tag{5.24}$$

$$\frac{\Gamma \vdash M : \Box A}{\Gamma \vdash \mathsf{mod}(\mathsf{unmod}(M[\epsilon], L(!))) = M : \Box A} \tag{5.25}$$

Lemma 5.6.5. *If $L(-)$ is a PRA then Rule 5.21 and Rule 5.22 are inter-derivable.*

Proof. We begin by showing how Rule 5.21 is encoded through Rule 5.22:

$$\frac{\dfrac{\overline{\Gamma \vdash ! : \mathbf{1}}}{L(\Gamma) \vdash L(!) : L(\mathbf{1})} \qquad \dfrac{\Gamma \vdash M : \Box A}{U(L(\Gamma), L(!)) \vdash M[\epsilon] : \Box A[L(\epsilon)]}}{L(\Gamma) \vdash \mathsf{unmod}(M) \triangleq \mathsf{unmod}(M[\epsilon], L(\mathbf{1})) : A}$$

We note that one of the triangle laws of an adjunction is used to ensure that the conclusion has type A; the unit and counit substitutions applied to it cancel out. The β and η laws for $\mathsf{unmod}(M)$ follow from the corresponding principles given to $\mathsf{unmod}(M[\epsilon], L(!))$.

For the reverse, we define $\mathsf{unmod}(M, \rho)$ as follows:

$$\frac{\overline{\Gamma \vdash \rho : L(\mathbf{1})} \qquad \dfrac{\dfrac{U(\Gamma, r) \vdash M : A}{L(U(\Gamma, r)) \vdash \mathsf{unmod}(M) : A}}{\Gamma \vdash \mathsf{unmod}(M)[\eta] : A[\eta]}}{\Gamma \vdash \mathsf{unmod}(M, \rho) \triangleq \mathsf{unmod}(M)[\eta] : A[\eta]}$$

Once again, both the substitution principle and computational equations follow directly from this definition and the corresponding equations on $\mathsf{unmod}(M)$. □

We refer to the type theory extending Fitch-style type theory with a parametric adjunction as *parametric Fitch-style type theory*, shortened to FitchTT following Gratzer et al. [Gra+22].

Remark 5.6.6. We are now able to see why the dependent adjunction for $\mathcal{R}$ in Section 5.4 has not caused the syntactic issues discussed in Section 5.5; $\mathcal{R}$ is a right adjoint so its action on contexts is a parametric right adjoint. The left adjoint to this context action sends $\Delta; \Gamma$ to Δ. In fact, the standard fix for the elimination rule—often justified as 'closing the rule under weakening'—is precisely the specialization of Rule 5.22 in this specific case:

$$\frac{\Delta \vdash M : \mathcal{R}\, A}{\Delta; \Gamma \vdash \mathsf{unmod}_{\mathcal{R}}(M) : A}$$

Notice that the substitution premise has vanished in this case as the context action of $\mathcal{R}$ preserves the terminal context. ◇

5.6.1 Rationalizing parametric adjunctions

While Theorem 5.6.3 and Lemma 5.6.5 assure us that parametric adjunctions arise in practice and offer a clean way to reformulate Fitch-style type theory, they do not give us much in the way of intuition for this extra operation on contexts. Moreover, while $\mathsf{unmod}(M, \rho)$ may behave well as a rule within type theory, it is a surprising to see a substitution appearing directly in a term. While we have introduced parametric adjunctions as a just-so fix for Rule 5.21, there is another path to Rule 5.22 which is potentially more intuitive.

Let us consider a particular dependent right adjoint: $\mathfrak{C} \to -$ for any closed type $\mathfrak{C}$. The left half of this dependent adjunction is given by context extension $L(\Gamma) = \Gamma.\mathfrak{C}$. Consider the rules of a dependent right adjunction specialized to this situation:

$$\frac{\Gamma.\mathfrak{C} \vdash M : A}{\Gamma \vdash \mathsf{mod}(M) : \mathfrak{C} \to A} \qquad \frac{\Gamma \vdash M : \mathfrak{C} \to A}{\Gamma.\mathfrak{C} \vdash \mathsf{unmod}(M) : A}$$

The introduction rule specialized in this situation is immediately recognizable: it is the standard λ-rule used to introduce dependent products. On the other hand, the elimination rule is less standard. It is a version of the application rule, but a highly specialized form. It applies a function to the variable in the context.[4] This rule is at least inter-derivable with the standard application: given a term $\Gamma \vdash N : \mathfrak{C}$, one can construct a substitution $\Gamma \vdash \mathsf{id}.N : \Gamma.\mathfrak{C}$ and then reindex $\mathsf{unmod}(M)$ along this substitution. In fact, terms of $\mathfrak{C}$ in context Γ correspond precisely with substitutions $\Gamma \longrightarrow \mathbf{1}.\mathfrak{C}$ so the typical application rule can be artificially rephrased as follows:

$$\frac{\Gamma \vdash M : \mathfrak{C} \to A \qquad \Gamma \vdash \rho : \mathbf{1}.\mathfrak{C}}{\Gamma \vdash M(\rho) : A[\mathsf{id}.\mathbf{v}[\rho \circ !]]}$$

One final observation is necessary.

Lemma 5.6.7. *The functor $-.\mathfrak{C}$ is a parametric right adjoint; its left adjoint sends $\Gamma \longrightarrow \mathbf{1}.\mathfrak{C}$ to Γ.*

Proof. This is a rephrasing of the well-known fact that pullback is right adjoint to the forgetful functor $\mathcal{C}/C \longrightarrow \mathcal{C}$. □

With this fact to hand, we can see the lightly modified application presented above is precisely Rule 5.22 specialized to this particular dependent adjunction. To crystallize the above into a slogan, dependent adjunctions give us sufficient structure to model the λ-rule and the additional PRA structure is exactly what is required to support the standard application rule. This applies even to standard dependent products, though the role played by parametric adjunctions was not detected until Gratzer et al. [Gra+22] attempted to generalize the application rule to other modalities. This model also offers some intuition for the role played by ρ in $\mathsf{unmod}(M, \rho)$; it is a generalized argument being supplied to M, an element of a generalized function.

[4] This rule is sometimes referred to as the `unlam` rule.

5.6.2 The semantics of parametric Fitch-style type theory

Fortunately, extending the semantics of Fitch-style type theory to account for the PRA structure on contexts requires only minor adaptations. In particular, Lemma 5.6.5 assures us that there is no need to alter the rules governing terms and types. In fact, the only extension needed is to require the left adjoint of the dependent adjunction to be a parametric right adjoint.

Definition 5.6.8. A model of parametric Fitch-style type theory consists of the following:

1. a model of type theory $\mathcal{C}, \tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T} : \mathbf{PSh}(\mathcal{C})$,
2. a dependent adjunction L, R from $(\mathcal{C}, \tau)$ to itself,
3. a functor $U : \mathcal{C}/L(\mathbf{1}) \longrightarrow \mathcal{C}$ shaping L into a parametric right adjoint.

In particular, any model of parametric Fitch-style type theory is a model of Fitch-style type theory and the converse holds if the left adjoint is a PRA. This last point together with the initiality of syntax can be used to quickly derive the conservativity of parametric Fitch-style type theory over ordinary Fitch-style type theory.

Theorem 5.6.9. *A judgment $\mathcal{J}$ of DRA is derivable if and only if the corresponding judgment is derivable in* $\mathsf{FitchTT}$.

Proof. As observed in Theorem 5.6.3, the left half of the dependent adjunction in the syntax of Fitch-style type theory is a PRA. Accordingly, the syntactic model of Fitch-style type theory $\mathcal{S}_{\mathsf{DRA}}$ is a model of parametric Fitch-style type theory. The initiality of syntax for parametric Fitch-style type theory then gives a morphism of syntactic models $f : \mathcal{S}_{\mathsf{FitchTT}} \longrightarrow \mathcal{S}_{\mathsf{DRA}}$.

Conversely, $\mathcal{S}_{\mathsf{FitchTT}}$ is clearly a model of Fitch-style type theory, so we obtain a morphism of models (of Fitch-style type theory) $g : \mathcal{S}_{\mathsf{DRA}} \longrightarrow \mathcal{S}_{\mathsf{FitchTT}}$. We can regard f as a morphism between Fitch-style type theories, whereby $f \circ g = \mathsf{id}$ by initiality. Unfolding, this yields the claimed bi-implication. □

5.6.3 Two interacting modalities through parametric Fitch-style type theory

We are finally in a position to address one of our original motivations in this chapter: a type theory capable of supporting both $\Box$ and $\blacktriangleright$ while allowing them to interact. Essentially, now that we have isolated a set of rules for a single modality which satisfy the necessary substitution principles, we may directly add two modalities.

To this end, we could extend the calculus of contexts and substitutions with two parametric adjunctions $(-/(- : \Box), -.\{\Box\})$ and $(-/(- : \blacktriangleright), -.\{\blacktriangleright\})$. We may then further add modal types with the formation and introduction rules first put forth in DRA along with Rule 5.22 for the elimination principle. We will use $\Box$ for the DRA associated to $-.\{\Box\}$ and $\blacktriangleright$ for the other. We then obtain the following rules among others:

$$\frac{\Gamma.\{\Box\} \vdash M : A}{\Gamma \vdash \mathsf{mod}_{\Box}(M) : \Box A} \qquad\qquad \frac{\Gamma.\{\blacktriangleright\} \vdash M : A}{\Gamma \vdash \mathsf{mod}_{\blacktriangleright}(M) : \blacktriangleright A}$$

This alone yields a workable calculus with two modalities, but neither modality enjoys anything in addition to the standard properties available to any dependent right adjoint—essentially only axiom K.

Ideally, we would ensure that $\blacktriangleright$ has a point, $\Box$ is an idempotent comonad, and that $\Box \circ \blacktriangleright \simeq \Box$. While we could directly postulate these properties, it is most expeditious to force the left adjoints to interact appropriately.

In particular, rather than adding left adjoints for only $\Box$ and $\blacktriangleright$, we will add left adjoints for all their possible composites. By adding equations and further substitutions between these composites, we are able to model all the aforementioned structure. It is helpful to describe this process slightly more abstractly using a small amount of 2-category theory.

Definition 5.6.10. Let $\mathcal{M}$ be the 2-category generated by the following data:

1. One object m.
2. Two morphisms ℓ and b subject to the identifications $b \circ b = b \circ \ell = b$.
3. A pair 2-cells $\mathsf{id} \longrightarrow \ell$ and $b \longrightarrow \mathsf{id}$ such that any pair of 2-cells with the same boundary are equal.

As a matter of convenience, we will write $\mu \leq \nu$ to indicate the (necessarily unique) 2-cell from μ to ν when one exists.

Rather than asking for a pair of functors on the category of contexts representing left adjoints to $\Box$ and $\blacktriangleright$, we will now ask for a 2-functor $F : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$[5] subject to the constraint that $F(m)$ is the category of contexts. Unfolding, this ensures that each 1-cell μ in $\mathcal{M}$ is assigned to a functor in the category of contexts $-.\{\mu\}$ and the 2-cells are assigned to natural transformations between them. Rather than individually specifying the PRA structure and DRAs for these modalities, we may now do so schematically by requiring that for each μ the functor $-.\{\mu\}$ is a PRA with left adjoint $-/(- : \mu)$ with unit and counit η and ϵ. The modalities are then added through the following rules:

$$\frac{\Gamma.\{\mu\} \vdash A\,\mathsf{type}}{\Gamma \vdash \langle \mu \mid A \rangle\,\mathsf{type}} \qquad \frac{\Gamma.\{\mu\} \vdash M : A}{\Gamma \vdash \mathsf{mod}_{\mu}(M) : \langle \mu \mid A \rangle} \qquad \frac{\Gamma \vdash \rho : \mathbf{1}.\{\mu\} \qquad \Gamma/(\rho : \mu) \vdash M : \langle \mu \mid A \rangle}{\Gamma \vdash \mathsf{unmod}_{\mu}(M, \rho) : A[\eta]}$$

In particular, our generating 1-cells of ℓ and f allow us to recover $\blacktriangleright A = \langle \ell \mid A \rangle$ and $\Box A = \langle b \mid A \rangle$. Crucially, no further coherences are needed to relate these modal types or the parametric left adjoints; all relevant structure is encoded by $-.\{-\}$. To see this in action, let us argue that $\Box\blacktriangleright A \simeq \Box A$. We will proceed by constructing a pair of definitionally inverse functions:

$$\begin{aligned}
&l : \Box\blacktriangleright A \to \Box A \\
&l = \lambda\, \mathsf{mod}_b(\mathsf{unmod}_\ell(\mathsf{unmod}_f(\mathbf{v}[\epsilon], !.\{b\})[\epsilon], !.\{\ell\}))
\end{aligned}$$

$$\begin{aligned}
&r : \Box A \to \Box\blacktriangleright A \\
&r = \lambda\, \mathsf{mod}_b(\mathsf{mod}_\ell(\mathsf{unmod}_b(\mathbf{v}[\epsilon], !.\{\ell\})))
\end{aligned}$$

[5] Recall that if $\mathcal{C}$ is a 2-category $\mathcal{C}^{\mathsf{coop}}$ is the 2-category obtained by reversing both the 1- and 2-cells.

Routine computation with the β and η rules for modalities then ensures that these two definitions are mutually inverse.

Remark 5.6.11. Note, however, that this 2-categorical abstraction does not enable us to encode Löb induction. This is intensional; as we shall see in Chapter 9, the addition of Löb induction disrupts normalization while the set of operations expressible using the 2-categorical framework described above does not. Indeed, in Chapter 8 we shall prove that a type theory similarly instrumented by an arbitrary 2-category satisfies normalization irrespective of the choice of 2-category. $\diamond$

To conclude the discussion of this example, we must argue that the standard model of synthetic guarded domain theory in the topos of trees $\mathbf{PSh}(\omega)$ supports a model of the above calculus. In particular, we wish to show that this calculus can be used to capture the salient features of $\mathbf{PSh}(\omega)$ that cannot be modeled in ordinary type theory. We emphasize that while the use of $\mathbf{PSh}(\omega)$ as a model of guarded recursion is well-known, a suitable internal language for carrying out guarded domain theory within $\mathbf{PSh}(\omega)$ is not. The existence of this model, put forward first by Gratzer et al. [Gra+22], is the first incorporate both $\Box$ and $\blacktriangleright$ as dependent right adjoints.

To this end, it suffices to argue that there is a 2-functor $I : \mathcal{M} \longrightarrow \mathbf{Cat}$ such that (1) $I(m) = \mathbf{PSh}(\omega)$ (2) each $I(\mu)$ is a PRA and the left half of a dependent adjunction and (3) the right adjoints of $I(f)$ and $I(\ell)$ are $\Box$ and $\blacktriangleright$, respectively. First, note that by Lemma 5.5.10, it suffices to show (2') that $I(\mu)$ is a left adjoint and a PRA for each μ.

To actually define I, first recall that $\mathbf{PSh}(-)$ induces a 2-functor $\mathbf{Cat}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ sending a functor f to the functor induced by precomposition f^*. We then define $I = \mathbf{PSh}(i(-))$ where $i : \mathcal{M} \longrightarrow \mathbf{Cat}$ is in turn defined as follows:

$$i(m) = \omega \qquad i(\ell)(n) = n + 1 \qquad i(b)(\_) = 0$$

It is clear that $I(m) = \mathbf{PSh}(\omega)$. Moreover, b^* is both a left and right adjoint so (2') is automatically satisfied. The final condition follows more-or-less directly from the definitions of $\Box$ and $\blacktriangleright$ given in e.g. by Birkedal et al. [Bir+12].

Theorem 5.6.12. *$\mathsf{FitchTT}$ induces a modal type theory capable of expressing both $\blacktriangleright$ and $\Box$ while also enjoying the expected interpretation in $\mathbf{PSh}(\omega)$.*

In fact, FitchTT can be parameterized in this manner by an arbitrary 2-category, so that Theorem 5.6.12 becomes a special case of a far more general result concerning models of FitchTT in presheaf categories. This idea is fully developed by Gratzer et al. [Gra+22], but we content ourselves with this special case.

While the above example and op. cit. more generally have shown that FitchTT is a flexible and well-behaved theory *when it is applicable*, we have traveled quite a distance from the situation in Section 5.3. We have gone from requiring a single lex functor preserving display maps to requiring nearly an adjoint triple to present a single modality. Accordingly, while the resulting calculus is quite well-behaved, it is far more restrictive; not every functor worth including in type theory lays at the end of an adjoint triple. While FitchTT may suffice for guarded recursion, we are therefore naturally led to ask whether all of this additional structure is truly necessary to obtain a syntactically well-behaved modal type theory.

5.7 Conclusions

This chapter has been a march through several closely related dependent type theories and we can now see with the benefit of hindsight that each type theory put forward an answer to the following question: what conditions can we impose upon a functor to internalize it within type theory with minimal complications?

For instance, if we assume the modality is fibered, no special alterations to the type theory need to be made. Once we discard this assumption—a necessity for certain examples—we are led to a whole menagerie of different structures.

1. We can simply accept the need to work and manipulate substitutions directly, though doing so sacrifices much of the convenience of type theory.

2. If we restrict attention to an adjunction whose left adjoint is lex, the situation can be improved with AdjTT—at the cost of having a slightly indirect relation between the theory and intended model.

3. If we must include two unrelated modalities, AdjTT will not suffice and—inspecting the required universal properties—we are led to impose the condition that each modality is a dependent right adjoint. However, the resulting syntax is still poorly behaved in general.

4. If we wish to rectify the poor syntactic behavior of plain dependent adjunctions, we can further insist that each left adjoint is a parametric right adjoint. The result is a working modal type theory, though with an admittedly complex syntax.

While each step from (1) to (2) to (3) to (4) is locally reasonable, there is a certain "race to the bottom" globally; we do not wish for the pinnacle of modal type theory to beautifully internalize arbitrary functors (which happen to be isomorphic to the identity).

In this regard, AdjTT is particularly interesting. It includes both halves of an adjunction and presents the left adjoint in a manner superficially similar to the right adjoint (consider the formation and introduction rules) but weakens the elimination rule to avoid requiring an adjoint triple in its semantics. In the next two chapters, we generalize this idea to arrive at the notion of a *weak dependent right adjoint.* The result is a type theory occupying a midpoint between a theory like FitchTT and AdjTT: it can accommodate vastly more general situations when compared with AdjTT, but requires far less of its modalities than FitchTT.

6 MTT: multimodal type theory

> Though it is painful, a certain greater degree of generality is called for.
>
> Dana Scott
> *Advice on modal logic*

In Chapter 5, we surveyed a broad array of modal type theories with each occupying a different point in the design space. Each modal type theory can be broadly summarized by two questions:

1. What semantic assumptions do we make of functors that are to be modalities?
2. How faithfully are these assumptions reflected in the syntax?

For instance, Fitch-style type theories like DRA [Bir+20] intend for modalities to be realized by right adjoints, but this requirement is not fully realized in the syntax: only a dependent right adjoint is required.

In this chapter, we introduce MTT, a theory we put forth as a canonical point in this design space. As with DRA and FitchTT, we intend for modalities to be realized by right adjoints but—unlike the aforementioned theories—the syntax capturing modalities is slightly weaker. While this may appear to be a deficiency, we shall see in future examples that this is an important feature of MTT. Crucially, MTT can incorporate arbitrary interacting right adjoints without imposing further structure—unlike FitchTT—and without undue syntactic burden—unlike DRA. Moreover, we shall see that it is possible to construct models of MTT with functors that are not right adjoints, similar to AdjTT.

We begin in Section 6.1 with an introduction to MTT informally, in the way that it will be written for the remainder of the thesis. In Section 6.2, we discuss the gritty details of the precise syntax of MTT. We conclude the discussion of the theory of MTT in Section 6.3 with a discussion of two key extensions to the base MTT theory: strict internal right adjoints and crisp induction principles. Finally, in Section 6.4 we carry out an extended study of MTT with one particular mode theory to cultivate intuition for working in the theory.

6.1 MTT, informally

Our goal in this section is to introduce MTT as it is used on paper. Firstly, we must note that MTT is not, properly speaking, a type theory. Rather, MTT is a function from abstract descriptions of collections of modalities—*mode theories*—to modal type theories. While we will often blur the line between MTT and a given instantiation, the difference is crucial: all of the theory of MTT, all of the metatheorems, and many of the combinators we develop will apply *irrespective* of the mode theory being considered.

6.1.1 Mode theories

Accordingly, the appropriate place to begin introducing MTT is not with a modal type theory but with a mode theory. For MTT, we will follow Licata and Shulman [LS16] and encode our modal situation as a (strict) 2-category $\mathcal{M}$. Intuitively, a mode theory encodes the data of modal situation as follows:

1. A 0-cell of $\mathcal{M}$ corresponds to a *mode*, a distinct copy of MLTT.
2. A 1-cell $\mu : n \longrightarrow m$ encodes a modality sending types from mode n to types of mode m.
3. A 2-cell $\alpha : \nu \longrightarrow \mu$ encodes a natural transformation—a family of coherent functions—from the modal type induced by ν to that induced μ.

We will further ensure that modal types in MTT 'respect the identity and composition'. That is, the modal type induced by $\mu \circ \nu$ will be equivalent to the composite of the modal types for μ and ν. Likewise, the modal type induced by id will be equivalent to the identity modality.

Using 2-categories to realize mode theories has a number of advantages. For instance, they shall make for convenient tools when describing the categorical semantics in Chapter 7. Even now, however, they allow us to conveniently describe modal situations by fixing a handful of generating 0-, 1-, and 2-cells. The established machinery of 2-categories then provide us with the appropriate mode theory.

Example 6.1.1 (A single endomodality)**.** *It is worth beginning by examining how one encodes the simplest possible modal situation as a mode theory. Suppose we wish to instantiate MTT with a mode theory $\mathcal{M}$ and thereby obtain Martin-Löf type theory extended with one modality and no additional data. As the result should only have one copy of Martin-Löf type theory, $\mathcal{M}$ will only contain one 0-cell m. The modality $\Box$ will be induced by a 1-cell $\mu : m \longrightarrow m$. The lack of additional constraints on the modality ensures that we need not add any 2-cells, but this is not a complete specification of $\mathcal{M}$.*

Indeed, as a 2-category, there must be some composition operator defined on $\mathcal{M}$. Choosing $\mu \circ \mu = \mu$ is valid, but this will not yield a modality with no additional constraints. Imposing $\mu \circ \mu = \mu$ will force the idempotence of $\Box$. We will therefore freely close $\mathcal{M}$ under composition so that it is generated as follows:

$$\hom(m, m) = \{\mu^n \mid n \in \mathbb{N}\}$$

In the above, μ^n representing the n-fold composition of μ with itself so $\mu^n \circ \mu^m = \mu^{n+m}$

Notice that the addition of all these 1-cells does not truly affect the resulting type theory. We will ensure that the modality induced by μ^n is equivalent to the n-fold iteration of the modality induced by n. The system will therefore be a conservative extension of MLTT extended with just a single modality.

Example 6.1.2. *Suppose we wish to extend the prior example so that the resulting instantiation of MTT has a* pointed modality*, a natural transformation from the identity to* $\square$*. This is readily captured by extending the mode theory in the prior example with a generating 2-cell* $\mathsf{id} \longrightarrow \mu$*. Again notice that the existence of a single 2-cell necessitates countably many 2-cells in general: not only* $\mathsf{id} \longrightarrow \mu$ *but also* $\mu^n \longrightarrow \mu^{n+1}$*.*

There are n *distinct 2-cells from* μ^n *to* μ^{n+1}*. We invite the interested reader to calculate a full description of the 2-cells of this mode theory and to provide a single equation that collapses all 2-cells with identical domain and codomain.*

Example 6.1.3 (A monadic modality)**.** *Consider the 2-category generated by a single 0-cell* m*, a single 1-cell* $\tau : m \longrightarrow m$*, and 2-cells* $\eta : \mathsf{id} \longrightarrow \tau$ *and* $\mu : \tau \circ \tau \longrightarrow \tau$ *subject to the following equations:*

$$\mathsf{id} = \mu \circ (\tau \star \eta) = \mu \circ (\eta \star \tau) : \tau \longrightarrow \tau \qquad \textit{(Unit laws)}$$
$$\mu \circ (\mu \star \tau) = \mu \circ (\tau \star \mu) : \tau \circ \tau \circ \tau \longrightarrow \tau \qquad \textit{(Associativity law)}$$

This mode theory describes a single modality τ *structured as a monad; the 2-cells* η *and* μ *serves as the unit and join, respectively. The equations imposed on 2-cells will ensure that (up to propositional equality) the modality will respect the monad laws.*

This mode theory has a recognizable property as a 2-category: it is the walking monad*. A 2-functor out of this mode theory to a 2-category* $\mathcal{C}$ *precisely classifies a monad in* $\mathcal{C}$*. In particular, maps to* **Cat** *classify monads in the standard sense.*

Example 6.1.4 (An idempotent comonadic modality)**.** *Consider now a mode theory with again one 0-cell and one generating 1-cell* m *and* μ *together with generating 2-cells* $\mu \longrightarrow \mathsf{id}$ *and* $\mu \longrightarrow \mu \circ \mu$*. Unlike the prior example, however, we subject this mode theory to the condition that any pair of 2-cells with the same domain and codomain are equal. More precisely, this mode theory is a category enriched over partial orders. The canonical inclusion of posets into categories allows us to regard this as a category enriched over categories i.e., a strict 2-category. As before, this 2-category enjoys a universal property, this time as the walking idempotent comonad.*

Remark 6.1.5. Mode theories arising from poset-enriched categories will prove to be a useful subclass of mode theories. We will refer to them as *posetal mode theories.* ◇

Example 6.1.6 (A pair of adjoint modalities)**.** *For this example, consider the 2-category generated by two 0-cells* m *and* n *and two generating 1-cells* $\mu : n \longrightarrow m$ *and* $\nu : m \longrightarrow n$ *together with a pair of 2-cells* $\epsilon : \nu \circ \mu \longrightarrow \mathsf{id}$ *and* $\eta : \mathsf{id} \longrightarrow \mu \circ \nu$ *subject to the following equations (the triangle identities):*

$$\mathsf{id} = (\epsilon \star \nu) \circ (\nu \star \eta)$$
$$\mathsf{id} = (\mu \star \epsilon) \circ (\eta \star \mu)$$

This mode theory classifies adjunctions. It admits a more concrete definition put forth by Schanuel and Street [SS86] (which can then be specialized to give direct combinatorial descriptions of the prior two examples)

For the remainder of this chapter, we will fix a mode theory $\mathcal{M}$, assuming additional structure upon it as necessary. We will use the letters m, n, and o to range over the 0-cells of $\mathcal{M}$ and refer to them as *modes.* Similarly, μ, ν, and ξ range over 1-cells (*modalities*) and α and β are used for 2-cells.

6.1.2 MTT with a single generating modality

Let us begin by investigating MTT with the mode theory described by Example 6.1.1: the 2-category freely generated one object m and one endomorphism μ. The first portion of MTT is a copy of MLTT for each mode. In this case, there is a single mode (m) so we begin with the standard judgments of Martin-Löf type theory.

The major alteration MTT introduces is through the variables available. When working in MLTT, we construct terms and types relative to a collection of variables $x : A$.[1] In MTT by contrast, each variable contains three distinct pieces of information: the name, the type, and the modal annotation. This last piece is the novel component, and it takes the form of an annotation $x :_{\mu^n} A$. Intuitively, this declaration stipulates that x is available only to terms being used to construct an element of an μ^n-modal type.

In particular, we can use a variable normally just when its annotation is $\mathsf{id} = \mu^0$. If we are given variables $x :_{\mu} \mathsf{Nat}$ and $y :_{\mathsf{id}} \mathsf{Nat}$, only y may be used freely:

Valid	Invalid
$\lambda\_.\, y : \mathsf{Nat} \to \mathsf{Nat}$	$x + y : \mathsf{Nat}$

If all annotations are identities, we recover standard Martin-Löf type theory.

While variables with non-trivial annotations cannot be used to construct elements of Nat or other standard types, they can be used when building elements of modal types $\langle \mu \mid A \rangle$. Specifically, to construct an element of $\langle \mu \mid A \rangle$ one uses the introduction form $\mathsf{mod}_{\mu}(-)$. The rule is that $\mathsf{mod}_{\mu}(M) : \langle \mu \mid A \rangle$ is well-typed with respect to a collection of variables if $M : A$ after updating the variables according to the following pair of rules:

1. All variables with annotation id are removed from scope.
2. A variable $x :_{\mu^{n+1}} A$ is replaced with $x :_{\mu^n} A$

Accordingly, given a variable $x :_{\mu} A$ we can form $\mathsf{mod}_{\mu}(x) : \langle \mu \mid A \rangle$; as we apply $\mathsf{mod}_{\mu}(-)$ the annotation on x is decremented and becomes id, whereby we are able to use it as a normal variable. This principle can be iterated so a variable with annotation μ^3 may be used when constructing $\langle \mu \mid \langle \mu \mid \langle \mu \mid A \rangle\rangle\rangle$.

More generally, recall that our mode theory has more than just the modality μ, it has all composites μ^n. Each composite induces a model type $\langle \mu^n \mid - \rangle$ and each of these has an introduction form $\mathsf{mod}_{\mu^n}(-)$. The rules for $\mathsf{mod}_{\mu^n}(M) : \langle \mu^n \mid A \rangle$ are a generalization of those for $\mathsf{mod}_{\mu}(-)$: all variables annotated with μ^m with $m < n$ are removed and those with annotations $m \geq n$ are updated to have annotation μ^{m-n}. We refer to this process as μ^n-restricting the context.

Fix $x_n :_{\mu^n} \mathsf{Nat}$ for $0 \leq n \leq 2$ for the following examples:

Valid	Invalid
$x_0 : \mathsf{Nat}$	$x_1 : \mathsf{Nat}$
$\mathsf{mod}_{\mu^2}(x_2) : \langle \mu^2 \mid \mathsf{Nat} \rangle$	$\mathsf{mod}_{\mu}(x_2) : \langle \mu \mid \mathsf{Nat} \rangle$
$\mathsf{mod}_{\mu}(\mathsf{mod}_{\mu}(x_2)) : \langle \mu \mid \langle \mu \mid \mathsf{Nat} \rangle\rangle$	$\mathsf{mod}_{\mu^2}(x_1) : \langle \mu^2 \mid \mathsf{Nat} \rangle$

In summary, the introduction principle for modal types allows us to pass from a variable $x :_{\mu^n} A$ to an element of $\langle \mu^n \mid A \rangle$. The role of the elimination principle is to

[1] This is usually crystallized by specifying a context, but we will avoid such descriptions at this level of formality.

allow us to reverse the process to some extent: when constructing a term depending on $M : \langle \mu \mid A \rangle$, we may assume that $M = \mathsf{mod}_\mu(y)$.

More precisely, given a type $B(x)$ depending on $x : \langle \mu^n \mid A \rangle$, we have $\mathsf{let}\ \mathsf{mod}_{\mu^n}(y) \leftarrow M_0\ \mathsf{in}\ M_1 : B(M_0)$ if $M_0 : \langle \mu^n \mid A \rangle$ and under the assumption $y :_{\mu^n} A$ we have $M_1 : B(y)$. This rule enjoys a β principle familiar to these 'pattern-matching' elimination rules:

$$(\mathsf{let}\ \mathsf{mod}_{\mu^n}(y) \leftarrow \mathsf{mod}_{\mu^n}(M_0')\ \mathsf{in}\ M_1) = M_1[M_0'/y]$$

With just this in place, we may construct a few elementary examples of modal programs.

Example 6.1.7 (Axiom K)**.** *We show that each modal type satisfies* axiom K*, a fundamental principle in modal logic amounting to the preservation of finite products. To this end, suppose we are given a pair of variables $x :_{\mathsf{id}} \langle \mu \mid A \rangle$ and $y :_{\mathsf{id}} \langle \mu \mid B \rangle$. Let us construct an element of $\langle \mu \mid A \times B \rangle$ as follows:*

$$F(x) = \mathsf{let}\ \mathsf{mod}_\mu(x') \leftarrow x\ \mathsf{in}\ \mathsf{let}\ \mathsf{mod}_\mu(y') \leftarrow y\ \mathsf{in}\ \mathsf{mod}_\mu((x, y))$$

Written more informally, we begin by destructuring x and y into $\mathsf{mod}_\mu(x')$ and $\mathsf{mod}_\mu(y')$ to shift from "elements of a modal type" to "elements of non-modal type, with annotations". Applying the introduction rule for the modality then removes these annotations and we may construct the necessary element of $A \times B$.

For the reverse direction, suppose we are given $x :_{\mathsf{id}} \langle \mu \mid A \times B \rangle$. We must now construct both $\langle \mu \mid A \rangle \times \langle \mu \mid B \rangle$, so it suffices to construct $\langle \mu \mid A \rangle$ and $\langle \mu \mid B \rangle$ separately. To construct $\langle \mu \mid A \rangle$, we use the elimination rule destructure x as $\mathsf{mod}_\mu(y)$ where $y :_\mu A \times B$. After applying the introduction form for $\langle \mu \mid A \rangle$, it suffices to construct A from $y :_{\mathsf{id}} A \times B$ so that $\pi_1 y$ suffices. The argument for $\langle \mu \mid B \rangle$ is identical, so we content ourselves with recording the entire term:

$$G(x) = \mathsf{let}\ \mathsf{mod}_\mu(y) \leftarrow x\ \mathsf{in}\ (\mathsf{mod}_\mu(\pi_1 y), \mathsf{mod}_\mu(\pi_2 y))$$

Properly speaking, we must also show that F and G are suitably inverse e.g. we must construct an element of $(x : \langle \mu \mid A \times B \rangle) \to \mathsf{Id}(F(G(x)), x)$. Accordingly, let us assume that we are given $x :_{\mathsf{id}} \langle \mu \mid A \times B \rangle$ and immediately destructure x into $\mathsf{mod}_\mu(y)$ where $y :_\mu A \times B$. It now suffices to construct an element of the following:

$$\mathsf{Id}(F(G(\mathsf{mod}_\mu(y))), \mathsf{mod}_\mu(y))$$

Let us now observe the following chain of definitional equalities:

$$\begin{aligned} &F(G(\mathsf{mod}_\mu(y))) \\ &= F((\mathsf{mod}_\mu(\pi_1 y), \mathsf{mod}_\mu(\pi_2 y))) \\ &= \mathsf{mod}_\mu((\pi_1 y, \pi_2 y)) \\ &= \mathsf{mod}_\mu(y) \end{aligned}$$

Accordingly, $\mathsf{refl}(\mathsf{mod}_\mu(y))$ *gives the required term.*

Notation 6.1.8. Axiom K induces a map $\langle \mu \mid A \to B \rangle \to \langle \mu \mid A \rangle \to \langle \mu \mid B \rangle$ which we denote $(\circledast)$.

Example 6.1.9 (The triviality of $\langle \mathsf{id} \mid - \rangle$)**.** *We are now also able to prove one of the* desiderata *of our system:* $\langle \mathsf{id} \mid - \rangle$ *should be the identity modality. More formally, we are able to construct an equivalence between* $\langle \mathsf{id} \mid A \rangle$ *and* A *for any type* A*.*

Let us begin with the forward direction and define $F : \langle \mathsf{id} \mid A \rangle \to A$*. The function can be summarized quite tersely: given* $x :_{\mathsf{id}} \langle \mathsf{id} \mid A \rangle$*, destructure* x *into* $\mathsf{mod}_{\mathsf{id}}(y)$ *and return* y*. Rendered as a term:*

$$F(x) = \mathsf{let}\ \mathsf{mod}_{\mathsf{id}}(y) \leftarrow x\ \mathsf{in}\ y$$

The reverse direction is even more concise: the introduction form $\mathsf{mod}_{\mathsf{id}}(-)$ *suffices.*

That these two operations form an equivalence is equivalent to the β *and propositional* η *principles presupposed for the elimination form.*

While we have proven some nice results already with these rules, they are insufficient to show the promised equivalence $\langle \mu \mid \langle \mu \mid - \rangle \rangle \simeq \langle \mu \circ \mu \mid - \rangle$. Indeed, we cannot even define the forward direction of such an equivalence. It is instructive to see how we get stuck. Fix $x :_{\mathsf{id}} \langle \mu \mid \langle \mu \mid A \rangle \rangle$, the only sensible move at this point is to apply the elimination principle and destructure x into $\mathsf{mod}_\mu(y)$ where $y :_\mu \langle \mu \mid A \rangle$. At this point, however, no sensible maneuver is available. We cannot apply further elimination principles owing to the modal annotation on y and applying the introduction form for $\langle \mu^2 \mid - \rangle$ will cause us to lose access to y entirely.

The solution to this problem is to enable the elimination principle to apply directly to variables with a non-trivial annotation. As an informal intuition, we will introduce a second modality ξ into the elimination principle—the *framing* modality—and allow the term being scrutinized to be available only under ξ.

More precisely, fix a type $B(x)$ depending on $x :_\xi \langle \nu \mid A \rangle$ and suppose we are given a term $M_0 : \langle \nu \mid A \rangle$ in a ξ-restricted context. To construct an element of $B(M_0)$, it suffices to consider construct an element of $B(\mathsf{mod}_\nu(y))$ where $y :_{\xi \circ \nu} A$. The term syntax for this generalized elimination principle is the following:

$$\mathsf{let}_\xi\ \mathsf{mod}_\nu(y) \leftarrow M_0\ \mathsf{in}\ M_1 : B(M_0)$$

For coherence, when $\xi = \mathsf{id}$ we will suppress it to recover the original elimination principle. Just as with the original elimination principle, moreover, there is a β principle associated with this rule:

$$\mathsf{let}_\xi\ \mathsf{mod}_\nu(y) \leftarrow \mathsf{mod}_\nu(M_0')\ \mathsf{in}\ M_1 = M_1[M_0'/y] : B(\mathsf{mod}_\nu(x'))$$

Let us return to the earlier example: a function from $\langle \mu \mid \langle \mu \mid A \rangle \rangle \to \langle \mu^2 \mid A \rangle$. We are now able to use the elimination principle with frame μ to avoid becoming stuck:

$$F(x) = \mathsf{let}\ \mathsf{mod}_\mu(x_0) \leftarrow x\ \mathsf{in}\ \mathsf{let}_\mu\ \mathsf{mod}_\mu(x_1) \leftarrow x_0\ \mathsf{in}\ \mathsf{mod}_{\mu^2}(x_1)$$

We are also able to define an inverse to F:

$$G(x) = \mathsf{let}\ \mathsf{mod}_{\mu^2}(x_0) \leftarrow x\ \mathsf{in}\ \mathsf{mod}_\mu(\mathsf{mod}_\mu(x_0))$$

The pair (F, G) then witnesses the promised equivalence $\langle \mu \mid \langle \mu \mid - \rangle \rangle \simeq \langle \mu^2 \mid - \rangle$. In fact, it is straightforward to generalize F and G to witness the following equivalence for any ν and ξ:

$$\langle \nu \mid \langle \xi \mid - \rangle \rangle \simeq \langle \nu \circ \xi \mid - \rangle$$

From this, we conclude that MTT with this instantiation consists essentially of one new modality $\langle \mu \mid - \rangle$ with all others arising from iterating this construction.

6.1.3 MTT with two modalities

While the prior subsection introduces MTT instantiated with such a simple mode theory, the situation becomes remarkably more complex when we consider more sophisticated instantiations. In this subsection, we will study MTT instantiated with a mode theory generated by a 0-cell m, a pair of 1-cells μ_0, μ_1, and a pair of 2-cells $\alpha, \beta : \mu_0 \longrightarrow \mu_1$.

Crucially, the 2-cells α and β should give rise to two natural transformations $\langle \mu_0 \mid - \rangle \to \langle \mu_1 \mid - \rangle$. Unfortunately, if we mindlessly adapt the rules from the prior subsection (annotate all variables with composites of μ_0 and μ_1) then this operation is not definable. It is again instructive to carry out a doomed attempt. Fix $x :_{\mathsf{id}} \langle \mu_0 \mid A \rangle$. Two possible moves are available: either we can apply the elimination principle for $\langle \mu_0 \mid - \rangle$ or the introduction principle for $\langle \mu_1 \mid - \rangle$. The latter causes us to lose access to x, so we opt for the first and destructure x into $\mathsf{mod}_{\mu_0}(y)$ where $y :_{\mu_0} A$. Unfortunately, at this point we are still lost; all that remains is to apply the introduction rule for $\langle \mu_1 \mid - \rangle$ but this causes us to lose access to both x and y.

Indeed, we require a slightly more refined introduction rule which recognizes that μ_0-variables should be usable for constructing elements of μ_1-modal types. This is not as straightforward as it appears; we cannot simply remove a μ_0 in place of a μ_1 because we must specify which 2-cell is being used to transform the variable, α or β. In fact, this problem is best solved by refusing to solve it.

Rather than annotating each variable with a modality and having μ-restriction strip back this annotation, we annotate each variable with a formal *division* of modalities: $x :_{\mu/\nu} A$. Intuitively, the annotation μ/ν describes a variable originally annotated with μ but which has since been subjected to a ν-restriction. This formal structure enables a succinct definition of restriction: if one ξ-restricts $x :_{\mu/\nu} A$ the result is $x :_{\mu/\nu\circ\xi} A$.

Remark 6.1.10. The rules for when $x :_{\mu/\nu} A$ is well-formed can be somewhat complex to describe when A depends on other variables in the present scope. Far more important than the precise rules, however, is the following observation: one can never arrive at an ill-formed variable declaration through binding variables, applying elimination principles, and restricting contexts. In particular, if one is constructing a term starting in the empty context there is no need to explicitly check that variables are well-formed. ◇

Remark 6.1.11. Applying restrictions is the only way to add to the denominator of an annotation, so the collection of variables in scope can be grouped by the denominator of their annotation and the set of possible denominators will form a chain id, ξ_0, $\xi_0 \circ \xi_1, \dots,$ $\xi_0 \circ \dots \circ \xi_n$. In our given mode theory, for instance, there is no way for both the variables $x :_{\mathsf{id}/\mu_0} \mathsf{Nat}$ and $y :_{\mathsf{id}/\mu_1} \mathsf{Nat}$ to be in scope simultaneously merely through applying the rules of the system starting from a closed context. When we eventually crystallize MTT as a formal system, we will opt for a more efficient representation of the context where the formal divisions are interspersed with variable declarations and apply to all variables that precede them in the context. The reader may find it helpful to keep this in mind in what follows. ◇

We must also revisit the rules for when variables are usable. Previously, $x :_\nu A$ was usable precisely when $\nu = \mathsf{id}$. In our new setting with formal division, a variable $x :_{\mu/\nu} A$ is usable just when there is a 2-cell $\gamma : \mu \longrightarrow \nu$. This 2-cell need not be unique—there are distinct two 2-cells $\mu_0 \longrightarrow \mu_1$—so we record which 2-cell at each use of a variable.

A similar operation can be done on each type: if A is a type in a μ-restricted context and $\gamma : \mu \longrightarrow \nu$ we can annotate each variable to obtain a new type A^γ in a ν-restricted context. In total, given a variable $x :_{\mu/\nu} A$ and a 2-cell $\gamma : \mu \longrightarrow \nu$ we have $x^\gamma : A^\gamma$.

Example 6.1.12. *Consider the type $A = \mathsf{Id}(a_0, a_1)$ where $a_0, a_1 :_{\mu_0/\mu_0} A$, in this situation $A^\alpha = \mathsf{Id}(a_0^\alpha, a_1^\alpha)$. Only one subtlety deserves mention: when computing $\langle \mu \mid A \rangle^\gamma$ one must be sure to whisker by μ so that $\langle \mu \mid A \rangle^\gamma = \langle \mu \mid A^{\gamma \star \mu} \rangle$.*

After making this revision, relatively little is required to specify the rest of the system. The introduction rule is virtually unchanged: $\mathsf{mod}_\mu(M) : \langle \mu \mid A \rangle$ is valid just when $M : A$ holds after μ-restricting the context. The elimination rule is likewise unchanged: $\mathsf{let}_\nu\ \mathsf{mod}_\mu(x) \leftarrow M_0\ \mathsf{in}\ M_1 : B(M_0)$ if (1) $M_0 : \langle \mu \mid A \rangle$ after ν-restricting the context and (2) $M_1 : B(\mathsf{mod}_\mu(x))$ assuming $x :_{\nu \circ \mu} A$. Finally, the β principle is entirely unchanged. The central change required for this generalization is in redefining what it means to μ-restrict a context and in how the variable rule must be adapted afterward.

Example 6.1.13. *Let us now revisit our goal of constructing natural transformations between modal types using α and β. We will handle both simultaneously and more generally by defining a combinator of the following type for each $\gamma : \mu \longrightarrow \nu$:*

$$\mathsf{coe}_\gamma : \langle \mu \mid A \rangle \to \langle \nu \mid A^\gamma \rangle$$

To this end, fix $x :_{\mathsf{id}} \langle \mu \mid A \rangle$ and destructure it into $\mathsf{mod}_\mu(y)$ for $y :_\mu A$. We must construct $\langle \nu \mid A^\gamma \rangle$, but after applying the introduction form we must construct A^γ with $y :_{\mu/\nu} A$. This is precisely the shape of the variable rule, so we are able to complete the construction with y^γ. In total:

$$\mathsf{coe}_\gamma(x) = \mathsf{let}\ \mathsf{mod}_\mu(y) \leftarrow x\ \mathsf{in}\ \mathsf{mod}_\nu(y^\gamma)$$

This combinator is suitably natural in A.

Our earlier proof that $\langle \mu \mid \langle \nu \mid A \rangle \rangle \simeq \langle \mu \circ \nu \mid A \rangle$ as well as the triviality of $\langle \mathsf{id} \mid - \rangle$ transfer verbatim to this new setting:

$$\mathsf{comp}_{\mu;\nu} : \langle \mu \mid \langle \nu \mid A \rangle \rangle \to \langle \mu \circ \nu \mid A \rangle$$
$$\mathsf{comp}_{\mu;\nu}(x) = \mathsf{let}\ \mathsf{mod}_\mu(x_0) \leftarrow x\ \mathsf{in}\ \mathsf{let}_\mu\ \mathsf{mod}_\nu(x_1) \leftarrow x_0\ \mathsf{in}\ \mathsf{mod}_{\mu \circ \nu}(x_1)$$

$$\mathsf{triv} : \langle \mathsf{id} \mid A \rangle \to A$$
$$\mathsf{triv}(x) = \mathsf{let}\ \mathsf{mod}_{\mathsf{id}}(x_0) \leftarrow x\ \mathsf{in}\ x_0$$

Example 6.1.14. *Even without a 2-cell $\mathsf{id} \longrightarrow \mu$, we are able to construct a map $\epsilon : \mathsf{Nat} \to \langle \mu \mid \mathsf{Nat} \rangle$ taking advantage of axiom K and the mapping out property associated with the natural numbers. Let us define ϵ by cases:*

$$\epsilon(0) = \mathsf{mod}_\mu(0)$$
$$\epsilon(1 + n) = \mathsf{let}\ \mathsf{mod}_\mu(n') \leftarrow \epsilon(n)\ \mathsf{in}\ \mathsf{mod}_\mu(1 + n')$$

While this provides a map $\epsilon : \mathsf{Nat} \to \langle \mu \mid \mathsf{Nat} \rangle$, the same technique does not allow us to define an inverse. In particular, $\langle \mu \mid \mathsf{Nat} \rangle$ has no mapping out property![2]

[2] If μ is furnished with a 2-cell $\alpha : \mu \longrightarrow \mathsf{id}$ we would be able to define a retraction to ϵ but not a section.

6.1.4 MTT, generally

There is almost nothing to be done to shift from MTT with a pair of modalities and a pair of 2-cells to MTT with an arbitrary mode theory. The only point which remains to be addressed is how multiple *modes* must be reflected. Fortunately, adding multiple modes does not introduce any truly new rules. Rather, modes are used to limit where modalities may be applied and to which types.[3]

The biggest change is that this version of MTT does not start with MLTT onto which annotated variables and modal types are added. Rather, we begin with an independent copy of MLTT for each mode. That is, we have a notion of types and terms *at mode m* rather than just types and terms. These separate copies of type theory do not interact with each other by default; dependent products at mode m must have a domain and codomain drawn from mode m.

In fact, the only way these different copies of type theory influence each other is through modal types. Given a 1-cell $\mu : n \longrightarrow m$, the type $\langle \mu \mid A \rangle$ is a type at mode m just when A is a type at mode n after μ-restricting the context.

Notice that it is only valid to construct a type or term in mode m using variables that themselves belong at mode m. When specifying $\langle \mu \mid A \rangle$, for instance, we are only able to ask that A is a well-formed type after restricting by μ. Moreover, we are only able to form the modal type $\langle \mu \mid A \rangle$ at mode m.

Remark 6.1.15*.* In general, a variable $x :_{\mu/\nu} A$ belongs at $\mathsf{dom}(\nu)$, though—much as with the well-formedness of $x :_{\mu/\nu} A$ itself—so long as one applies mode-correct rules the collection of variables in scope will always be mode-correct. ◇

Notation 6.1.16. When there is potential ambiguity, we will shorten "$M : A$ at mode m" to $M : A \mathbin{@} m$.

Beyond the issues of ensuring the mode-correctness of types and terms, no alterations are needed to the rules of MTT compared to the prior subsection. For instance, given a pair of modalities $\mu : n \longrightarrow m$ and $\nu : m \longrightarrow o$, the elimination form $\mathsf{let}_\nu\ \mathsf{mod}_\mu(x) \leftarrow M_0\ \mathsf{in}\ M_1$ has type $B(M_0)$ at mode o under the following conditions:

1. $B(x)$ is a type at mode o depending on $x :_\nu \langle \mu \mid A \rangle$.
2. After ν-restricting the context, $M_0 : \langle \mu \mid A \rangle$ at mode m.
3. $M_1(y) : B(\mathsf{mod}_\mu(y))$ assuming $y :_{\nu \circ \mu} A$ at mode o.

6.1.5 Adjoint modalities

Let us consider MTT instantiated with a mode theory containing a pair of adjoint modalities, a pair $\mu : n \longrightarrow m$ and $\nu : m \longrightarrow n$ along with 2-cells $\eta : \mathsf{id} \longrightarrow \mu \circ \nu$ and $\epsilon : \nu \circ \mu \longrightarrow \mathsf{id}$ subject to the triangle equations specified in Example 6.1.6. Using what we have developed so far, we explore the implications of this mode theory.

First, we observe that we have two modal types $\langle \mu \mid - \rangle$ and $\langle \nu \mid - \rangle$ and the 2-cells induce a unit and counit transformation between them:

[3] Category theorists often joke that objects exist merely so morphisms have something to point to, the same is doubly true for modes and modalities.

$$\begin{aligned}&\eta : A \to \langle \mu \mid \langle \nu \mid A^{\eta} \rangle\rangle \\ &\eta = \mathsf{comp}_{\mu;\nu}^{-1} \circ \mathsf{coe}_{\eta} \circ \mathsf{triv}^{-1}\end{aligned}$$

$$\begin{aligned}&\epsilon : \langle \nu \mid \langle \mu \mid A \rangle\rangle \to A^{\epsilon} \\ &\epsilon = \mathsf{triv} \circ \mathsf{coe}_{\epsilon} \circ \mathsf{comp}_{\nu;\mu}\end{aligned}$$

Just as important as the existence of these two functions, the term-level reflection of the unit and counit 2-cells satisfies the triangle identities up to propositional equality.

Lemma 6.1.17. *η and ϵ satisfy the two triangle identities i.e.:*

1. *Given a type A @ n, there exists a term $(x : \langle \mu \mid A \rangle) \to \mathsf{Id}(x, \mathsf{mod}_{\mu}(\epsilon) \circledast \eta(x))$*
2. *Given a type B @ m, there exists a term $(x : \langle \nu \mid B \rangle) \to \mathsf{Id}(x, \epsilon(\mathsf{mod}_{\nu}(\eta) \circledast x))$*

(Recall $(\circledast)$ from Notation 6.1.8.)

Proof. In both cases, the proofs amount to applying the elimination to destructure x and using reflexivity i.e.:

$$\begin{aligned}&\mathsf{tr1}(x) = \mathsf{let}\ \mathsf{mod}_{\mu}(y) \leftarrow x\ \mathsf{in}\ \mathsf{refl}(\mathsf{mod}_{\mu}(y)) \\ &\mathsf{tr2}(x) = \mathsf{let}\ \mathsf{mod}_{\nu}(y) \leftarrow x\ \mathsf{in}\ \mathsf{refl}(\mathsf{mod}_{\nu}(y))\end{aligned}$$

However, this proof is hardly informative as is. To facilitate intuition, we will explicitly compute $\mathsf{mod}_{\mu}(\epsilon) \circledast \eta(x)$ from the first identity after replacing x by $\mathsf{mod}_{\mu}(y)$ to see that it indeed reduces appropriately.

$$\begin{aligned}&\mathsf{mod}_{\mu}(\epsilon) \circledast \eta(\mathsf{mod}_{\mu}(y)) \\ &= \mathsf{mod}_{\mu}(\epsilon) \circledast \mathsf{mod}_{\mu}(\mathsf{mod}_{\nu}(\mathsf{mod}_{\mu}(y^{\eta \star \mu}))) \\ &= \mathsf{mod}_{\mu}(\epsilon(\mathsf{mod}_{\nu}(\mathsf{mod}_{\mu}(y^{\eta \star \mu})))) \\ &= \mathsf{mod}_{\mu}((y^{\eta \star \mu})^{\epsilon}) \\ &= \mathsf{mod}_{\mu}(y^{(\mu \star \epsilon) \circ (\eta \star \mu)}) \\ &= \mathsf{mod}_{\mu}(y)\end{aligned}$$

In particular, in the final step, we apply the appropriate triangle identity which we had previously assumed for 2-cells. □

Remark 6.1.18. The calculations around 2-cells in Lemma 6.1.17 are intimidating at first glance. Fortunately, they can be handled automatically in an implementation. We shall show that type-checking in MTT is decidable provided the underlying mode theory is decidable and the walking adjoint mode theory from Example 6.1.6 is seen to be decidable from the combinatorial description of Schanuel and Street [SS86]. ◇

6.2 MTT, formally

Having cultivated some intuition for how MTT should be used, it is possible to carry on to develop various examples and case studies within MTT. Even in Lemma 6.1.17, however, we have seen that the ever-present subtleties regarding substitutions in modal type theory do arise to some extent in MTT.

$$\frac{}{\vdash \mathbf{1}\,\mathsf{cx} \mathbin{@} m} \qquad \frac{\mu : n \longrightarrow m \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} m}{\vdash \Gamma.\{\mu\}\,\mathsf{cx} \mathbin{@} n}$$

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A\,\mathsf{type} \mathbin{@} n}{\vdash \Gamma.(\mu \mid A)\,\mathsf{cx} \mathbin{@} m}$$

$$\frac{\vdash \Gamma\,\mathsf{cx} \mathbin{@} m}{\vdash \Gamma = \Gamma.\{\mathsf{id}\}\,\mathsf{cx} \mathbin{@} m} \qquad \frac{\mu : n \longrightarrow m \qquad \nu : m \longrightarrow o \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} o}{\vdash \Gamma.\{\nu \circ \mu\} = \Gamma.\{\nu\}.\{\mu\}\,\mathsf{cx} \mathbin{@} n}$$

Figure 6.1: Contexts in MTT

In this section, we will sweat the details and give a precise definition of MTT. In particular, we will present MTT as a particular *generalized algebraic theory (GAT)* [Car78].[4]. The resulting specification gives rise to a fully precise syntax of MTT replete with annotations as well as defining a category of models among which syntax is initial. We will return to the induced category of models in Chapter 7.

Remark 6.2.1. Why have we chosen to specify MTT as a GAT rather than using a more ergonomic logical framework (e.g., that of Nordström et al. [NPS90] or Uemura [Uem21])? Roughly, what makes more modern logical frameworks easier to use than GATs is their handling of binding. They feature some form of function space which is used to express variable binding in the object theory. This frees the user of the need to specify substitutions or contexts of their theory by allowing them to essentially reuse their metatheoretic counterparts. While this story has proven highly effective for standard Martin-Löf style type theories, it cannot be directly adapted for MTT. Contexts in MTT contain an adjoint action induced by modalities and this precludes modeling them using contexts in the metatheory. Without this convenience, these more modern LFs are essentially just as usable as GATs for our purposes.

Researchers have not yet proposed a similarly convenient modal metatheory for specifying type theories such as MTT, arguably because it is still unclear what features such a modal LF would need to support. We leave exploring such possibilities to future work. ◇

As previously discussed, MTT is properly a function from mode theories to type theories. Accordingly, for the remainder of this section, we fix a mode theory $\mathcal{M}$. Each judgment will be rendered as a sort with a distinct copy for each mode $m : \mathcal{M}$. Just as with standard Martin-Löf type theory, the basic judgments are $\vdash \Gamma\,\mathsf{cx} \mathbin{@} m$, $\Gamma \vdash A\,\mathsf{type} \mathbin{@} m$, $\Gamma \vdash M : A \mathbin{@} m$, and $\Gamma \vdash \delta : \Delta \mathbin{@} m$.

6.2.1 Contexts

We begin with contexts which are specified by the three rules in Fig. 6.1.

[4] In fact, MTT can be realized by the slightly simpler structure of a *quotient inductive inductive type* [KKA19]

As a reader may expect, the rules for context formation differ in several ways from MLTT. Most prominently, in addition to the operations for the empty context and extension by a variable, we have $\Gamma.\{\mu\}$ which restricts access to the variables in Γ. Intuitively, $\Gamma.\{\mu\}$ behaves like a left adjoint to the modal type associated with μ as with Fitch-style type theories Section 5.5. We note, in particular, that as a left adjoint $-.\{\mu\}$ acts contravariantly and sends contexts in m to contexts in n when $\mu : n \longrightarrow m$. In Section 6.1, we represented this action by division $-/\mu$ and annotated each variable with the restrictions applied to it. In the GAT syntax, we optimize slightly and record the modal restrictions separately in the context and avoid duplicating them on each variable.

The rule extending a context by a variable is also different from standard Martin-Löf type theory. Intuitively, $\Gamma.(\mu \mid A)$ adds a variable of type A to the context but under the modality μ. For this statement to be well-formed, A must be a type amenable to μ application. This is why A lives a different mode than Γ—n rather than m—and in a different context $\Gamma.\{\mu\}$. In Section 6.1, this was written $\Gamma, x :_\mu A$.

Example 6.2.2. *In total, therefore, an informal set of variables $x :_{\mu/\nu} A$, $y :_{\mathsf{id}/\nu} B$, and $z :_{\mathsf{id}} C$ would be realized by the context $\mathbf{1}.(\mu \mid A).(\mathsf{id} \mid B).\{\nu\}.(\mathsf{id} \mid C)$.*

Finally, unlike Martin-Löf type theory MTT has two non-congruence rules for context equality. These rules ensure that $-.\{-\}$ respect composition in $\mathcal{M}$ so that e.g., restricting by ν then μ is the same as restricting by $\nu \circ \mu$. In the informal syntax, this rule was nearly invisible because restriction was designed to eagerly collapse a sequence of restrictions into a restriction by a composite.

6.2.2 Substitutions

Like most modal type theories, the heart of MTT is in its rules for substitutions. We have specified a representative selection of the rules for $\Gamma \vdash \delta : \Delta \ @ \ m$ in Fig. 6.2.

We have elided two classes of equations from this figure: routine equalities governing e.g., associativity of composition and those equations governing $\delta.M$. We will return to the latter when we have developed terms as they mention various term operations.

Of what remains, the most notable detail is the action of the mode theory on the category of substitutions. Phrased categorically, there is a strict 2-functor from $\mathcal{M}^{\mathsf{coop}}$ which sends the object m to the category contexts and substitutions at mode m. We have already met the action of this 2-functor on objects of these categories $-.\{-\}$. Substitutions are likewise augmented with a functorial action restricting by a modality. In addition, it is precisely here where 2-cells are integrated into MTT. For each 2-cell $\alpha : \mu_0 \longrightarrow \mu_1$ we add a natural transformation between the functors $-.\{\mu_1\} \longrightarrow -.\{\mu_0\}$. Again, notice that as we are working with left adjoints to modalities the variance is flipped; such a natural transformation will eventually induce a transformation from μ_0 modal types to μ_1 modal types.

The sequence of equations at the end of Fig. 6.2 encodes the 2-functoriality equations. While complex, the essence of the equations is to ensure that modal restrictions can be functorially manipulated by 2-cells and that doing so commutes with substitutions being applied to the contexts being restricted.

$$\frac{\vdash \Gamma\,\mathsf{cx} \mathbin{@} m}{\Gamma \vdash \mathsf{id} : \Gamma \mathbin{@} m}$$

$$\frac{\vdash \Gamma_0, \Gamma_1, \Gamma_2\,\mathsf{cx} \mathbin{@} m \qquad \Gamma_0 \vdash \gamma_1 : \Gamma_1 \mathbin{@} m \qquad \Gamma_1 \vdash \gamma_2 : \Gamma_2 \mathbin{@} m}{\Gamma_0 \vdash \gamma_2 \circ \gamma_1 : \Gamma_2 \mathbin{@} m}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \qquad \vdash \Gamma, \Delta\,\mathsf{cx} \mathbin{@} m \\ \Delta.\{\mu\} \vdash A\,\mathsf{type} \mathbin{@} n \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash M : A[\delta.\{\mu\}] \mathbin{@} n \end{array}}{\Gamma \vdash \delta.M : \Delta.(\mu \mid A) \mathbin{@} m}$$

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A\,\mathsf{type} \mathbin{@} n}{\Gamma.(\mu \mid A) \vdash \mathbf{p} : \Gamma \mathbin{@} m}$$

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma, \Delta\,\mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m}{\Gamma.\{\mu\} \vdash \delta.\{\mu\} : \Delta.\{\mu\} \mathbin{@} n}$$

$$\frac{\mu_0, \mu_1 : n \longrightarrow m \qquad \alpha : \mu_0 \longrightarrow \mu_1 \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu_1\} \vdash \Gamma.\{\alpha\} : \Gamma.\{\mu_0\} \mathbin{@} n}$$

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu\} \vdash \mathsf{id} = \mathsf{id}.\{\mu\} : \Gamma.\{\mu\}}$$

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma_0, \Gamma_1, \Gamma_2\,\mathsf{cx} \mathbin{@} m \qquad \Gamma_0 \vdash \gamma_1 : \Gamma_1 \mathbin{@} m \qquad \Gamma_1 \vdash \gamma_2 : \Gamma_2 \mathbin{@} m}{\Gamma_0.\{\mu\} \vdash \gamma_1.\{\mu\} \circ \gamma_0.\{\mu\} = (\gamma_1 \circ \gamma_0).\{\mu\} : \Gamma_2.\{\mu\}}$$

$$\frac{\vdash \Gamma, \Delta\,\mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m}{\Gamma \vdash \delta = \delta.\{\mathsf{id}\} : \Gamma}$$

$$\frac{\mu : n \longrightarrow m \qquad \nu : m \longrightarrow o \qquad \vdash \Gamma, \Delta\,\mathsf{cx} \mathbin{@} o \qquad \Gamma \vdash \delta : \Delta \mathbin{@} o}{\Gamma.\{\nu \circ \mu\} \vdash \delta.\{\nu \circ \mu\} = \delta.\{\nu\}.\{\mu\} : \Gamma.\{\nu \circ \mu\} \mathbin{@} n}$$

$$\frac{\mu_0, \mu_1 : n \longrightarrow m \qquad \alpha : \mu_0 \longrightarrow \mu_1 \qquad \vdash \Gamma, \Delta\,\mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m}{\Gamma.\{\mu_1\} \vdash \Delta.\{\alpha\} \circ \delta.\{\mu_1\} = \delta.\{\mu_0\} \circ \Gamma.\{\alpha\} : \Delta.\{\mu_0\} \mathbin{@} n}$$

$$\frac{\mu_0, \mu_1 : n \longrightarrow m \qquad \nu_0, \nu_1 : m \longrightarrow o \qquad \alpha : \mu_0 \longrightarrow \mu_1 \qquad \beta : \nu_0 \longrightarrow \nu_1 \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} o}{\Gamma.\{\nu_1 \circ \mu_1\} \vdash \Gamma.\{\nu_0\}.\{\alpha\} \circ \Gamma.\{\beta\}.\{\mu_1\} = \Gamma.\{\beta \bullet \alpha\} : \Gamma.\{\nu_0 \circ \mu_0\} \mathbin{@} n}$$

$$\frac{\mu_0, \mu_1, \mu_2 : n \longrightarrow m \qquad \alpha_1 : \mu_0 \longrightarrow \mu_1 \qquad \alpha_2 : \mu_1 \longrightarrow \mu_2 \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu_2\} \vdash \Gamma.\{\alpha_1\} \circ \Gamma.\{\alpha_2\} = \Gamma.\{\alpha_2 \circ \alpha_1\} : \Gamma.\{\mu_0\} \mathbin{@} n}$$

Figure 6.2: Substitutions in MTT

$$\frac{\vdash \Gamma, \Delta \;\mathsf{cx} \mathrel{@} m \qquad \Gamma \vdash \delta : \Delta \mathrel{@} m \qquad \Delta \vdash A \;\mathsf{type} \mathrel{@} m}{\Gamma \vdash A[\delta] \;\mathsf{type} \mathrel{@} m} \qquad \frac{\vdash \Gamma \;\mathsf{cx} \mathrel{@} m}{\Gamma \vdash \mathcal{U}, \mathsf{Nat} \;\mathsf{type} \mathrel{@} m}$$

$$\frac{\vdash \Gamma \;\mathsf{cx} \mathrel{@} m \qquad \Gamma \vdash A \;\mathsf{type} \mathrel{@} m \qquad \Gamma \vdash M, N : A \mathrel{@} m}{\Gamma \vdash \mathsf{Id}(A, M, N) \;\mathsf{type} \mathrel{@} m}$$

$$\frac{\vdash \Gamma \;\mathsf{cx} \mathrel{@} m \qquad \Gamma \vdash A \;\mathsf{type} \mathrel{@} m \qquad \Gamma.(\mathsf{id} \mid A) \vdash B \;\mathsf{type} \mathrel{@} m}{\Gamma \vdash \Sigma(A, B) \;\mathsf{type} \mathrel{@} m}$$

$$\frac{\vdash \Gamma \;\mathsf{cx} \mathrel{@} m \qquad \Gamma \vdash M : \mathcal{U} \mathrel{@} m}{\Gamma \vdash \mathsf{El}(M) \;\mathsf{type} \mathrel{@} m}$$

Figure 6.3: Mode-local types in MTT

6.2.3 Types

Types in MTT decompose into two classes: mode-local types and modal types. Mode-local types take arguments from only a single mode and produce a type at the same mode and their rules are essentially identical to the standard connectives of MLTT. For instance, dependent sums and intensional identity types are translated from MLTT to MTT simply by adding $@\, m$ to each judgment and replacing $\Gamma.A$ with $\Gamma.(\mathsf{id} \mid A)$. As in any type theory, there are multiple ways of encoding universes. For our purposes, we have chosen to specify a Tarski universe. The rules for mode local types of MTT are recorded in Fig. 6.3 though we have elided the standard substitution rules and strict equations governing El.

Remark 6.2.3. It is straightforward to extend MTT with a hierarchy of universes. For the most part, this has no impact on what follows and we leave the details to the interested reader. ◇

Remark 6.2.4. The original work on MTT by Gratzer et al. [Gra+21; Gra+20a] used a Coquand-style universe. Accordingly, the type judgments of the theory were stratified by level and a universe merely internalized the judgment of this level. This presentation is equivalent to what we discuss here. Any model of a Coquand universe is also a model of a Tarski universe and vice versa simply by *defining* level n types to be elements of $\mathcal{U}_n$. ◇

Notation 6.2.5. We will often elide El in terms and computation.

There are two kinds of non-local types: modal types proper and modal function types. The former is specified similarly to dependent right adjoints Section 5.5 using the modal restriction operation previously discussed. Notice that for a modality $\mu : n \longrightarrow m$ because $-.\{\mu\}$ sends contexts in mode m to contexts in mode n, the adjoint modal type $\langle \mu \mid - \rangle$ sends mode n types to mode m types.

Modal function types, on the other hand, originate from Nuyts and Devriese [ND18] and serve as a useful convenience in MTT. Essentially, after generalizing context extension

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A\,\mathsf{type} \mathbin{@} n}{\Gamma \vdash \langle \mu \mid A \rangle\,\mathsf{type} \mathbin{@} m}$$

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma\,\mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A\,\mathsf{type} \mathbin{@} n \qquad \Gamma.(\mu \mid A) \vdash B\,\mathsf{type} \mathbin{@} m}{\Gamma \vdash (\mu \mid A) \to B\,\mathsf{type} \mathbin{@} m}$$

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma, \Delta\,\mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta.\{\mu\} \vdash A\,\mathsf{type} \mathbin{@} n}{\Gamma \vdash \langle \mu \mid A \rangle[\delta] = \langle \mu \mid A[\delta.\{\mu\}] \rangle\,\mathsf{type} \mathbin{@} m}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \\ \vdash \Gamma, \Delta\,\mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta.\{\mu\} \vdash A\,\mathsf{type} \mathbin{@} n \qquad \Delta.(\mu \mid A) \vdash B\,\mathsf{type} \mathbin{@} m \end{array}}{\Gamma \vdash ((\mu \mid A) \to B)[\delta] = (\mu \mid A[\delta.\{\mu\}]) \to B[(\delta \circ \mathbf{p}).\mathbf{v}]\,\mathsf{type} \mathbin{@} m}$$

Figure 6.4: Modal types in MTT

to include modal annotations $\Gamma.(\mu \mid A)$, it is natural to allow dependent products to bind not just elements of A but elements of A subject to some modal annotation: $(\mu \mid A) \to B$. The formation rules for these types are specified in Fig. 6.4.

6.2.4 Terms

Finally, we must specify the term judgment in MTT. The rules specifying terms of mode-local types are identical to those typically found in MTT, so we elide them in this discussion. A notable exception to this pattern is the rule for variables. This must be adapted to account for modal context extensions and modal restrictions. In this presentation of MTT, the variable rule is rather direct: one can access the first variable in the context just when its annotation precisely matches the modal restriction. Intuitively, this rule plays the role of a "counit" for the adjunction between a modal type and its action on contexts. The reader may also compare it with the variable rule presented in Section 6.1 where the annotation and the modality divided it was required to cancel. This rule, along with the introduction and elimination rules for remaining types—modal types and modal dependent products—are specified in Fig. 6.5.

6.3 Possible extensions to MTT

The prior two sections have discussed the syntax of MTT at two levels of precision. The result is a general framework for modal type theory which can be used in a wide number of situations (Part III). There are, however, a variety of possible alternations one can make to MTT to allow it to more conveniently capture a particular situation. For instance, one might replace the intensional identity type with an extensional identity type to obtain extensional MTT or instead postulate the univalence axiom [Uni13]. These additions may disrupt certain properties of MTT e.g. normalization and canonicity, but

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma\ \mathsf{cx} \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathrel{@} n}{\Gamma.(\mu \mid A).\{\mu\} \vdash \mathbf{v} : A[\mathbf{p}.\{\mu\}]}$$

$$\frac{\mu : n \longrightarrow m \qquad \vdash \Gamma\ \mathsf{cx} \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathrel{@} n \qquad \Gamma.\{\mu\} \vdash M : A \mathrel{@} n}{\Gamma \vdash \mathsf{mod}_\mu(M) : \langle \mu \mid A \rangle \mathrel{@} m}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \\ \nu : m \longrightarrow o \qquad \vdash \Gamma\ \mathsf{cx} \mathrel{@} o \qquad \Gamma.\{\nu \circ \mu\} \vdash A\ \mathsf{type} \mathrel{@} n \qquad \Gamma.(\nu \mid \langle \mu \mid A \rangle) \vdash B\ \mathsf{type} \mathrel{@} o \\ \Gamma.\{\nu\} \vdash M_0 : \langle \mu \mid A \rangle \mathrel{@} m \qquad \Gamma.(\nu \circ \mu \mid A) \vdash M_1 : B[\mathsf{id}.\mathsf{mod}_\mu(\mathbf{v})] \mathrel{@} o \end{array}}{\Gamma \vdash \mathsf{let}_\nu\ \mathsf{mod}_\mu(-) \leftarrow M_0\ \mathsf{in}\ M_1 : B[\mathsf{id}.M_0] \mathrel{@} o}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \qquad \vdash \Gamma\ \mathsf{cx} \mathrel{@} m \\ \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathrel{@} n \qquad \Gamma.(\mu \mid A) \vdash B\ \mathsf{type} \mathrel{@} m \qquad \Gamma.(\mu \mid A) \vdash M : B \mathrel{@} m \end{array}}{\Gamma \vdash \lambda M : (\mu \mid A) \to B}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \qquad \vdash \Gamma\ \mathsf{cx} \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathrel{@} n \\ \Gamma.(\mu \mid A) \vdash B\ \mathsf{type} \mathrel{@} m \qquad \Gamma \vdash M_0 : (\mu \mid A) \to B \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash M_1 : A \mathrel{@} m \end{array}}{\Gamma \vdash M_0(M_1) : B[\mathsf{id}.M_1]}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \\ \vdash \Gamma, \Delta\ \mathsf{cx} \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathrel{@} n \qquad \Gamma \vdash \delta : \Delta \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash M : A[\delta.\{\mu\}] \end{array}}{\Gamma.\{\mu\} \vdash \mathbf{v}[\delta.M] = M : A[\delta.\{\mu\}]}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \\ \nu : m \longrightarrow o \qquad \vdash \Gamma\ \mathsf{cx} \mathrel{@} o \qquad \Gamma.\{\nu \circ \mu\} \vdash A\ \mathsf{type} \mathrel{@} n \qquad \Gamma.(\nu \mid \langle \mu \mid A \rangle) \vdash B\ \mathsf{type} \mathrel{@} o \\ \Gamma.\{\nu \circ \mu\} \vdash M_0 : A \mathrel{@} m \qquad \Gamma.(\nu \circ \mu \mid A) \vdash M_1 : B[\mathsf{id}.\mathsf{mod}_\mu(\mathbf{v})] \mathrel{@} o \end{array}}{\Gamma \vdash \mathsf{let}_\nu\ \mathsf{mod}_\mu(-) \leftarrow \mathsf{mod}_\mu(M_0)\ \mathsf{in}\ M_1 = M_1[\mathsf{id}.M_0] : B[\mathsf{id}.M_0] \mathrel{@} o}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \qquad \vdash \Gamma\ \mathsf{cx} \mathrel{@} m \\ \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathrel{@} n \qquad \Gamma.(\mu \mid A) \vdash B\ \mathsf{type} \mathrel{@} m \qquad \Gamma \vdash M : (\mu \mid A) \to B \mathrel{@} m \end{array}}{\Gamma \vdash M = \lambda((M[\mathbf{p}])(\mathbf{v})) : (\mu \mid A) \to B \mathrel{@} m}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \qquad \vdash \Gamma\ \mathsf{cx} \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathrel{@} n \\ \Gamma.(\mu \mid A) \vdash B\ \mathsf{type} \mathrel{@} m \qquad \Gamma \vdash M_0 : (\mu \mid A) \to B \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash M_1 : A \mathrel{@} n \end{array}}{\Gamma \vdash (\lambda M_0)(M_1) = M_0[\mathsf{id}.M_1] : B[\mathsf{id}.M_1] \mathrel{@} m}$$

Figure 6.5: Terms in MTT

the resulting theory remains quite useful for pen-and-paper proofs.[5] Similarly, one may extend each theory with a variety of inductive types, exotic type-constructors, etc

While some of these modifications may alter the behavior of modalities, none of

[5] Indeed, much of Chapter 8 will work within extensional MTT.

them directly change the rules for modal types and so there are no surprises e.g. passing from MTT to extensional MTT when compared with the passage from intensional to extensional MLTT. In this section, we discuss two changes that do directly involve modalities and therefore have no counterpart in standard MLTT: the addition of crisp induction principles and strict dependent right adjoints.

6.3.1 Crisp induction principles for identity types

The first extension serves to precisely describe the identity type $\mathsf{Id}(\langle \mu \mid A \rangle, M_0, M_1)$. In many applications of interest, we construct an identification of two terms $P : \mathsf{Id}(A, N_0, N_1)$ and wish to obtain an identification between $\mathsf{mod}_\mu(N_0)$ and $\mathsf{mod}_\mu(N_1)$ in $\langle \mu \mid A \rangle$. Put tersely, we would like $\langle \mu \mid - \rangle$ to commute with identity types. This is not, however, generally valid. Even though modalities behave like right adjoints and identity types behave like equalizers, these analogies are too imperfect to carry over the familiar categorical proof that right adjoints preserve equalizers. This failure should be understood as a similar phenomenon to the failure of function extensionality in MLTT; it may not hold for the base theory but it does hold in most models of interest and so it is frequently useful to postulate.

Fortunately, forcing modalities to commute with identity types is an easier task than forcing dependent products to commute with identity types. In order to force $\langle \mu \mid - \rangle$ to respect identity types, we can extend the J elimination rule for identity types to a *crisp* induction principle whose scrutinee is typed in context $\Gamma.\{\mu\}$ rather than Γ:

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \qquad \vdash \Gamma\,\mathsf{cx} \mathrel{@} m \\ \Gamma.\{\mu\} \vdash A\,\mathsf{type} \mathrel{@} n \qquad \Gamma.(\mu \mid A).(\mu \mid A[\mathbf{p}]).(\mu \mid \mathsf{Id}(A[\mathbf{p}^2], \mathbf{v}[\mathbf{p}], \mathbf{v})) \vdash B\,\mathsf{type} \mathrel{@} m \\ \Gamma.(\mu \mid A) \vdash M : B[\mathbf{p}.\mathbf{v}.\mathbf{v}.\mathsf{refl}(\mathbf{v})] \mathrel{@} m \\ \Gamma.\{\mu\} \vdash N_0, N_1 : A \mathrel{@} n \qquad \Gamma.\{\mu\} \vdash P : \mathsf{Id}(A, N_0, N_1) \mathrel{@} n \end{array}}{\Gamma \vdash \mathsf{J}_\mu(B, M, P) : B[\mathsf{id}.N_0.N_1.P] \mathrel{@} m} \tag{6.1}$$

$$\frac{\begin{array}{c} \mu : n \longrightarrow m \qquad \vdash \Gamma\,\mathsf{cx} \mathrel{@} m \\ \Gamma.\{\mu\} \vdash A\,\mathsf{type} \mathrel{@} n \qquad \Gamma.(\mu \mid A).(\mu \mid A[\mathbf{p}]).(\mu \mid \mathsf{Id}(A[\mathbf{p}^2], \mathbf{v}[\mathbf{p}], \mathbf{v})) \vdash B\,\mathsf{type} \mathrel{@} m \\ \Gamma.(\mu \mid A) \vdash M : B[\mathbf{p}.\mathbf{v}.\mathbf{v}.\mathsf{refl}(\mathbf{v})] \mathrel{@} m \qquad \Gamma.\{\mu\} \vdash N : A \mathrel{@} n \end{array}}{\Gamma \vdash \mathsf{J}_\mu(B, M, \mathsf{refl}(N)) = M[\mathsf{id}.N] : B[\mathsf{id}.N.N.\mathsf{refl}(N)] \mathrel{@} m} \tag{6.2}$$

Remark 6.3.1. The term "crisp induction" originates from spatial type theory [SS12; Shu18] where certain variables were designated *crisp*. This type theory is essentially a special case of MTT and crisp induction principles as described in Shulman [Shu18] and crisp variables can be generalized to variables with a non-identity modal annotation. ◇

We observe that the addition of a crisp induction principle does actually allow us to characterize $\mathsf{Id}(\langle \mu \mid A \rangle, -, -)$. To prove this we again avail ourselves of informal MTT.

Lemma 6.3.2. *Fix $\mu : n \longrightarrow m$ along with $x, y :_\mu A$ for some $A :_\mu \mathcal{U}$. There is a canonical map $\mathsf{Id}(\langle \mu \mid A \rangle, \mathsf{mod}_\mu(x), \mathsf{mod}_\mu(y)) \to \langle \mu \mid \mathsf{Id}(A, x, y) \rangle$ and this map is invertible in the presence of J_μ.*

Proof. Let us begin by defining the aforementioned map. Accordingly, consider the following type family:

$$B(a : \langle \mu \mid A \rangle, b : \langle \mu \mid A \rangle, \_ : \mathsf{Id}(\langle \mu \mid A \rangle, a, b)) =$$

$$\begin{aligned}&\mathsf{let}\ \mathsf{mod}_\mu(a_0) \leftarrow a\ \mathsf{in}\\&\mathsf{let}\ \mathsf{mod}_\mu(b_0) \leftarrow b\ \mathsf{in}\\&\langle \mu \mid \mathsf{Id}(A, a_0, b_0)\rangle\end{aligned}$$

We may then use the (standard) induction principle on Id with motive B to obtain our canonical map:

$$\begin{aligned}&\phi : \mathsf{Id}(\langle \mu \mid A\rangle, \mathsf{mod}_\mu(x), \mathsf{mod}_\mu(y)) \to \langle \mu \mid \mathsf{Id}(A, x, y)\rangle\\&\phi = \mathsf{J}(B, (x.\ \mathsf{let}\ \mathsf{mod}_\mu(x_0) \leftarrow x\ \mathsf{in}\ \mathsf{mod}_\mu(\mathsf{refl}(x_0))), -)\end{aligned}$$

We will now construct an inverse to ϕ using crisp induction:

```
$\psi : \langle \mu \mid \mathsf{Id}(A, x, y)\rangle \to \mathsf{Id}(\langle \mu \mid A\rangle, \mathsf{mod}_\mu(x), \mathsf{mod}_\mu(y))$
$\psi(p) =$
  let $\mathsf{mod}_\mu(p_0) \leftarrow p$ in
  $\mathsf{J}_\mu($
    $w\, z\, \_.\ \mathsf{Id}(\langle \mu \mid A\rangle, \mathsf{mod}_\mu(w), \mathsf{mod}_\mu(z))$,
    $x.\ \mathsf{refl}(\mathsf{mod}_\mu(x))$,
    $p_0$
  )
```

The proof that these two maps are pointwise inverses follows directly from induction on $\langle \mu \mid A\rangle$ and the computation rules associated with J and J_μ. □

Remark 6.3.3. A converse to this lemma is true up to definitional equality: if the canonical map $\mathsf{Id}(\langle \mu \mid A\rangle, \mathsf{mod}_\mu(x), \mathsf{mod}_\mu(y)) \to \langle \mu \mid \mathsf{Id}(A, x, y)\rangle$ is an equivalence then one can define a term with the appropriate type for J_μ, but the computation rule holds only up to propositional equality. ◇

Finally, we remark that in the presence of extensional equality these crisp induction principles are automatic:

Lemma 6.3.4. *Assuming equality reflection,* $\langle \mu \mid \mathsf{Id}(a, b)\rangle \simeq \mathsf{Id}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b))$.

Proof. We construct a map $\langle \mu \mid \mathsf{Id}(a, b)\rangle \to \mathsf{Id}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b))$. To this end, fix $x :_\mu \mathsf{Id}(a, b)$. We will now show that $\mathsf{mod}_\mu(a)$ is definitionally equal to $\mathsf{mod}_\mu(b)$. By congruence, it suffices to argue that a is convertible with b after restricting the context by μ. Invoking equality reflection, it therefore suffices to construct a proof $\mathsf{Id}(a, b)$, but x is precisely an element of such a type. It is routine to verify that this map is inverse to the canonical map $\mathsf{Id}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b)) \to \langle \mu \mid \mathsf{Id}(a, b)\rangle$. □

Crisp induction, generally

The concept of a crisp induction principle can be applied to types other than identity types. We will consider crisp induction principles for booleans, natural numbers, coproducts, and propositional truncations in due course. In each case, these principles are used to ensure that modalities commute with the subject of the crisp induction principle.

In addition to mode-local types from Martin-Löf type theory, one can also discuss crisp induction principles for MTT-specific types. For instance, we note that modal types in MTT always have crisp induction principles. We are always able to eliminate a variable of type $\langle \mu \mid A \rangle$ even if that variable has a non-identity annotation. This is the role of the framing modality ν in $\mathsf{let}_\nu\ \mathsf{mod}_\mu(-) \leftarrow M_0\ \mathsf{in}\ M_1$. In certain models of MTT we will be forced to consider situations that do not validate these rules. If the elimination rule for μ with frame ν is not validated, we shall say that ν cannot frame μ or, symmetrically, that μ does not have ν-crisp induction principle.

This most often arises with a pair of adjoint modalities $\nu \dashv \mu$ with the left adjoint ν admitting no μ-crisp induction principle. Just as with identity types, crisp induction principles for modal types can be captured through a (suitably definitional) equivalence between two types:

Lemma 6.3.5. *Without the assumption of any crisp induction principles for modalities, the ν-framed elimination principle for $\langle \mu \mid A \rangle$ is inter-derivable with a map $\beta : \langle \nu \mid \langle \mu \mid A \rangle \rangle \longrightarrow \langle \nu \circ \mu \mid A \rangle$ such that $\beta(\mathsf{mod}_\nu(\mathsf{mod}_\mu(M))) = \mathsf{mod}_{\nu\circ\mu}(M)$.*

Proof. The existence of ν-framed elimination for $\langle \mu \mid A \rangle$ has already been shown to be sufficient for defining β, so we turn to the other direction. Fix an inverse β with the specified definitional property and encode ν-framed elimination as follows:

$$\mathsf{let}_\nu\ \mathsf{mod}_\mu(-) \leftarrow M_0\ \mathsf{in}\ M_1 = \mathsf{let}\ \mathsf{mod}_{\nu\circ\mu}(-) \leftarrow \beta(\mathsf{mod}_\nu(M_0))\ \mathsf{in}\ M_1$$

□

6.3.2 Strict dependent right adjoints

We now turn to the second extension: strict dependent right adjoints. Notice that MTT has abandoned the more symmetric elimination principle based on transposition introduced in Section 5.5. In general, adding this principle to MTT without the additional structure discussed in Section 5.6 leads to the same issues that plagued DRA and closely related theories. However, if the modality comes from a right adjoint in the mode theory, Shulman [Shu23] observes that the more complex rules presented in Section 5.6 simplify considerably and can be added to MTT without additional judgmental structure.

Remark 6.3.6. Essentially, the context action $.\{\mu\}$ associated with a right adjoint μ is itself a right adjoint (2-functors preserve right adjoints). This means that it is, in particular, a parametric right adjoint so the ideas of Section 5.6 can be transferred. However, as a full right adjoint some structure degenerates. In particular, as $\mathbf{1}.\{\mu\}$ is terminal half of the data required by the elimination rule in the general case is uniquely determined. ◇

Accordingly, fix $\nu \dashv \mu$. We may extend MTT with *strict dependent right adjoint* for μ through the following rules:

$$\frac{\vdash \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathbin{@} n}{\Gamma \vdash \mu \Rightarrow A\ \mathsf{type} \mathbin{@} m}$$

$$\frac{\vdash \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type} \mathbin{@} n \qquad \Gamma.\{\mu\} \vdash M : A \mathbin{@} n}{\Gamma \vdash \{M\}_\mu : \mu \Rightarrow A \mathbin{@} m}$$

$$\frac{\vdash \Gamma\ \mathsf{cx} \mathbin{@} n \qquad \Gamma.\{\nu \circ \mu\} \vdash A\ \mathsf{type} \mathbin{@} n \qquad \Gamma.\{\nu\} \vdash M : \langle \mu \mid A \rangle \mathbin{@} m}{\Gamma \vdash {!_\mu}\, M : A[\Gamma.\{\epsilon\}] \mathbin{@} n}$$

$$\frac{\vdash \Gamma\ \mathsf{cx} \ @\ m \qquad \Gamma.\{\nu \circ \mu\} \vdash A\ \mathsf{type} \ @\ n \qquad \Gamma.\{\nu \circ \mu\} \vdash M : A \ @\ n}{\Gamma \vdash\ !_{\mu}\,\{M\}_{\mu} = M[\Gamma.\{\epsilon\}] : A[\Gamma.\{\epsilon\}] \ @\ n}$$

$$\frac{\vdash \Gamma\ \mathsf{cx} \ @\ m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type} \ @\ n \qquad \Gamma \vdash M : \mu \Rightarrow A \ @\ m}{\Gamma \vdash \{!_{\mu}\, M[\Gamma.\{\eta\}]\}_{\mu} = M : \mu \Rightarrow A \ @\ m}$$

In these rules, η and ϵ are the 2-cells in $\mathcal{M}$ witnessing respectively the unit and counit of the adjunction $\nu \dashv \mu$.

These rules are interderivable with those of Section 5.5 but the elimination rule satisfies a substitution law. Indeed, one can see this as a specialization of Section 5.6 and so Lemma 5.6.5 applies to show that the above rules are interderivable with the rules shaping $\mu \Rightarrow -$ into a dependent right adjoint.

Finally, we note that the major improvement offered by $\mu \Rightarrow -$ is the final rule giving it an η law. This observation was made first by Nuyts and Devriese [ND21] in the context of extensional MTT where the two types coincide perfectly. We will prove a slight strengthening of their result to apply in intensional MTT and without using μ-crisp induction principles in Section 6.4 (see Corollary 6.4.6) but we note the statement here:

Lemma 6.3.7. *The standard modal type $\langle \mu \mid - \rangle$ satisfies all the rules governing $\mu \Rightarrow -$ with the exception of the η law.*

Corollary 6.3.8. *The standard modal type $\langle \mu \mid - \rangle$ is equivalent to $\mu \Rightarrow -$.*

The extra η law allows us to simplify some aspects of MTT however:

Lemma 6.3.9. *In MTT, the modal dependent products $(\mu \mid A) \to B$ may be encoded using modal types and ordinary dependent products excepting the η law which holds propositionally. If the modality μ admits a strict dependent right adjoint, the η law may be recovered.*

Proof. For the first claim, we encode $(\mu \mid A) \to B$ as follows:

$$C = (\mathsf{id} \mid \langle \mu \mid A \rangle) \to \mathsf{let}\ \mathsf{mod}_{\mu}(-) \leftarrow \mathbf{v}\ \mathsf{in}\ B[(\mathbf{p} \circ \mathbf{p}).\mathbf{v}]$$

We note now that C is equivalent to $(\mu \mid A) \to B$. For instance, given $M : (\mu \mid A) \to B$, we may define $\lambda M[\mathbf{p}](\mathsf{mod}_{\mu}(\mathbf{v}))$. This equivalence is definitional only in one direction so that the β law is validated definitionally but the η law holds only propositionally. If we are given a strict dependent right adjoint the η law also holds definitionally by computation. □

6.4 Adjoint modalities

We conclude this chapter with an in-depth case study of a particularly important mode theory: the adjoint mode theory $\mathcal{M}_{\mathsf{adj}}$ described in Example 6.1.6. Adjoint functors are ubiquitous in category theory and adjoint modalities are nearly as common. Importantly, adjoint modalities mirror many of the properties of adjoint functors and we can prove several results corresponding to classical lemmas about adjoints. We gather several of these results here, both for future use and to demonstrate the mechanics of working in MTT.

For reference, $\mathcal{M}_{\mathsf{adj}}$ contains two modes m, n, two generating 1-cells $\mu : n \longrightarrow m, \nu : m \longrightarrow n$ and two generating 2-cells $\eta : \mathsf{id} \longrightarrow \mu \circ \nu, \epsilon : \mathsf{id} \longrightarrow \mu \circ \nu$. This 2-cells are subject to a pair of equations known as the triangle identities:

$$\begin{tikzcd}
\mu \arrow[r, "\eta \star \mu"] \arrow[dr, "\mathsf{id}_\mu"'] & \mu \circ \nu \circ \mu \arrow[d, "\mu \star \epsilon"] \\
& \mu
\end{tikzcd} \tag{6.3}$$

$$\begin{tikzcd}
\nu \arrow[r, "\nu \star \eta"] \arrow[dr, "\mathsf{id}_\nu"'] & \nu \circ \mu \circ \nu \arrow[d, "\epsilon \star \nu"] \\
& \nu
\end{tikzcd} \tag{6.4}$$

We have already discussed this mode theory at some length in Section 6.1.5. There, we showed that this mode theory induces the expected modalities and natural transformations in MTT e.g.:

$$\eta : A \to \langle \mu \circ \nu \mid A^{\eta} \rangle \tag{6.5}$$

$$\epsilon : \langle \nu \circ \mu \mid A \rangle \to A^{\epsilon} \tag{6.6}$$

We also argued that these combinators also reflect the triangle identities (Lemma 6.1.17).

6.4.1 Internal transposition

We resume our study of this mode theory by discussing other presentations of adjoint mode theories.

In particular, when working in category theory one often uses a natural bijection of hom-sets. Accordingly, one might hope for an equivalence of functions relating $A \to \langle \mu \mid B \rangle$ to $\langle \nu \mid A \rangle \to B$. Unfortunately, these two types do not inhabit the same mode, so they cannot be directly compared. Fortunately, we have two modalities that allow us to move types from mode m to n or vice versa, so two variations of the standard transposition equivalence are well-moded:

$$(A :_{\nu} \mathcal{U})(B :_{\mathsf{id}} \mathcal{U}) \to \langle \nu \mid A \to \langle \mu \mid B \rangle \rangle \simeq (\langle \nu \mid A \rangle \to B^{\epsilon}) \mathbin{@} n$$

$$(A :_{\mathsf{id}} \mathcal{U})(B :_{\mu} \mathcal{U}) \to (A \to \langle \mu \mid B \rangle) \simeq \langle \mu \mid \langle \nu \mid A^{\eta} \rangle \to B \rangle \mathbin{@} m$$

Remark 6.4.1. It would be a mistake to regard the mismatch of modes as the only problem. Even if $m = n$ so that the expected equivalence was well-formed, it would not typically be provable. Essentially the problems are similar to those described in Section 5.1: requiring an internal equivalence corresponds to asking for an isomorphism of exponential objects but frequently only a bijection of hom-sets is valid. This can be glossed over only when the modalities correspond to endofunctors *internally*. That is when there exists a map:

$$(A \to B) \to \langle \mu \mid A \rangle \to \langle \mu \mid B \rangle$$

However, taking $A = \mathsf{Unit}$, this yields a point for both modalities which is often too strong a requirement. ◇

Maps corresponding to both forms of transposition are definable in MTT, though only the second is an equivalence. This reformulation of transposition is not without precedence. It corresponds to a familiar reformulation in category theory available whenever the left adjoint preserves products.

Lemma 6.4.2. *In extensional MTT the following equivalence is inhabited:*[6]

$$(A :_{\mathsf{id}} \mathcal{U})(B :_{\mu} \mathcal{U}) \to (A \to \langle \mu \mid B \rangle) \simeq \langle \mu \mid \langle \nu \mid A^{\eta} \rangle \to B \rangle \mathrel{@} m$$

Proof. Fix $A :_{\mathsf{id}} \mathcal{U}$ and $B :_{\mu} \mathcal{U}$ for the remainder of the proof.

We begin by defining a map $\langle \mu \mid \langle \nu \mid A^{\eta} \rangle \to B \rangle \to (A \to \langle \mu \mid B \rangle)$. To this end, we fix $f :_{\mathsf{id}} \langle \mu \mid \langle \nu \mid A^{\eta} \rangle \to B \rangle$ along with $a :_{\mathsf{id}} A$. We construct an element of $b : \langle \mu \mid B \rangle$ by first applying the unit to a, obtaining $a_0 :_{\mathsf{id}} \langle \mu \mid \langle \nu \mid A \rangle \rangle$ and using axiom K: $b = f \circledast a_0$. In total:

$$F(f, a) = f \circledast \eta\, a$$

Let us now define the putative inverse map. Fix $g :_{\mathsf{id}} A \to \langle \mu \mid B \rangle$. In order to construct an element of $\langle \mu \mid \langle \nu \mid A^{\eta} \rangle \to B \rangle$ we begin by μ-restricting the context so that it suffices to construct $\langle \nu \mid A^{\eta} \rangle \to B$ with a variable $g :_{\mathsf{id}/\mu} A \to \langle \mu \mid B \rangle$. Let us bind $a :_{\nu} A^{\eta}$. Using the counit, it suffices to define an element of $\langle \nu \mid \langle \mu \mid B \rangle \rangle$, which we obtain from $\mathsf{mod}_{\nu}(f^{\eta}(a))$. In total:

$$G(g) = \mathsf{mod}_{\mu}(\lambda a'.\ \mathsf{let}\ \mathsf{mod}_{\nu}(a) \leftarrow a'\ \mathsf{in}\ \epsilon\, \mathsf{mod}_{\nu}(g^{\eta}(a)))$$

It remains to construct identifications between $F \circ G$ and id along with $G \circ F$ and id. Let us begin with $F \circ G$. Using function extensionality, we calculate:

$$\begin{aligned}
&F(G(g))(a)\\
&= G(g) \circledast \eta\, a\\
&= G(g) \circledast \mathsf{mod}_{\mu}(\mathsf{mod}_{\nu}(a^{\eta}))\\
&= \mathsf{mod}_{\mu}(\mathsf{let}\ \mathsf{mod}_{\nu}(a_0) \leftarrow \mathsf{mod}_{\nu}(a^{\eta})\ \mathsf{in}\ \epsilon\, \mathsf{mod}_{\nu}(g^{\eta}(a_0)))\\
&= \mathsf{mod}_{\mu}(\epsilon\, \mathsf{mod}_{\nu}(g^{\eta}(a^{\eta})))\\
&= \mathsf{mod}_{\mu}(\epsilon) \circledast \eta(g\, a) && \text{(Lemma 6.1.17)}\\
&= g\, a
\end{aligned}$$

The reverse direction is similar. Fix $f = \mathsf{mod}_{\mu}(f_0) : \langle \mu \mid \langle \nu \mid A^{\eta} \rangle \to B \rangle$ so $F(f) = \lambda a.\ \mathsf{mod}_{\mu}(f_0(a^{\eta}))$ Note that the following holds by calculation:

$$\begin{aligned}
&G(F(f))\\
&= \mathsf{mod}_{\mu}(\lambda a'.\ \mathsf{let}\ \mathsf{mod}_{\nu}(a) \leftarrow a'\ \mathsf{in}\ \epsilon\, \mathsf{mod}_{\nu}(F(f)^{\eta}(a)))\\
&= \mathsf{mod}_{\mu}(\lambda a'.\ \mathsf{let}\ \mathsf{mod}_{\nu}(a) \leftarrow a'\ \mathsf{in}\ \epsilon\, \mathsf{mod}_{\nu}(F(\mathsf{mod}_{\mu}(f_0^{\eta \star \mu}))(a)))\\
&= \mathsf{mod}_{\mu}(\lambda a'.\ \mathsf{let}\ \mathsf{mod}_{\nu}(a) \leftarrow a'\ \mathsf{in}\ \epsilon\, \mathsf{mod}_{\nu}(\mathsf{mod}_{\mu}(f_0^{\eta \star \mu}(\mathsf{mod}_{\nu}(a^{\nu \star \eta})))))
\end{aligned}$$

[6] Gratzer et al. [Gra+21] claim that only function extensionality is required but both function extensionality and crisp identity induction for μ are necessary.

$$
\begin{aligned}
&= \mathsf{mod}_\mu(\lambda a'.\ \mathsf{let}\ \mathsf{mod}_\nu(a) \leftarrow a'\ \mathsf{in}\ f_0^{(\mu\star\epsilon)\circ(\eta\star\mu)}(\mathsf{mod}_\nu(a^{(\epsilon\star\nu)\circ(\nu\star\eta)})))\\
&= \mathsf{mod}_\mu(\lambda a'.\ \mathsf{let}\ \mathsf{mod}_\nu(a) \leftarrow a'\ \mathsf{in}\ f_0(\mathsf{mod}_\nu(a)))
\end{aligned}
$$

The result follows from Lemma 6.3.4 along with function extensionality. □

6.4.2 Stronger elimination rules for internal right adjoints

Remark 6.4.3. Variants of the results in this subsection were first established for extensional MTT by Nuyts and Devriese [ND21]. ◇

For purely formal reasons $-.\{\nu\} \dashv -.\{\mu\}$. This, in turn, implies several important observations about the behavior of adjoint modalities. For instance, $-.\{\mu\}$ can be "computed" on a context using ν-annotations:

Lemma 6.4.4. *For any context* $\vdash \Gamma\,\mathsf{cx}$*, modality* $\xi : o \longrightarrow m$*, and* $\Gamma.\{\xi\} \vdash A\,\mathsf{type}$*:*

$$\Gamma.(\xi \mid A).\{\mu\} \cong \Gamma.\{\mu\}.(\nu \circ \xi \mid A^{\eta\star\xi})$$

Proof. We will first argue that $\Gamma.(\xi \mid A).\{\mu\}$ and $\Gamma.\{\mu\}.(\nu \circ \xi \mid A^{\eta\star\xi})$ are isomorphic as they represent the same functor. To this end, we make use of the universal property of context extension in MTT: a substitution $\Delta_0 \longrightarrow \Delta_1.(\xi \mid A)$ is determined by (1) a substitution $\Delta_0 \vdash \delta : \Delta_1$ and (2) a term $\Delta_0 \vdash M : A[\delta.\{\xi\}]$.

Fix a context Δ in mode n. Using the above universal property along with transposition, a substitution $\Delta \longrightarrow \Gamma.\{\mu\}.(\nu \circ \xi \mid A^{\eta\star\xi})$ is determined by (1) a substitution $\Delta.\{\nu\} \vdash \gamma : \Gamma$ and (2) a term $\Delta.\{\nu \circ \xi\} \vdash M : A^{\eta\star\xi}[\widehat{\gamma}.\{\nu \circ \xi\}]$ naturally Δ. Unfolding the definition of transposition, $A^{\eta\star\xi}[\widehat{\gamma}.\{\nu \circ \xi\}]$ is simply $A[\gamma.\{\xi\}]$.

Next, a substitution $\Delta \longrightarrow \Gamma.(\xi \mid A).\{\mu\}$ is determined by (1) a substitution $\Delta.\{\nu\} \vdash \gamma : \Gamma$ and (2) a term $\Delta.\{\nu \circ \xi\} \vdash M : A[\gamma.\{\xi\}]$ naturally in Δ.

The two contexts are therefore isomorphic by the Yoneda lemma. □

Theorem 6.4.5. *Given any context* $\vdash \Gamma\,\mathsf{cx}$ *and* $\Gamma.\{\mu\} \vdash A\,\mathsf{type}$*, there is a pair of substitutions*

$$
\begin{gathered}
\Gamma.(\mu \mid A) \vdash \gamma^{\rightarrow} : \Gamma.(\mathsf{id} \mid \langle\mu \mid A\rangle)\\
\Gamma.(\mathsf{id} \mid \langle\mu \mid A\rangle) \vdash \gamma^{\leftarrow} : \Gamma.(\mu \mid A)
\end{gathered}
$$

Moreover, $\gamma^{\leftarrow} \circ \gamma^{\rightarrow} = \mathsf{id}$ *and, if one assumes extensional equality,* $\gamma^{\rightarrow} \circ \gamma^{\leftarrow} = \mathsf{id}$*.*

Proof. One direction of this isomorphism holds regardless of the precise properties of μ:

$$\Gamma.(\mu \mid A) \vdash \gamma^{\rightarrow} \triangleq \mathbf{p}.\mathsf{mod}_\mu(\mathbf{v}) : \Gamma.(\mathsf{id} \mid \langle\mu \mid A\rangle) \tag{6.7}$$

The inverse direction is more subtle:

$$\Gamma.(\mathsf{id} \mid \langle\mu \mid A\rangle) \vdash \gamma^{\leftarrow} \triangleq \mathbf{p}.M : \Gamma.(\mu \mid A) \tag{6.8}$$

Here, M must be a term of the following type:

$$\Gamma.(\mathsf{id}_m \mid \langle\mu \mid A\rangle).\{\mu\} \vdash M : A[\mathbf{p}.\{\mu\}]$$

To define this, consider the following term:

$$\frac{\begin{array}{c}\Gamma.(\mathsf{id}_n \mid \langle \mu \mid A\rangle).\{\mu \circ \nu\} \vdash \mathbf{v}^\eta : \langle \mu \mid A[\{\eta \star \mathsf{id}_\mu\}]\rangle \\ \Gamma.(\mathsf{id}_m \mid \langle \mu \mid A\rangle).\{\mu\}.(\nu \circ \mu \mid A) \vdash \mathbf{v}^\epsilon : A[\mathbf{p}.\{\nu\}]\end{array}}{\Gamma.(\mathsf{id}_m \mid \langle \mu \mid A\rangle).\{\mu\} \vdash M \triangleq \mathsf{let}_\nu\ \mathsf{mod}_\mu(-) \leftarrow \mathbf{v}^\eta\ \mathsf{in}\ \mathbf{v}^\epsilon : A[\mathbf{p}.\{\mu\}]}$$

By computation, we immediately have $\gamma^{\leftarrow} \circ \gamma^{\rightarrow} = \mathsf{id}$. In the reverse direction, we must show that the following terms are definitionally equivalent

$$\Gamma.(\mathsf{id} \mid \langle \mu \mid A\rangle) \vdash \mathbf{v} = \mathsf{mod}_\mu(\mathsf{let}_\nu\ \mathsf{mod}_\mu(-) \leftarrow \mathbf{v}^\eta\ \mathsf{in}\ \mathbf{v}^\epsilon) : \langle \mu \mid A[\mathbf{p}.\{\mu\}]\rangle \tag{6.9}$$

This equation is true *propositionally*, by performing induction on $\mathbf{v}$. Therefore, in the presence of extensional equality this holds definitionally as well. □

In a certain sense, these two results demonstrate that μ induces a weak CwF morphism. The most important consequence of this is the promised relationship between $\langle \mu \mid -\rangle$ and $\mu \Rightarrow -$:

Corollary 6.4.6. *The modal type $\langle \mu \mid -\rangle$ satisfies all the rules of $\mu \Rightarrow -$ except for the η law. This holds only assuming ν-crisp induction principles for $\langle \mu \mid -\rangle$.*

Proof. We begin by defining $!_\mu\, M$ as follows:

$$\frac{\Gamma \vdash M : \langle \mu \mid A\rangle}{\Gamma.\{\mu\} \vdash\ !\, M = \mathbf{v}[\gamma^{\leftarrow}.\{\mu\} \circ \mathsf{id}.M.\{\mu\}] : A}$$

To show the β law holds, we must show the following:

$$\mathbf{v}[(\gamma^{\leftarrow} \circ \mathsf{id}.\mathsf{mod}_\mu(M)).\{\mu\}] = M$$

Let us first rewrite $\mathsf{id}.\mathsf{mod}_\mu(M)$ as $\mathbf{p}.\mathbf{v} \circ \mathsf{id}.M$. We then observe that this is precisely $\gamma^{\rightarrow} \circ \mathsf{id}.M$ whence we have the following:

$$\begin{aligned}&\mathbf{v}[(\gamma^{\leftarrow} \circ \mathsf{id}.\mathsf{mod}_\mu(M)).\{\mu\}]\\ &= \mathbf{v}[(\gamma^{\leftarrow} \circ \gamma^{\rightarrow} \circ \mathsf{id}.M).\{\mu\}]\\ &= M\end{aligned}$$

Finally, the propositional η law holds by induction. □

Remark 6.4.7. We have drawn attention to the lack of need for μ-crisp ν-induction in anticipation of a model in Chapter 7 for MTT with $\mathcal{M}_{\mathsf{adj}}$ which does not validate it. ◇

6.4.3 Crisp induction principles for internal left adjoints

A classic result of elementary category is *right adjoints preserve limits (RAPL)*. We are therefore naturally led to wonder if similar results hold in MTT. Some of the force of this result is undercut by the fact that all MTT modalities preserve finite products and extensional identity types. While some special cases (such as infinite products) are unique to internal right adjoints, in this subsection we primarily consider the dual result: *left adjoints preserve colimits*. In particular, we show that in MTT internal left adjoints preserve booleans and intensional identity types, a fact which is certainly false for arbitrary modalities within MTT.

We begin with the (somewhat academic) fact that internal right adjoints preserve infinitary products.

Remark 6.4.8. Infinitary products should not be conflated with *dependent products* which appear superficially similar and coincide in e.g., **Set**. Infinite products in type theory have an introduction rule with infinitely many premises and therefore no requirement that the assignment from index to term be computable. The existence of all such small infinitary products ensures that the category of contexts is cotensored over **Set** $\diamond$

Lemma 6.4.9. *In an extension of* MTT *with infinite products, internal right adjoints preserve infinite products.*

Proof. Write $\bigotimes_{i \in I} A_i$ for the infinitary product indexed by the set I. Axiom K together with the introduction rule for $\bigotimes_i -$ induces a canonical map of the following form:

$$F(x) = \langle \mathsf{mod}_\mu(\pi_i) \circledast x \rangle_i : \langle \mu \mid \bigotimes_i A_i \rangle \to \bigotimes_i \langle \mu \mid A_i \rangle$$

We will show that this map is invertible. The inverse to this map is given as follows:

$$G(x) = \mathsf{mod}_\mu(\langle \pi_i(!\,x) \rangle_i) : \bigotimes_i \langle \mu \mid A_i \rangle \to \langle \mu \mid \bigotimes_i A_i \rangle$$

Here we have crucially capitalized on the definable $!\,-$ as presented in Corollary 6.4.6. The fact that these two maps are pseudo-inverses follows from a routine calculation and the β and propositional η laws for $!\,-$. □

We now turn to the fact that internal left adjoints preserve colimits. These proofs follow the general recipe outlined in Shulman [Shu18]: first one constructs a crisp induction principle for a type and then one uses this principle to establish the desired equivalence. Accordingly, the key result is that such crisp induction principles are definable for internal left adjoints.

Lemma 6.4.10. *The following crisp induction principle is derivable:*

$$\frac{\begin{array}{c} \Gamma.\{\nu \circ \mu\} \vdash C : (\nu \mid \mathsf{Bool}) \to \mathfrak{U} \mathbin{@} n \\ \Gamma.\{\nu\} \vdash M : \mathsf{Bool} \mathbin{@} m \qquad \Gamma.\{\nu \circ \mu\} \vdash N_0 : C(\mathsf{tt}) \mathbin{@} n \qquad \Gamma.\{\nu \circ \mu\} \vdash N_1 : C(\mathsf{ff}) \mathbin{@} n \end{array}}{\Gamma \vdash \mathsf{if}_\nu(C, M, N_0, N_1) : C^\epsilon(M) \mathbin{@} n}$$

Furthermore, this term satisfies the expected β *rules e.g.* $\mathsf{if}_\nu(C, \mathsf{tt}, N_0, N_1) = N_0$

Proof. We begin by constructing a "helper function" of the following type:

$$\Gamma.\{\nu\} \vdash h : (x :_{\mathsf{id}} \mathsf{Bool}) \to \langle \mu \mid C(\mathsf{tt}) \rangle \to \langle \mu \mid C(\mathsf{ff}) \rangle \to \langle \mu \mid C(x^\eta) \rangle$$

The definition of h is given using the standard induction principle for booleans:

$$h(\mathsf{tt}, y_0, y_1) = y_0$$
$$h(\mathsf{ff}, y_0, y_1) = y_1$$

To be explicit, the motive of this induction is $\Gamma.\{\nu\}.(\mathsf{id} \mid \mathsf{Bool}) \vdash \langle \mu \mid C(\mathbf{v}^{\eta}) \rangle\ \mathsf{type}$.

With h to hand, we are now able to define if_{ν}:

$$\mathsf{if}_{\nu}(C, M, N_0, N_1) = \epsilon(\mathsf{mod}_{\nu}(h(M, \mathsf{mod}_{\mu}(N_0), \mathsf{mod}_{\mu}(N_1))))$$

That this term has the appropriate type requires some consideration. Inspecting the type of h, we see that this term has type $(C(M^{\nu \star \eta}))^{\epsilon}$ and one triangle identities then ensures that this is equal to $C^{\epsilon}(M)$ as required.

The β rules follow more-or-less directly from the β rules associated with if. □

Lemma 6.4.11. *$\langle \nu \mid - \rangle$ preserves booleans i.e. $\langle \nu \mid \mathsf{Bool} \rangle \simeq \mathsf{Bool}$.*

Proof. For any modality in MTT, there is a canonical map $\mathsf{Bool} \to \langle \nu \mid \mathsf{Bool} \rangle$:

$$F(x) = \mathsf{if}(\langle \nu \mid \mathsf{Bool} \rangle, x, \mathsf{mod}_{\nu}(\mathsf{tt}), \mathsf{mod}_{\nu}(\mathsf{ff}))$$

To define the inverse map, we require the crisp version of if:

$$G(x) = \mathsf{let}\ \mathsf{mod}_{\nu}(x_0) \leftarrow x\ \mathsf{in}\ \mathsf{if}_{\nu}((\lambda \_ .\, \mathsf{Bool}), x, \mathsf{tt}, \mathsf{ff})$$

To verify that these maps assemble into an inverse, we use (crisp) induction on booleans along with the β rules of if and if_{ν}. □

Corollary 6.4.12. *Internal left adjoints preserve coproducts.*

Proof. Coproducts can be encoded by dependent sums and booleans, both of which are preserved by internal left adjoints (in particular, dependent sums are preserved by all MTT modalities). □

Lemma 6.4.13. *Intensional identity types are preserved by $\langle \nu \mid - \rangle$. If $\Gamma.\{\nu\} \vdash a_0, a_1 : A \mathrel{@} m$ then there is a canonical equivalence $\mathsf{Id}(\mathsf{mod}_{\nu}(a_0), \mathsf{mod}_{\nu}(a_1)) \simeq \langle \nu \mid \mathsf{Id}(a_0, a_1) \rangle$.*

Proof. We will prove this using a similar recipe to the booleans. To begin with, consider the following variation on crisp identity induction (compared with Rule 6.1 the motive is placed in a more restrictive context):

$$\frac{\begin{array}{c}\Gamma.\{\nu\} \vdash A\ \mathsf{type} \mathrel{@} m \\ \Gamma.\{\nu \circ \mu\} \vdash C : (a_0, a_1 :_{\nu} A^{\nu \star \eta})(p :_{\nu} \mathsf{Id}(a_0, a_1)) \to \mathcal{U} \mathrel{@} n \\ \Gamma.\{\nu \circ \mu\} \vdash M : (a :_{\nu} A^{\nu \star \eta}) \to B(a, a, \mathsf{refl}) \mathrel{@} n \\ \Gamma.\{\nu\} \vdash N_0, N_1 : A \mathrel{@} m \qquad \Gamma.\{\nu\} \vdash P : \mathsf{Id}(A, N_0, N_1) \mathrel{@} m\end{array}}{\Gamma \vdash \mathsf{J}'_{\nu}(B, M, P) : B^{\epsilon}(N_0, N_1, P) \mathrel{@} n}$$

As before, we define a helper function in context $\Gamma.\{\nu\}$:

$$h : (a_0, a_1 :_{\mathsf{id}} A)(p :_{\mathsf{id}} \mathsf{Id}(a_0, a_1))(f :_{\mu} (a :_{\nu} A^{\nu \star \eta}) \to C(a, a, \mathsf{refl})) \to \langle \mu \mid C(a_0^{\eta}, a_1^{\eta}, p^{\eta}) \rangle$$

We define h through standard identity induction:

$$h(a, a, \mathsf{refl}, f) = \mathsf{mod}_{\mu}(f(a^{\eta}))$$

With h to hand, we define J'_{ν} as follows:

$$\mathsf{J}'_{\nu}(C, M, P) = \epsilon(\mathsf{mod}_{\nu}(h(N_0, N_1, P, M)))$$

Once again, careful calculation with the triangle identities ensures that this is well-typed.

Finally, it remains to derive the promised equivalence. The interesting direction is the map $G : \langle \nu \mid \mathsf{Id}(a_0, a_1)\rangle \to \mathsf{Id}(\mathsf{mod}_\nu(a_0), \mathsf{mod}_\nu(a_1))$. There is some remaining subtlety, as we must ensure that the motive is well-defined:

$$G(x) = \mathsf{J}'_\nu((\lambda a_0, a_1, \_.\ \mathsf{Id}_{\langle \mu \mid A^{\nu \star \eta}\rangle}(\mathsf{mod}_\nu(a_0), \mathsf{mod}_\nu(a_1))), (\lambda a.\ \mathsf{refl}(a)), x)$$

In particular, we must check that $\langle \mu \mid A^{\nu \star \eta}\rangle^\epsilon = \langle \mu \mid A\rangle$, but this follows from a triangle identity. The inverse map and the computations ensuring that they are equivalences are all routine. □

Corollary 6.4.14. *Rule 6.1 is derivable for ν though the β law may hold only propositionally.*

Proof. By virtue of Remark 6.3.3 and Lemma 6.4.13. □

The following lemma is proven by essentially identical means to the three prior results.

Lemma 6.4.15. *The type of natural numbers is preserved by $\langle \nu \mid - \rangle$.*

Proof. We will only sketch this proof, as it follows the same template as those described above. We will first define a crisp induction principle for Nat from which the lemma follows directly. Let us fix $\Gamma.(\nu \mid \mathsf{Nat}) \vdash C\,\mathsf{type}$. We begin by constructing a helper function of the following type:

$$\Gamma.\{\nu\} \vdash h : (x : \mathsf{Nat}) \to \langle \mu \mid C\,\mathsf{z}\rangle \to \langle \mu \mid (n :_\nu \mathsf{Nat}) \to C\,n^\eta \to C(\mathsf{suc}\,n^\eta)\rangle \to \langle \mu \mid C\,x^\eta\rangle$$

As before, this function is defined using the ordinary elimination principle for Nat:

$$\begin{aligned}&h(\mathsf{z}, z, s) = z\\ &h(\mathsf{suc}\,n, z, s) = s \circledast \eta\,n \circledast h(n, z, s)\end{aligned}$$

We may then transpose to define a crisp version of rec:

$$\mathsf{rec}_\nu(C, z, s, n) = \epsilon(\mathsf{mod}_\nu(h\,n\,\mathsf{mod}_\mu(z)\,\mathsf{mod}_\mu(s)))$$

The desired equivalence is then proven from rec_ν. □

We will eventually be interested in the behavior of internal left adjoints on strict propositional truncations[7] so we record the following for future use.

Given a type A @ m, the truncation $|A|$ is another type at mode m such equipped with a map $\eta : A \to |A|$ such that any map $f : A \to B$ together with an identification $(a_0, a_1 : A) \to \mathsf{Id}(f\,a_0, f\,a_1)$ induces an extension $|A| \to B$ along η. Written formally, we have the following elimination principle:

$$\frac{\begin{array}{c}\Gamma \vdash M : |A| \mathbin{@} m\\ \Gamma.(\mathsf{id} \mid A) \vdash N : B[\mathsf{id}.\eta\,\mathbf{v}] \mathbin{@} m \qquad \Gamma.(\mathsf{id} \mid A).(\mathsf{id} \mid A) \vdash P : \mathsf{Id}(N[\mathbf{p}], N[\mathbf{p}^2.\mathbf{v}]) \mathbin{@} m\end{array}}{\Gamma \vdash \mathsf{unpack}(N, P, M) : B[\mathsf{id}.M]}$$

$$\frac{\begin{array}{c}\Gamma \vdash M : A\\ \Gamma.(\mathsf{id} \mid A) \vdash N : B[\mathsf{id}.\eta\,\mathbf{v}] \qquad \Gamma.(\mathsf{id} \mid A).(\mathsf{id} \mid A) \vdash P : \mathsf{Id}(N[\mathbf{p}], N[\mathbf{p}^2.\mathbf{v}]) \mathbin{@} m\end{array}}{\Gamma \vdash \mathsf{unpack}(N, P, \eta\,M) = N[\mathsf{id}.M] : B[\mathsf{id}.M]}$$

[7] In fact, we shall only be interested in this in the case of extensional MTT.

Lemma 6.4.16. *If an internal right adjoint preserves identity types, the corresponding internal left adjoint preserves truncations when they exist.*

Proof. As before, we prove this by constructing a crisp version of the elimination principle for $|A|$:

$$\frac{\begin{array}{c}\Gamma.\{\nu\} \vdash A\,\mathsf{type} \mathbin{@} m \qquad \Gamma.\{\nu \circ \mu\} \vdash B : (a :_{\nu} |A^{\nu\star\eta}|) \to \mathcal{U} \mathbin{@} n \\ \Gamma.\{\nu \circ \mu\} \vdash N : (a :_{\nu} |A^{\nu\star\eta}|) \to B(\eta\, a) \mathbin{@} n \\ \Gamma.\{\nu \circ \mu\} \vdash P : (a_0, a_1 :_{\nu} |A^{\nu\star\eta}|) \to \mathsf{Id}(N(a_0), N(a_1)) \mathbin{@} n \qquad \Gamma.\{\nu\} \vdash M : |A| \mathbin{@} m\end{array}}{\Gamma \vdash \mathsf{unpack}'(M, N) : B^{\epsilon}(M) \mathbin{@} n}$$

Once more we proceed by constructing a helper function h at mode m in context $\Gamma.\{\nu\}$ with the following type:

$$\begin{aligned}\Gamma.\{\nu\} \vdash & h : (a :_{\mathsf{id}} |A|)(f :_{\mu} (a :_{\nu} |A^{\nu\star\eta}|) \to B(\eta\, a)) \\ & \quad \to (p :_{\mu} (a_0, a_1 :_{\nu} |A^{\nu\star\eta}|) \to \mathsf{Id}(f(a_0), f(a_1))) \to \langle \mu \mid B(a^{\eta}) \rangle\end{aligned}$$

We are able to once again define h in a straightforward manner using unpack (writing ι for the map $\langle \mu \mid \mathsf{Id}(x, y) \rangle \to \mathsf{Id}(\mathsf{mod}_{\mu}(x), \mathsf{mod}_{\mu}(y))$):

$$\Gamma.\{\nu\} \vdash h(t, f, p) = \mathsf{unpack}(a.\, \mathsf{mod}_{\mu}(f(a^{\eta})), a_0, a_1.\, \iota(\mathsf{mod}_{\mu}(p\, a_0^{\eta}\, a_1^{\eta})), t)$$

For readability, we have written unpack using named binders.

Finally, we define the crisp principle as follows:

$$\mathsf{unpack}'(N, P, M) = \epsilon(\mathsf{mod}_{\nu}(h(M, N, P)))$$

□

Remark 6.4.17. Requiring the right adjoint preserves identity types in the above is an exception to the general pattern that left adjoints preserve types with a mapping out property. It is, however, clearly necessary. In Shulman [Shu18], a similar construction is used; in that setting the right adjoint is shown to preserve identity types using a standard argument for idempotent monads unavailable for general MTT modalities. ◇

7 Semantics of MTT

[W]hen everything comes from the sky, this convinces nobody.

Jean-Yves Girard
The Blind Spot

In this chapter, we investigate the semantics of MTT. We note, however, that many of the fundamental questions about the model theory of MTT have already been answered in Section 6.2. By defining MTT as a generalized algebraic theory, we have automatically inherited a definition of a model and, more broadly, a category of models. Furthermore, the same machinery ensures that (fully-annotated) syntax is initial in this category.

This does not mean that nothing remains to be done. For instance, while generalized algebraic theories yield a particular description of a model, it is hardly the most convenient. In Section 7.1 we repackage the definition into a more recognizable structure based on natural models [Awo18]. In the process, we isolate the semantic structure of MTT into a weakening of dependent right adjoints we term *weak dependent right adjoints*.

We have carefully arranged for modalities in MTT to have an elimination rule which respects substitution. The result is considerably more complex than the transposition rule available for dependent right adjoints (DRAs) (Sections 5.5 and 6.3) and it is usually easier to construct a DRA semantically. In Section 7.2 we show that DRAs to least model MTT and present several recognition principles for them in practice. In particular, we combine these two results to show that MTT admits models in any 2-diagram of topoi connected by right adjoints.

This raises an interesting question: given that all of the semantic models of MTT encountered at this point use dependent right adjoints rather than the weaker notion of modality present in MTT, do non-syntactic models of weak dependent right adjoints even exist? In Section 7.3 we construct a non-syntactic weak dependent right adjoint based on the gluing model first discussed in Section 5.4. Interestingly, this model still supports equality reflection, showing that weak dependent right adjoints with equality reflection are still weaker than full dependent right adjoints. We further show that the other extension to MTT discussed in Section 6.3—crisp induction principles—are independent of MTT as well.

7.1 Natural models of MTT

Throughout this section, we fix a mode theory $\mathcal{M}$ and consider a model of MTT instantiated with $\mathcal{M}$. We describe the data of a model of MTT as a series of components.

7.1.1 Contexts and substitutions

The basis for a model of MLTT is its *category of contexts*. We wish to adapt this idea to MTT. As a warm-up, let us consider first the syntax of MTT. Contexts and substitutions *at each mode* organize into a category. Rather than a single unified category of contexts, we instead have a category Cx_m associated with each mode $m : \mathcal{M}$.

In fact, there is far more structure than this available. For each modality $\mu : n \longrightarrow m$, the operation $\Gamma \mapsto \Gamma.\{\mu\}$ induces a functor $L(\mu) : \mathsf{Cx}_m \longrightarrow \mathsf{Cx}_n$. The definitional equalities $\vdash \Gamma.\{\nu \circ \mu\} = \Gamma.\{\nu\}.\{\mu\}\,\mathsf{cx}$ ensures that these functors satisfy $L(\mu) \circ L(\nu) = L(\nu \circ \mu)$. Similarly, the equations forcing $\Gamma.\{\mathsf{id}\} = \Gamma$ and $\gamma.\{\mathsf{id}\} = \gamma$ force $L(\mathsf{id})$ to be the identity functor. In addition to these equations, 2-cells in MTT $\alpha : \nu \longrightarrow \mu$ induce natural transformations $L(\alpha) : L(\mu) \longrightarrow L(\nu)$. Finally, a series of definitional equalities in MTT ensure that the assignment $\alpha \mapsto L(\alpha)$ is suitably functorial.

We can summarize this data:

Lemma 7.1.1. *Contexts and substitutions in MTT form a 2-functor $\mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$.*

This leads to the first component of a model of MTT:

Component 7.1.1. *A model of contexts of MTT consists of a 2-functor $\mathcal{I} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$.*

Immediately, however, we have additional requirements that should be imposed upon this. For instance, given a model of contexts $\mathcal{I}$ each category of contexts $\mathcal{I}(m)$ should support the empty context. This can be expeditiously specified by requiring each $\mathcal{I}(m)$ have a terminal object:

Component 7.1.2. *A model of contexts $\mathcal{I}$ supports the empty context if each $\mathcal{I}(m)$ has a terminal object.*

Remark 7.1.2. Notice that we have separated Component 7.1.2 explicitly rather than asking for $\mathcal{I}$ to be valued in categories with terminal objects. Indeed, the category of categories with terminal objects is typically spanned by functors that preserve the terminal object, but $\mathcal{I}(\mu)$ need not do this. Within syntax, we have not required $\mathbf{1}.\{\mu\} = \mathbf{1}$ and several natural models fail to validate this equation. ◇

Thus far we have specified what a model must support to interpret (1) the category structure of contexts and substitutions (the identity, composition, etc.) (2) empty context and the universal maps onto it (3) modal restriction and the action of 2-cells on modal restriction. To proceed further, however, we require some interpretation of terms and types.

7.1.2 Types, terms, and context extension

Just as with a model of MLTT, terms and types in MTT can be organized into presheaves over the relevant category of contexts. Moreover, the pattern from contexts continues: we will split terms and types into a collection of presheaves, one for each mode so that e.g. types at mode m will be modeled by a presheaf over contexts at mode m.

Component 7.1.3. *A model of contexts $\mathcal{I}$ models terms and types when equipped with a map of presheaves $\tau_m : \mathcal{T}^{\bullet}_m \longrightarrow \mathcal{T}_m$ for each $m : \mathcal{M}$.*

We now turn to the closure of contexts under extension by variables. This procedure is complicated by MTT's modal context extension, so we begin by recalling the procedure for MLTT as described in Chapter 3.

Recall that a representable natural transformation $f : X \longrightarrow Y : \mathbf{PSh}(\mathcal{C})$ is a morphism in $\mathbf{PSh}(\mathcal{C})$ whose fibers over representable presheaves are representable:

$$\begin{tikzcd}
\mathbf{y}(d) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & Y \arrow[d] \\
\mathbf{y}(c) \arrow[r] & X
\end{tikzcd}$$

The crucial insight of *natural models* is that this property precisely captures context extensions. We can package the terms and types of our theory into a pair of presheaves $\mathcal{T}^\bullet$ and $\mathcal{T}$ over the category of contexts together with a projection $\tau : \mathcal{T}^\bullet \longrightarrow \mathcal{T}$. The requirement that τ be a representable natural transformation is equivalent to requiring all context extensions to exist:

$$\begin{tikzcd}
\mathbf{y}(\Gamma.A) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^\bullet \arrow[d] \\
\mathbf{y}(\Gamma) \arrow[r, "\lfloor A \rfloor"'] & \mathcal{T}
\end{tikzcd}$$

This capitalizes on the insight that the context $\Gamma.A$ has a universal property:

$$\hom(\Delta, \Gamma.A) \cong \textstyle\sum_{\gamma:\hom(\Delta,\Gamma)} \mathsf{Tm}(\Delta, A[\gamma])$$

The situation in MTT is complicated by the existence of *modal context extensions* $\Gamma.(\mu \mid A)$. Let us begin by observing that modal context extensions enjoy a similar universal property in the syntax of MTT:

Lemma 7.1.3. *A substitution $\Gamma \vdash \delta : \Delta.(\mu \mid A)$ is determined naturally in Γ by a substitution $\Gamma \vdash \delta_0 : \Delta$ and $\Gamma.\{\mu\} \vdash M : A[\delta_0.\{\mu\}]$.*

Proof. Given δ, we define $\delta_0 = \mathbf{p} \circ \delta$ and $M = \mathbf{v}[\delta]$. This assignment is bijective, with the inverse sending δ_0 and M to $\delta_0.M$. □

If we assume $\mu = \mathsf{id}$, Lemma 7.1.3 ensures that $\Gamma.(\mu \mid A)$ has the same universal property as $\Gamma.A$. To model cases where μ is not the identity, we must describe the semantic counterpart to a term in a context restricted by μ.

Notation 7.1.4. Given a model of contexts $\mathcal{I}$, we will frequently use $\mathcal{I}(\mu)^*$ for some $\mu : n \longrightarrow m$. When no ambiguity can arise, we will abbreviate this to μ^* to avoid unnecessary clutter. We will similarly shorten $\mathcal{I}(\mu)_!$ to $\mu_!$.

Crucially, given a modality $\mu : n \longrightarrow m$, the presheaf $\mu^*\mathcal{T}_n$ models types in context restricted by μ. To see this, fix $C : \mathcal{I}(m)$ and apply the Yoneda lemma:

$$(\mu^*\mathcal{T}_n)(C)$$

$$
\begin{aligned}
&= \hom(\mathbf{y}(C), \mu^*\mathcal{T}_n) \\
&= \hom(\mu_!\mathbf{y}(C), \mathcal{T}_n) \\
&= \hom(\mathbf{y}(\mathcal{I}(\mu)(C)), \mathcal{T}_n) \\
&= \mathcal{T}_n(\mathcal{I}(\mu)(C))
\end{aligned}
$$

This pattern holds generally: if one regards some presheaf $X : \mathbf{PSh}(\mathcal{I}(m))$ as classifying an object in the generic context, μ^*X classifies the same object in the generic context *restricted by* μ. Accordingly, we will use $\mu^*\mathcal{T}_m^\bullet$ to represent terms in a restricted context. Since μ^* respects pullbacks (in fact, all limits and colimits), we may carve out terms of some type by pullback first and then apply μ^* or begin by applying μ^* and then restricting.

Returning to modal context extension, in light of Lemma 7.1.3 and the above discussion, extending $C : \mathcal{I}(m)$ by a type $\lfloor A \rfloor : \mathbf{y}(C) \longrightarrow \mu^*\mathcal{T}_n$ should be an object an object D satisfying the following property:

$$\mathbf{y}(D) \cong \mathbf{y}(C) \times_{\mu^*\mathcal{T}_n} \mu^*\mathcal{T}_n^\bullet$$

In other words, $\mathcal{I}$ supports μ-modal context extension just when $\mu^*(\tau_n)$ is representable. By requiring this for all modalities $\mu : n \longrightarrow m$ we arrive at the next requirement.

Component 7.1.4. *A modal of types and terms $\mathcal{I}$ supports modal context extension if for each $\mu : n \longrightarrow m : \mathcal{M}$, the natural transformation $\mu^*\tau_n$ is representable.*

The legs of the pullback cone encode weakening and the variable rule.

The representability of $\mu^*\tau_n$ enables the following succinct characterization of the polynomial functor $\mathbf{P}_{\mu^*\tau_n}$ which we record for future use:

Lemma 7.1.5. *Given $C : \mathcal{I}(m)$, a C-point of $\mathbf{P}_{\mu^*\tau_n}(X)$ is a type $\lfloor A \rfloor : \mathbf{y}(C) \longrightarrow \mu^*\mathcal{T}_n$ and an element of $X(C')$, where C' is the μ-modal extension of C by A.*

Proof. The proof follows the same ideas as Awodey [Awo18, Proposition 6]. □

Closure under type operators While the addition of modal context extension suffices to encode the theory of contexts and substitutions, it remains to close each $\mathcal{T}_m$ and τ_m under the type and term formers of MTT. Recall from Section 6.2 that connectives in MTT come in two classes: modal and mode-local. The mode-local connectives have rules only working with a single mode and their encoding in models precisely follows the relevant definitions in MLTT. Accordingly, the structure for identity types, dependent sums, natural numbers, etc. are specified exactly as in Section 3.4.

Component 7.1.5. *A model of terms and types $\mathcal{I}$ interprets mode-local connectives when τ_m is equipped with codes closing it under dependent sums, booleans, a Tarski universe, and the natural numbers for each $m : \mathcal{M}$*

It remains only to describe the requirements for modal types and modal dependent products. We begin with the latter—they enjoy an η law which makes them particularly simple to describe.

Component 7.1.6. *A model of types and terms $\mathcal{T}$ supports modal dependent products when equipped with a choice of pullback square of the following shape for each $\mu : n \longrightarrow m$:*

$$\begin{tikzcd}
\mathbf{P}_{\mu^*\tau_n}(\mathcal{T}^\bullet_m) \arrow[r] \arrow[d, "\mathbf{P}_{\mu^*\tau_n}(\tau_m)"'] \arrow[dr, phantom, very near start, "\lrcorner"] & \mathcal{T}^\bullet_m \arrow[d, "\tau_m"] \\
\mathbf{P}_{\mu^*\tau_n}(\mathcal{T}_m) \arrow[r] & \mathcal{T}_m
\end{tikzcd} \tag{7.1}$$

This can be unfolded using Lemma 7.1.5 to see that they encode the expected structure.

Specifying modal types is complicated by the lack of a unicity principle. We are not able to structure the formation, introduction, and elimination rules into a single pullback square. We therefore first isolate the formation and introduction rules.

Component 7.1.7. *A model of types and terms $\mathcal{T}$ supports formation and introduction rules for modal types when equipped with a choice of the following square for each $\mu : n \longrightarrow m$:*

$$\begin{tikzcd}
\mu^*\mathcal{T}^\bullet_n \arrow[r] \arrow[d, "\mu^*\tau_n"'] & \mathcal{T}^\bullet_m \arrow[d, "\tau_m"] \\
\mu^*\mathcal{T}_n \arrow[r] & \mathcal{T}_m
\end{tikzcd} \tag{7.2}$$

Let us begin by rephrasing the elimination rule in terms of the category of contexts. Like any *pattern-matching* elimination rule, modal elimination can be captured by a family of pullback-stable lifts. Specifically, the rule ensures that the canonical substitution $\Gamma.(\nu \circ \mu \mid A) \longrightarrow \Gamma.(\nu \mid \langle \mu \mid A \rangle)$ is *stably left orthogonal* to maps $\Delta.(\mathsf{id} \mid B) \longrightarrow \Delta$. That is, there exists a pullback-stable choice of lift for every diagram of the following shape:

$$\begin{tikzcd}
\Gamma.(\nu \circ \mu \mid A) \arrow[r] \arrow[d] & \Delta.(\mathsf{id} \mid B) \arrow[d] \\
\Gamma.(\nu \mid \langle \mu \mid A \rangle) \arrow[r] \arrow[ur, dotted] & \Delta
\end{tikzcd} \tag{7.3}$$

Pullback-stability makes this definition particularly complicated to succinctly capture. Following the treatment of intensional identity types or booleans, we capture this through a left lifting structure (Definition 3.4.16).

It remains to find two specific maps to use as input for the lifting structure. These should encode the generic versions of the two classes of maps we require lifts for. The right-hand map has already been described: we will use τ_m as a generic map corresponding to $\Delta.(\mathsf{id} \mid A) \longrightarrow \Delta$. Indeed, by construction such maps precisely correspond to a pullback of τ_m to a representable fiber. It remains to construct the left-hand map meant to represent the generic map $\Gamma.(\nu \circ \mu \mid A) \longrightarrow \Gamma.(\nu \mid \langle \mu \mid A \rangle)$.

To this end, consider the following pullback:

$$\begin{tikzcd}
\mu^*\mathcal{T}^\bullet_n \arrow[r, dashed, "m"] \arrow[dr] \arrow[rr, bend left] & \mu^*\mathcal{T}_n \times_{\mathcal{T}_m} \mathcal{T}^\bullet_m \arrow[r] \arrow[d] & \mathcal{T}^\bullet_m \arrow[d] \\
& \mu^*\mathcal{T}_n \arrow[r] & \mathcal{T}_m
\end{tikzcd} \tag{7.4}$$

Fix an element $C : \mathcal{J}(m)$ and a point $\lfloor A \rfloor : \mathbf{y}(C) \longrightarrow \mu^*\mathcal{T}_n$. Pulling back m along $\lfloor A \rfloor$ yields a map over $\mathbf{y}(C)$ and, calculating, this map interprets the canonical substitution sending the generic annotated variable to the generic variable of modal type. With this observation to hand, we can crystallize *non-crisp* modal elimination for μ as a lifting structure of the following type:

$$m \pitchfork \tau_m : \mathbf{PSh}(\mathcal{J}(m))/\mu^*\mathcal{T}_n \tag{7.5}$$

Notice that this lifting structure must take place in the slice over $\mu^*\mathcal{T}_n$. This should be compared to Awodey [Awo18] where the lifting structure for the identity type also takes place in a slice to account for the type parameter of the identity type.

To account for modal elimination with a framing modality $\nu : m \longrightarrow o$, we must modify the left-hand morphism and ask for the following lifting structure:

$$\nu^* m \pitchfork \tau_o : \mathbf{PSh}(\mathcal{J}(o))/(\nu \circ \mu)^*\mathcal{T}_n$$

Component 7.1.8. *A model $\mathcal{J}$ supporting modal formation and introduction rules supports modal elimination when it is equipped with lifting structures of the following type for each $\mu : n \longrightarrow m$ and $\nu : m \longrightarrow o$:*

$$\nu^* m \pitchfork \tau_o : \mathbf{PSh}(\mathcal{J}(o))/(\nu \circ \mu)^*\mathcal{T}_n$$

Note that when $\nu = \mathsf{id}$, $\nu^* = \mathsf{id}$ and therefore this also captures Structure 7.5.

Finally, we must explicitly require that universes in each mode are closed under the non-mode local connectives.

Component 7.1.9. *Fix a model $\mathcal{J}$ with Components 7.1.5 to 7.1.7 and write $\upsilon_m : \mathcal{U}^\bullet_m \longrightarrow \mathcal{U}_m$ for the interpretation of the Tarski universe in mode m. $\mathcal{J}$ supports codes for modal types and modal dependent products when equipped a choice of morphisms making the following squares commute for each $\mu : n \longrightarrow m$:*

$$\begin{tikzcd}
\mathbf{P}_{\mu^*\upsilon_n}(\mathcal{U}_m) \arrow[r] \arrow[d, dotted] & \mathbf{P}_{\mu^*\tau_n}(\mathcal{T}_m) \arrow[d] & \mu^*\mathcal{U}_n \arrow[r] \arrow[d, dotted] & \mu^*\mathcal{T}_n \arrow[d] \\
\mathcal{U}_m \arrow[r] & \mathcal{T}_m & \mathcal{U}_m \arrow[r] & \mathcal{T}_m
\end{tikzcd}$$

Definition 7.1.6. A model of MTT consists of Components 7.1.1 to 7.1.9. We will write $\mathcal{J}$ to represent the entirety of the model.

While less useful, we are also able to rephrase morphisms of models in this language.

Definition 7.1.7. A morphism of models $\alpha : \mathcal{I} \longrightarrow \mathcal{J}$ of MTT consists of a strict 2-natural transformation between the 2-functors of contexts along with a choice of pullback square for each $m : \mathcal{M}$:

$$\begin{tikzcd}
\mathcal{T}^{\bullet}_{\mathcal{I},m} \arrow[r, "\mathcal{T}^{\bullet}_{\alpha}"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}^{\bullet}_{\mathcal{J},m} \arrow[d] \\
\mathcal{T}_{\mathcal{I},m} \arrow[r, "\mathcal{T}_{\alpha}"'] & \mathcal{T}_{\mathcal{J},m}
\end{tikzcd}$$

Moreover, we require that α, $\mathcal{T}_\alpha$, and $\mathcal{T}^{\bullet}_{\alpha}$ commute strictly with choice of structure in Components 7.1.2 and 7.1.4 to 7.1.9.

As mentioned earlier, this definition is a reformulation of the model yielded by the machinery of generalized algebraic theories. This fact yields the following theorem:

Theorem 7.1.8. *The (fully annotated) syntax of MTT organizes into the initial model.*

We conclude this section by obtaining reaping some low-hanging fruit from this reformulation.

Theorem 7.1.9. *A pseudofunctor $F : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ equipped further with Components 7.1.2 to 7.1.9 can be promoted to a model of MTT whose underlying 2-functor is equivalent to F.*

Proof. First, recall that a pseudofunctor F can always be strictified to a 2-naturally equivalent 2-functor $\mathcal{I}$. Denote the 2-natural equivalence α. We shall show that $\mathcal{I}$ can be endowed with the remaining structure of an MTT model. The categories $F(m)$ and $\mathcal{I}(m)$ are equivalent, so the existence of an empty context in $F(m)$ immediately transfers to $\mathcal{I}(m)$. Moreover, the pseudofunctors $\mathcal{M} \longrightarrow \mathbf{Cat}$ given by $\mathbf{PSh}(F(-))$ and $\mathbf{PSh}(\mathcal{I}(-))$ are equivalent and so the remaining structure transfers from $\mathbf{PSh}(F(-))$ to $\mathbf{PSh}(\mathcal{I}(-))$. □

Remark 7.1.10. While we will not crystallize the connection here, the previous result can be seen as a conservativity theorem. One may consider a version of MTT where the definitional equalities $\Gamma.\{\nu\}.\{\mu\} = \Gamma.\{\nu \circ \mu\}$ and $\Gamma.\{\mathsf{id}\} = \Gamma$ are weakened to coherent isomorphisms. The previous theorem then shows that the embedding of this weakened theory into MTT is conservative. ◇

7.2 (Weak) dependent right adjoints and models of MTT

While the Definition 7.1.6 precisely matches syntax, most models which arise in practice are stricter than this definition requires. In particular, typical semantic models are built from *dependent right adjoints (DRAs)* (Definition 5.5.3). In this section, we discuss the precise relationship between the semantic requirements of MTT and DRAs and prove several recognition principles for models of MTT.

7.2.1 Models of MTT from dependent right adjoints

We begin by recalling the definition of a dependent adjunction between two natural models from Section 5.5:

Definition 7.2.1 (Dependent right adjoint)**.** Given a pair of natural models of type theory $\left(\mathcal{C}, \tau_{\mathcal{C}} : \mathcal{T}^{\bullet}_{\mathcal{C}} \longrightarrow \mathcal{T}_{\mathcal{C}}\right)$ and $\left(\mathcal{D}, \tau_{\mathcal{D}} : \mathcal{T}^{\bullet}_{\mathcal{D}} \longrightarrow \mathcal{T}_{\mathcal{D}}\right)$, a *dependent right adjoint (DRA)* from $\mathcal{C}$ to $\mathcal{D}$ consists of the following data:

1. A functor $L : \mathcal{D} \longrightarrow \mathcal{C}$
2. A choice of pullback square of the following shape in $\mathbf{PSh}(\mathcal{D})$:

$$\begin{tikzcd}
L^*\mathcal{T}^{\bullet}_{\mathcal{C}} \arrow[r] \arrow[d] \arrow[dr, phantom, very near start, "\lrcorner"] & \mathcal{T}^{\bullet}_{\mathcal{D}} \arrow[d] \\
L^*\mathcal{T}_{\mathcal{C}} \arrow[r] & \mathcal{T}_{\mathcal{D}}
\end{tikzcd} \tag{7.6}$$

Notation 7.2.2. In the context of Definition 7.2.1, when L is fixed we refer to the existence of Diagram 7.6 as a DRA over L.

Remark 7.2.3. In Definition 7.2.1 we have not required that L be a left adjoint nor that the type action simulating the right adjoint extend to contexts. Frequently, however, both of these will be true: L will have a right adjoint R and the type and term components of a DRA will come from R. By standard adjoint yoga, we may rephrase Diagram 7.6 to the following diagram:

$$\begin{tikzcd}
R_!\mathcal{T}^{\bullet}_{\mathcal{C}} \arrow[r] \arrow[d] \arrow[dr, phantom, very near start, "\lrcorner"] & \mathcal{T}^{\bullet}_{\mathcal{D}} \arrow[d] \\
R_!\mathcal{T}_{\mathcal{C}} \arrow[r] & \mathcal{T}_{\mathcal{D}}
\end{tikzcd}$$

This observation is a rephrasing of the lemma due to Birkedal et al. [Bir+20] that right adjoints extending to weak CwF morphisms organize into dependent right adjoints. Having done the work of restating DRAs in terms of natural models, however, it becomes a near tautology. In particular, as any right adjoint is flat [Bor94] and so the above square is cartesian if R is a weak CwF morphism. ◇

We wish to relate this definition to the requirements for modeling modalities in MTT (Components 7.1.7 and 7.1.8) as well as the other modal features (Components 7.1.4 and 7.1.6). To facilitate such a comparison, we introduce the following notion:

Definition 7.2.4 (Weak dependent right adjoint)**.** Given a pair of natural models of type theory $\left(\mathcal{C}, \tau_{\mathcal{C}} : \mathcal{T}^{\bullet}_{\mathcal{C}} \longrightarrow \mathcal{T}_{\mathcal{C}}\right)$ and $\left(\mathcal{D}, \tau_{\mathcal{D}} : \mathcal{T}^{\bullet}_{\mathcal{D}} \longrightarrow \mathcal{T}_{\mathcal{D}}\right)$, a *weak dependent right adjoint (wDRA)* from $\mathcal{C}$ to $\mathcal{D}$ consists of the following data:

1. A functor $L : \mathcal{D} \longrightarrow \mathcal{C}$

2. A choice of commuting square of the following shape in $\mathbf{PSh}(\mathcal{D})$:

$$\begin{tikzcd} L^*\mathcal{T}^\bullet_{\mathcal{C}} \arrow[r] \arrow[d] & \mathcal{T}^\bullet_{\mathcal{D}} \arrow[d] \\ L^*\mathcal{T}_{\mathcal{C}} \arrow[r] & \mathcal{T}_{\mathcal{D}} \end{tikzcd} \tag{7.7}$$

3. A choice of lifting structure of the following shape:

$$\left(L^*\mathcal{T}^\bullet_{\mathcal{C}} \longrightarrow L^*\mathcal{T}_{\mathcal{C}} \times_{\mathcal{T}_{\mathcal{D}}} \mathcal{T}^\bullet_{\mathcal{D}}\right) \pitchfork \tau_{\mathcal{D}} : \mathbf{PSh}(\mathcal{D})/L^*\mathcal{T}_{\mathcal{C}}$$

The essential difference between a weak DRA and a DRA lies in the third point of Definition 7.2.4: a weak DRA requires only that the canonical gap map $L^*\mathcal{T}^\bullet_{\mathcal{C}} \longrightarrow L^*\mathcal{T}_{\mathcal{C}} \times_{\mathcal{T}_{\mathcal{D}}} \mathcal{T}^\bullet_{\mathcal{D}}$ be stably orthogonal to display maps, while a proper DRA requires it to be an isomorphism. This should be compared with the distinction between extensional identity types and intensional identity types or strong and weak coproducts in type theory. Just as in these other situations, we weaken the universal property of a connective by requiring only that a certain map be invertible only "from the perspective of a type" rather than a true isomorphism.

Lemma 7.2.5. *Any DRA induces a weak DRA.*

Proof. Fix a pair of natural models $\tau_{\mathcal{C}} : \mathbf{PSh}(\mathcal{C})$ and $\tau_{\mathcal{D}} : \mathbf{PSh}(\mathcal{D})$ along with a DRA between them: a functor $L : \mathcal{D} \longrightarrow \mathcal{C}$ and cartesian square $\alpha : L^*\tau_{\mathcal{C}} \longrightarrow \tau_{\mathcal{D}}$. We wish to show that this data induces a weak DRA. The first two requirements of Definition 7.2.4 are satisfied immediately by L and α respectively. It remains only to show that the canonical gap map $m : L^*\mathcal{T}^\bullet_{\mathcal{C}} \longrightarrow L^*\mathcal{T}_{\mathcal{C}} \times_{\mathcal{T}_{\mathcal{D}}} \mathcal{T}^\bullet_{\mathcal{D}}$ is stably orthogonal $\tau_{\mathcal{D}}$ in $\mathbf{PSh}(\mathcal{D})/L^*\mathcal{T}_{\mathcal{C}}$.

To this end, let us notice first that m is an isomorphism in this category: by assumption $L^*\mathcal{T}^\bullet_{\mathcal{C}}$ is the apex of pullback $\mathcal{T}_{\mathcal{C}} \times_{\mathcal{T}_{\mathcal{D}}} \mathcal{T}^\bullet_{\mathcal{D}}$. Next, we note that an isomorphism $\iota : A \longrightarrow B$ is stably left orthogonal to any $g : X \longrightarrow Y$. Indeed, the appropriate section can be constructed as follows: $s : \iota \pitchfork g = \lambda(h_0, h_1).\ h_0 \circ \iota^{-1}$ □

A model of MTT requires slightly more than a weak DRA for each $\mu : n \longrightarrow m$. In particular, Component 7.1.8 requires more than the third point of Definition 7.2.4 to accommodate crisp induction principles for modalities. We can rephrase this additional requirement into a "pseudofunctoriality" requirement in light of Lemma 6.3.5.

Lemma 7.2.6. . *Fix natural models $\left(\mathcal{C}_i, \tau_i : \mathcal{T}^\bullet_i \longrightarrow \mathcal{T}_i\right)_{i \in \{0,1,2\}}$ and a pair of wDRAs:*

- *A weak DRA given by a functor $L_0 : \mathcal{C}_1 \longrightarrow \mathcal{C}_0$ along with the following:*

$$\begin{tikzcd} L_0^*\mathcal{T}_0^\bullet \arrow[r, "\alpha_0^\bullet"] \arrow[d] & \mathcal{T}_1^\bullet \arrow[d] \\ L_0^*\mathcal{T}_0 \arrow[r, "\alpha_0"'] & \mathcal{T}_1 \end{tikzcd} \qquad s_0 : \langle L_0^*[\tau_0], \alpha_0^\bullet \rangle \pitchfork \tau_1 : \mathbf{PSh}(\mathcal{C}_1)/L_0^*\mathcal{T}_0$$

- *A weak DRA given by a functor $L_1 : \mathcal{C}_1 \longrightarrow \mathcal{C}_0$ along with the following:*

$$\begin{tikzcd}
L_1^*\mathcal{T}_1^\bullet \arrow[r, "\alpha_1^\bullet"] \arrow[d] & \mathcal{T}_2^\bullet \arrow[d] \\
L_1^*\mathcal{T}_1 \arrow[r, "\alpha_1"'] & \mathcal{T}_2
\end{tikzcd} \qquad s_1 : \langle L_1^*[\tau_1], \alpha_1^\bullet \rangle \pitchfork \tau_2 : \mathbf{PSh}(\mathcal{C}_2)/L_1^*\mathcal{T}_1$$

The following are equivalent:

1. *There exists a lifting structure $L_1^* m_0 \pitchfork \tau_2 : \mathbf{PSh}(\mathcal{C}_2)/(L_0 \circ L_1)^*\mathcal{T}_0$ where $m_0 : L_0^*\mathcal{T}_0^\bullet \longrightarrow L_0^*\mathcal{T}_0 \times_{\mathcal{T}_1} \mathcal{T}_1^\bullet$*
2. *There exists a weak DRA from τ_0 to τ_2 over $L_0 \circ L_1$ given by the commuting square $(\beta, \beta^\bullet) : (L_0 \circ L_1)\tau_0{}^* \longrightarrow \tau_2$ and $s_2 : \langle (L_0 \circ L_1)^*[\tau_0], \beta^\bullet \rangle \pitchfork \tau_2$ along with a map making the following diagram commute:*

$$\begin{tikzcd}
 & (L_0 \circ L_1)^*\tau_0 \arrow[dl] \arrow[dr] & \\
\tau_2[\alpha_1 \circ L_1^*\alpha_0] \arrow[rr, dotted] \arrow[dr] & & \tau_2[\beta] \arrow[dl] \\
 & (L_0 \circ L_1)^*\mathcal{T}_0 &
\end{tikzcd} \tag{7.8}$$

In particular, any model of MTT is equipped with a choice of weak DRAs for each morphism in $\mathcal{M}$ along with a collection of *compositor* morphisms fitting into diagrams like Diagram 7.8 witnessing the "pseudofunctoriality" of this weak DRAs.

Corollary 7.2.7. *In the situation of Lemma 7.2.6, if the weak DRAs over L_0 and L_1 are DRAs, then both conditions hold automatically.*

Proof. It suffices to show that the second condition holds. That is, that there is a (weak) DRA over $L_0 \circ L_1$ and a map fitting into Diagram 7.8.

First, let us note that DRAs compose: given cartesian squares $\alpha_0 : L_0^*\tau_0 \longrightarrow \tau_1$ and $\alpha_1 : L_1^*[\tau_1] \longrightarrow \tau_2$, the composite $\alpha_1 \circ L_1^*\alpha_0$ is a cartesian square $(L_0 \circ L_1)^*\tau_0 \longrightarrow \tau_2$ and thus gives rise to a DRA τ_0 to τ_2. With this choice, we may take the identity to fill Diagram 7.8. □

This result demonstrates that a choice of DRAs is sufficient to model the modalities in MTT. In fact, it also suffices to realize all the modal structure of MTT. In particular, modalized context extension and modal dependent products.

Lemma 7.2.8. *Fix a pair of natural models $\left(\mathcal{C}_i, \tau_i : \mathcal{T}_i^\bullet \longrightarrow \mathcal{T}_i\right)_{i \in \{0,1\}}$ along with a DRA $L : \mathcal{C}_1 \longrightarrow \mathcal{C}_0$ and $\alpha : L^*\tau_0 \longrightarrow \tau_1$. The map $L^*\tau_0$ is a representable natural transformation*

Proof. This follows immediately from the fact that representable natural transformations are stable under pullback. □

Lemma 7.2.9. *Fix a pair of natural models* $(\mathcal{C}_i, \tau_i : \mathcal{T}_i^\bullet \longrightarrow \mathcal{T}_i)_{i \in \{0,1\}}$ *along with a DRA* $L : \mathcal{C}_1 \longrightarrow \mathcal{C}_0$ *and* $\alpha : L^*\tau_0 \longrightarrow \tau_1$*. If* τ_1 *is closed under dependent products i.e., there is a cartesian square* $\mathbf{P}_{\tau_1}(\tau_1) \longrightarrow \tau_1$ *then it is closed under modal dependent products in the sense of Component 7.1.6.*

Proof. This follows immediately from the observation that $\mathbf{P}_{-}$ sends pullback squares to pullback squares, so $\mathbf{P}_{L^*\tau_0}(\tau_1)$ is a pullback of $\mathbf{P}_{\tau_1}(\tau_1)$. □

We may summarize the entirety of this discussion through the following theorem:

Theorem 7.2.10. *The following data suffices to construct a model of MTT:*

- *A pseudofunctor* $\mathcal{I} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$,
- *for each* $m : \mathcal{M}$*, a choice of terminal object in* $\mathcal{I}(m)$*,*
- *for each* $m : \mathcal{M}$*, a representable natural transformation* $\tau_m : \mathbf{PSh}(\mathcal{I}(m))$ *closed under dependent sums, dependent products, identity types, a Tarski universe, and natural numbers.*
- *for each* $\mu : n \longrightarrow m$*, a DRA from* τ_n *to* τ_m *over* $\mathcal{I}(\mu)$ *such that the Tarski universe in each mode is closed under the DRA.*

Proof. Using Theorem 7.1.9, it suffices to construct Components 7.1.2 to 7.1.9.

The assumptions of this theorem automatically give us Components 7.1.2, 7.1.3 and 7.1.5. Lemma 7.2.5 and Corollary 7.2.7 together with our assumed supply of dependent right adjoints yields Components 7.1.7 and 7.1.8. Lemma 7.2.8 and Lemma 7.2.9 yield Component 7.1.4 and Component 7.1.6, respectively. Finally, our assumption that each universe is closed under these DRAs gives Component 7.1.9. □

7.2.2 Constructing dependent right adjoints

Theorem 7.2.10 gives us a powerful tool for constructing models of MTT. In this subsection, we discuss some of the more common situations which satisfy these assumptions. In particular, when dealing with suitably well-behaved (presentable) categories we are often able to derive all of these assumptions from the pseudofunctor of adjoints. In less well-behaved cases where the machinery of the presentable categories is not available, we discuss an extension of the strictification theorem of Section 5.5 to handle Tarski universes.

When working with Grothendieck topoi, the local universes strictification introduced in Section 3.5.2 is an unnecessary complication. It is instead easier to build a model of MLTT (or MTT) through a generic family of small sheaves following Section 3.5.1. To this end, we recall a corollary of Corollary 3.3.13 exposed by Gratzer et al. [GSS22]:

Corollary 7.2.11. *In a Grothendieck topos* $\mathcal{E}$*, there exists a cardinal* λ *such that for any inaccessible cardinals* $\kappa_1 > \kappa_0 \rhd \lambda$ *such that there exists exists a model of MLTT in* $\mathcal{E}$ *all connectives and a strict Tarski universe where types are interpreted as relatively* κ_1*-compact families and small types are relative* κ_0*-small families.*

In particular, in the model described by Corollary 7.2.11, the family of types and terms $\tau : \mathbf{PSh}(\mathcal{E})$ is given by $\mathbf{y}(\upsilon_{\kappa_1})$. We wish to describe under what assumptions a collection of these models can be extended to a model of MTT.

Theorem 7.2.12. *Fix a pseudofunctor $\mathcal{I} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ where $\mathcal{I}(m)$ is a Grothendieck topos for all $m : \mathcal{M}$ and $\mathcal{I}(\mu)$ has a right adjoint $\bar{\mathcal{I}}(\mu)$ for all $\mu : n \longrightarrow m$. There exists a cardinal λ such that for arbitrary inaccessible cardinals $\kappa_1 > \kappa_0 \rhd \lambda$ this pseudofunctor can always be extended to a model of MTT where mode-local connectives are interpreted by Corollary 7.2.11.*

Proof. Our construction will factor through Theorem 7.2.10. Accordingly, we must construct models of MLTT in each $\mathcal{I}(m)$ along with DRAs between these models.

We begin by recalling that as right adjoints between presentable categories each $\bar{\mathcal{I}}(\mu)$ is accessible. In particular, using Adámek and Rosický [AR94, Theorem 2.19] there exists a cardinal λ such that for all $\kappa \rhd \lambda$ each $\bar{\mathcal{I}}(\mu)$ is κ-accessible and preserves κ-compact objects. As we have assumed that each $\mathcal{I}(m)$ is a topos, by Theorem 3.3.12 there exists some λ_m such for any cardinal $\kappa \rhd \lambda$ there exists a generic family for relatively κ-compact maps satisfying realignment. We may further increase λ_m to ensure that if $\kappa \rhd \lambda_m$ then κ-compact objects in $\mathcal{I}(m)$ are closed under finite limits.

Using Adámek and Rosický [AR94, Example 2.13 (6)], we may now choose a cardinal $\bar{\lambda}$ such that $\bar{\lambda} \rhd \lambda$ and $\bar{\lambda} \rhd \lambda_m$ for all $m : \mathcal{M}$. Fix two inaccessible cardinals $\kappa_1 > \kappa_0 \rhd \bar{\lambda}$. By construction and Theorem 3.3.12, we obtain the following:

- Each $\bar{\mathcal{I}}(\mu)$ is both κ_0- and κ_1-accessible.
- Each $\mathcal{I}(m)$ has a generic maps $\upsilon_{m,0}$ and $\upsilon_{m,1}$ for relatively κ_0-compact and relatively κ_1-compact families respectively.
- In each $\mathcal{I}(m)$, both κ_0-compact and κ_1-compact objects are stable under finite limits.

Now, Corollary 7.2.11 ensures that each $\mathcal{I}(m)$ interprets MLTT with a Tarski universe using $\mathbf{y}(\upsilon_{m,1})$ as the universe of types. It remains to construct the necessary DRAs between these models.

In this situation the pullback diagram from Definition 7.2.1 can be simplified. First, by standard adjoint yoga, $\mathcal{I}(\mu)^* = \bar{\mathcal{I}}(\mu)_!$. Secondly, as the Yoneda embedding is full and faithful and preserves limits, it is both necessary and sufficient to construct pullback diagrams of the following shapes for each $\mu : n \longrightarrow m$:

$$\begin{tikzcd}
\bar{\mathcal{I}}(\mu)(\mathcal{U}^{\bullet}_{n,0}) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{U}^{\bullet}_{m,0} \arrow[d] \\
\bar{\mathcal{I}}(\mu)(\mathcal{U}_{n,0}) \arrow[r, "\alpha_{\mu,0}"'] & \mathcal{U}_{m,0}
\end{tikzcd}
\qquad
\begin{tikzcd}
\bar{\mathcal{I}}(\mu)(\mathcal{U}^{\bullet}_{n,1}) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{U}^{\bullet}_{m,1} \arrow[d] \\
\bar{\mathcal{I}}(\mu)(\mathcal{U}_{n,1}) \arrow[r, "\alpha_{\mu,1}"'] & \mathcal{U}_{m,1}
\end{tikzcd}$$

Furthermore, to ensure cumulativity we must ensure that the following diagram commutes:

$$\begin{tikzcd}
\bar{\mathcal{I}}(\mu)(\mathcal{U}_{n,0}) \arrow[r] \arrow[d] & \mathcal{U}_{m,1} \arrow[d] \\
\bar{\mathcal{I}}(\mu)(\mathcal{U}_{n,1}) \arrow[r] & \mathcal{U}_{m,1}
\end{tikzcd}$$

We will construct the necessary pullback squares by showing that $\bar{\mathcal{I}}(\mu)$ preserves relatively κ_0- and κ_1-compact families and then using the genericity of $\upsilon_{m,i}$. Accordingly, fix a relatively κ_0-compact family $f : X \longrightarrow Y : \mathcal{I}(n)$—the argument for κ_1 is identical. We wish to show that $\bar{\mathcal{I}}(\mu)(f)$ is relatively κ_0-small, so we begin by fixing a κ_0-compact object $Z : \mathcal{I}(m)$ and a morphism $Z \longrightarrow \bar{\mathcal{I}}(\mu)(X)$. Consider the following pullback diagrams:

$$\begin{tikzcd}
Z \times_{\bar{\mathcal{I}}(\mu)(X)} \bar{\mathcal{I}}(\mu)(Y) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \bar{\mathcal{I}}(\mu)(\mathcal{I}(\mu)(Z) \times_Y X) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \bar{\mathcal{I}}(\mu)(X) \arrow[d] \\
Z \arrow[r] & \bar{\mathcal{I}}(\mu)(\mathcal{I}(\mu)(Z)) \arrow[r] & \bar{\mathcal{I}}(\mu)(Y)
\end{tikzcd}$$

Notice that the right-hand square is indeed a pullback as $\bar{\mathcal{I}}(\mu)$ preserves limits. As Z is κ_0-compact and both $\bar{\mathcal{I}}(\mu)$ preserves κ_0-direct colimits, $\mathcal{I}(\mu)(Z)$ is κ_0-compact. Accordingly, $\mathcal{I}(\mu)(Z) \times_Y X$ is κ_0 compact because $X \longrightarrow Y$ is relatively κ_0-compact by assumption. Finally, as $\bar{\mathcal{I}}(\mu)$ preserves κ_0-compact objects, $Z \times_{\bar{\mathcal{I}}(\mu)(X)} \bar{\mathcal{I}}(\mu)(Y)$ is the finite limit of κ_0-compact objects and therefore κ_0-compact.

For any $\mu : n \longrightarrow m$, we therefore conclude that $\bar{\mathcal{I}}(\mu)(\upsilon_{n,i})$ is relatively κ_i-compact. As $\upsilon_{m,0}$ is generic among such maps, we may therefore choose a pullback square of the following shape:

$$\begin{tikzcd}
\bar{\mathcal{I}}(\mu)(\mathcal{U}^{\bullet}_{n,0}) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{U}^{\bullet}_{m,0} \arrow[d] \\
\bar{\mathcal{I}}(\mu)(\mathcal{U}_{n,0}) \arrow[r] & \mathcal{U}_{m,0}
\end{tikzcd}$$

Rather than doing the same for $\upsilon_{m,1}$, however, we will use realignment to coherently extend this choice. In particular, using realignment we choose a pullback square $\bar{\mathcal{I}}(\mu)(\upsilon_{n,1}) \longrightarrow \upsilon_{m,1}$ extending the pullback square $\bar{\mathcal{I}}(\mu)(\upsilon_{n,0}) \longrightarrow \upsilon_{m,1}$. These choices then satisfy the third component of Theorem 7.2.10 and complete the proof. □

Remark 7.2.13. This theorem substantially generalizes the recognition principles for dependent right adjoints given by Gratzer et al. [Gra+21]. In particular, using the machinery of accessible functors allows us to completely avoid any need to restrict the forms of the right adjoints given as input to Theorem 7.2.12. ◇

Remark 7.2.14. Much of the proof of Theorem 7.2.12 can be generalized to apply to arbitrary *presentable* categories. However, the existence of suitably well-behaved universes is far less clear cut than Theorem 3.3.12 and these must constructed by more specialized means. ◇

Corollary 7.2.15 (Soundness of MTT)**.** *Regardless of the mode theory, there is no closed proof of* $\mathsf{Id}(0,1)$ *in MTT.*

Proof. This follows from Theorem 7.2.12 instantiated with the 2-functor $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ sending each mode to $\mathbf{Set}$ and each 1- and 2-cell to the identity. In particular, because $0 \neq 1$ in the $\mathbf{Set}$ model of MLTT, MTT is consistent. □

7.2.3 An extension of the local universes construction

In some circumstances, a hierarchy of suitably strict universes (Theorem 3.3.12) might be unavailable. Most typically, this comes from working with an elementary topos not based over $\mathbf{Set}$. In these circumstances, it is more difficult to construct a model of MTT and arbitrary right adjoints are insufficient. In this subsection, we extend the local universes construction and its extension by Shulman [Shu19] to Tarski universes to apply to MTT.

Remark 7.2.16. Note that Shulman [Shu19] only extends the local universes construction to non-cumulative Tarski universes. This in particular means that equations that ensure that applying El to a dependent product yields a dependent product on the nose. However, in the presence of multiple Tarski universes lifts between them do not necessarily preserve chosen codes. For our purposes, the distinction is immaterial as we consider only one universe. This is, however, one of the places for which the presence of hierarchy of universes offers a challenge over a single universe. ◇

Remark 7.2.17. More recently, Shulman [Shu23] has given a version of the local universes construction for MTT with (weak) dependent right adjoints. This work does not discuss universes, so we explain the extension here. We also take this opportunity to specialize the proof from op. cit. to our case and thereby simplify some of the notations and constructions; while Shulman [Shu23] deals with strictifying a weak model in a way compatible with the *codextrification* procedure of op. cit., we focus only on the first point. ◇

Definition 7.2.18. In a category of display maps $(\mathcal{C}, \mathscr{D})$, recall that a map $f : A \longrightarrow B$ is said to be anodyne when it is weakly left orthogonal to display maps. A map is stably anodyne if any pullback of it is anodyne.

Definition 7.2.19. Fix a pair of categories with display maps and a functor $R : (\mathcal{C}, \mathscr{D}_{\mathcal{C}}) \longrightarrow (\mathcal{D}, \mathscr{D}_{\mathcal{D}})$. A pre weak dependent right adjoint over R consists of a factorization of choice of factorization $R\big(f : E \longrightarrow B\big) = m(f) \circ i(f)$ into a stably anodyne map over $R(B)$ followed by a display map.

Definition 7.2.20. Given a pre weak DRA $R : (\mathcal{C}, \mathscr{D}_{\mathcal{C}}) \longrightarrow (\mathcal{D}, \mathscr{D}_{\mathcal{D}}), m, i$ along with another functor $G : (\mathcal{D}, \mathscr{D}_{\mathcal{D}}) \longrightarrow (\mathcal{E}, \mathscr{D}_{\mathcal{E}})$, this R satisfies G-crisp induction principles if $G(i(f))$ is stably anodyne in $\mathcal{E}$.

Theorem 7.2.21. *Fix a pseudofunctor* $\mathcal{I} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ *equipped with the following:*

- *Each* $\mathcal{I}(m)$ *is a locally Cartesian closed category with display maps* $\mathscr{D}_m$ *and display maps are closed under* id*, composition, pushfoward and arbitrary maps admit a factorization as an anodyne map followed by a display map.*

- *The functor $\mathcal{I}(\mu)$ has a right adjoint R_μ and a pre weak DRA i_μ, m_μ over R_μ.*
- *In each $\mathcal{I}(m)$ there exists a display map $\upsilon_m : \mathcal{U}^\bullet_m \longrightarrow \mathcal{U}_m$ closed under dependent sums, identity types etc. such that $\mathcal{U}_m \longrightarrow \mathbf{1}$ is also a display map.*
- *If $\mu : n \longrightarrow m$ and $p \in \mathscr{D}_n$ then $\mathscr{D}_m$ is closed under pushforward along $R_\mu(p)$. We further require a map $\mathsf{ProdCode}_\mu : \mathbf{P}_{R_\mu(\upsilon_{[n]})} \longrightarrow \mathcal{U}_m$ classifying modal dependent products.*
- *For each $\mu : n \longrightarrow m$, there exists a map $\mathsf{ModCode}_\mu : R_\mu(\mathcal{U}_n) \longrightarrow \mathcal{U}_m$ such that for any display map $p : E \longrightarrow B \in \mathscr{D}_n$ classified by $f : B \longrightarrow \mathcal{U}_n$, the family $m_\mu(p)$ is classified by $\mathsf{ModCode}_\mu \circ R_\mu(f)$.*

The local universes models defined by Shulman [Shu19] in each $\mathcal{I}(m)$ assemble into a model of MTT without any crisp induction principles. Moreover, if the weak DRA over μ satisfies R_ν-crisp induction, the μ-modal types have ν-crisp induction. Finally, if any i_μ is an isomorphism, then $\langle \mu \mid - \rangle$ is realized by a proper DRA.

Notation 7.2.22. To ease notation, given $f : E \longrightarrow B \in \mathscr{D}_n$, we write $\mu(E)$ for the domain of display map $m_\mu(f)$.

Proof. We begin by recalling the variant of the local universes construction given by Shulman [Shu19] to account for Tarski universes. Let us denote the standard local universes construction $\coprod_{p \in \mathscr{D}_m} \mathbf{y}(p)$ over $\mathcal{I}(m)$ by $\lambda_m : \mathcal{L}^\bullet_m \longrightarrow \mathcal{L}_m$. The refined universe of types is then defined as follows:

$$\tau_m = \mathbf{y}(\upsilon_m) + \lambda_m \tag{7.9}$$

It follows that e.g., $\mathcal{T}_m$ is precisely $\mathbf{y}(\mathcal{U}_m) + \mathcal{L}_m$. The results of Shulman [Shu19] along with the assumptions on υ_m and $\mathscr{D}_m$ ensure that τ_m is a model of extensional type theory with a Tarski universe.

Modal context extension We now show that $\mathcal{T}_m$ enjoys modal context extension. In particular, we must show that for any $\nu : m \longrightarrow o$ that $\mathcal{I}(\mu)^*\tau_m$ is a representable natural transformation. First, we notice that the following isomorphism:

$$\mathcal{I}(\mu)^*\tau_m \cong \mathbf{y}(R_\mu(\upsilon_m)) + \textstyle\coprod_{p \in \mathscr{D}_m} \mathbf{y}(R_\mu(p))$$

As the coproduct of representable families, $\mathcal{I}(\mu)^*\tau_m$ is therefore representable.

Formation and introduction rules for modal types We now construct a square representing the formation $\alpha_\mu : \mathcal{I}(\mu)^*\tau_n \longrightarrow \tau_m$ for all $\mu : n \longrightarrow m$. We will only show the definition of $\mathsf{cod}(\alpha_\mu)$ as the construction of $\mathsf{dom}(\alpha)$ and the verification of commutativity are similar:

$$\begin{aligned}
\mathsf{cod}(\alpha)_X\big(f : \mathcal{I}(\mu)(X) \longrightarrow \mathcal{U}_n\big) &= \mathsf{ModCode}_\mu \circ \widehat{f} : X \longrightarrow \mathcal{U}_m \\
\mathsf{cod}(\alpha)_X\big(p : E \longrightarrow B, f : \mathcal{I}(\mu)(X) \longrightarrow B\big) &= \big(m_\mu(p) : \mu(E) \longrightarrow R_\mu B, \widehat{f} : X \longrightarrow R_\mu(B)\big)
\end{aligned}$$

It now follows nearly tautologically that the Tarski universe $\mathcal{U}_m$ at mode m is closed strictly under modal types by restricting α to the subobject of $\mathcal{I}(\mu)^*\mathbf{y}(\mathcal{U}_n)$ of $\mathcal{I}(\mu)^*\mathcal{T}_n$.

Modal elimination principles Given a modality $\mu : n \longrightarrow m$, we must now argue that α_μ enjoys a lifting structure witnessing modal elimination. Let us write M for the product $\mathcal{I}(\mu)^*\mathcal{T}_n \times_{\mathcal{T}_m} \mathcal{T}^\bullet_m$ and write i for the induced map $\mathcal{I}(\mu)^*\mathcal{T}^\bullet_n \longrightarrow M$.

Following Awodey [Awo18], we must construct a family of diagonal lifts fitting into the following square, naturally in Z:

$$\begin{tikzcd}
Z \times_{\mathcal{I}(\mu)^*\mathcal{T}^\bullet_n} \mathcal{I}(\mu)^*\mathcal{T}^\bullet_n \arrow[r] \arrow[d] & \mathcal{I}(\mu)^*\mathcal{T}^\bullet_n \times \mathcal{T}^\bullet_m \arrow[d] \\
Z \times_{\mathcal{I}(\mu)^*\mathcal{T}^\bullet_n} M \arrow[r] & \mathcal{I}(\mu)^*\mathcal{T}^\bullet_n \times \mathcal{T}_m
\end{tikzcd}$$

As described in Proposition 30 of op. cit., it suffices to consider the case where Z is representable, and so we may reduce the case where $Z = \lfloor A \rfloor : \mathbf{y}(X) \longrightarrow \mathcal{I}(\mu)^*\mathcal{T}_n$. Let us suppose that $\lfloor A \rfloor$ factors through the right disjunct of $\mathcal{T}_n$—our assumptions on $\mathsf{ModCode}_\mu$ ensure the other case reduces easily to this—so that it may be decomposed into a display map $p : E \longrightarrow B \in \mathscr{D}_n$ along with a map $f : \mathcal{I}(\mu)(X) \longrightarrow B$.

In this case, we may transpose the above square and calculate, so that it suffices to obtain a family of natural lifts to a diagram of the following shape:

$$\begin{tikzcd}
\mathbf{y}(X \times_{R_\mu(B)} R_\mu(E)) \arrow[r] \arrow[d] & \mathcal{T}^\bullet_m \arrow[d] \\
\mathbf{y}(X \times_{R_\mu(B)} \mu(E)) \arrow[r] & \mathcal{T}_m
\end{tikzcd}$$

Now utilizing the definition of τ_m, we may further reduce this to finding a natural family of lifts for the following diagram for some fixed display map $F \longrightarrow C \in \mathscr{D}_m$:

$$\begin{tikzcd}
X \times_{R_\mu(B)} R_\mu(E) \arrow[r] \arrow[d] & F \arrow[d] \\
X \times_{R_\mu(B)} \mu(E) \arrow[r] & C
\end{tikzcd}$$

Finally, Proposition 31 in Awodey [Awo18] shows that there is a universal such X, so we can solve this lifting problem naturally by reducing to this universal case and observing that the left-hand map is anodyne.

While we leave the details to the reader, an identical argument suffices to show that μ-modal types satisfy a ν-crisp induction principle if $R_\nu(i_\mu)$ is stably anodyne.

Modal dependent products The case of modal dependent products follows the case of dependent products detailed by Shulman [Shu19] and Shulman [Shu23] *mutatis mutandis* as we have required display maps to be closed under pushforward along maps $R_\mu(p)$ where p is a display map. We outline the details for completeness.

Fix a modality $\mu : n \longrightarrow m$. We must define a cartesian map classifying modal dependent products $\alpha : \mathbf{P}_{\mathcal{I}(\mu)^*\tau_n}(\tau_m) \longrightarrow \tau_m$. We first note that by exactness, it suffices to construct cartesian squares $\beta : \mathbf{P}_{\mathcal{J}(\mu)^*\lambda_n}(\lambda_m) \longrightarrow \lambda_m$ and $\mathbf{P}_{\mathcal{J}(\mu)^*\mathbf{y}(\upsilon_n)}(\mathbf{y}(\upsilon_m)) \longrightarrow \mathbf{y}(\upsilon_m)$ separately. The latter exists by assumption, so we must construct only β.

We begin with $\mathsf{cod}(\beta)$. To this end, fix the input to $\mathsf{cod}(\beta)$ at $X : \mathcal{J}(m)$: two types $\big(p : E_n \longrightarrow B_n, f : \mathcal{J}(\mu)(X) \longrightarrow B_n\big)$ and $\big(q : E_m \longrightarrow B_m, g : X \times_{R_\mu(B_n)} R_\mu(E_m) \longrightarrow B_m\big)$.

We begin by defining a $\pi : B_n \rhd_\mu B_m \longrightarrow R_\mu(B_n)$ as follows:

$$\pi = R_\mu(p)_*(R_\mu(E_n) \times B_m)$$

By transposition, f and g induce a map $\langle f, g\rangle : X \longrightarrow B_n \rhd_\mu B_m$ and this is evidently natural in X. Note that the counit of $R_\mu(p)^* \dashv R_\mu(p)_*$ induces $e : \pi^*(R_\mu(E_n)) \longrightarrow B_m$. Finally, we define a display map $p \rhd_\mu q$ over $B_n \rhd_\mu B_m$ as follows:

$$p \rhd_\mu q = \pi^*(R_\mu(p))_*(e^*(q))$$

Finally, we define $\mathsf{cod}(\beta)_X$ as follows:

$$\mathsf{cod}(\beta)_X((p, f), (q, g)) = (p \rhd_\mu q, \langle f, g\rangle)$$

An identical calculation to e.g. Shulman [Shu23] shows that this map induces the necessary cartesian square. □

7.2.4 Models of MTT from gluing

We conclude this section with an instance of a semantic model which does *not* arise from Theorem 7.2.10. This model adapts the gluing model for AdjTT elucidated in Section 5.4. For the remainder of this section, fix a pair of locally Cartesian closed display map categories $(\mathcal{C}, \mathscr{D}_\mathcal{C})$ and $(\mathcal{D}, \mathscr{D}_\mathcal{D})$, along with an adjunction $F \dashv G$ between them such that both F and G preserve display maps and finite limits. We will assume $\mathscr{D}_\mathcal{D}$ and $\mathscr{D}_\mathcal{C}$ are closed under the composition, identity, diagonals, pushforward, and the other standard operations required to interpret Martin-Löf type theory (Section 3.5.2). We will further require both $\mathcal{C}$ and $\mathcal{D}$ admit a universe of small display maps closed $\upsilon_\mathcal{C}$ and $\upsilon_\mathcal{D}$ respectively. These universes are closed under all connectives, and there exist cartesian squares $\alpha : F(\upsilon_\mathcal{C}) \longrightarrow \upsilon_\mathcal{D}$ and $\beta : F(\upsilon_\mathcal{D}) \longrightarrow \upsilon_\mathcal{C}$ respectively.

We will show that we obtain a model of MTT with the adjoint mode theory $\mathcal{M}_{\mathsf{adj}}$ introduced in Example 6.1.6, though without μ-crisp ν-elimination. We note, however, that this principle is validated if $F \dashv G$ presents an idempotent comonad.

Recall from Section 5.4 that gluing along F results in the following chain of adjoints:

$$\begin{tikzcd}
\mathcal{C} \arrow[r, "\mathbf{j}_*" description] & \mathbf{Gl}(F) \arrow[l, shift right=4, "\mathbf{j}^*" description] \arrow[l, shift left=4, "\mathbf{j}^!" description] \arrow[r, shift left=2, "\mathbf{i}^*" description] & \mathcal{D} \arrow[l, shift left=2, "\mathbf{i}_*" description]
\end{tikzcd}$$

We will show that we can interpret MTT with adjoint modalities $\nu \dashv \mu$ into $\mathcal{C}$ and $\mathbf{Gl}(F)$ using $\mathbf{j}^!$ to interpret μ and $\mathbf{j}_*$ to interpret ν. As both of these are right adjoints, one might hope that Theorem 7.2.10 applies. This is, however, not the case. In particular, we will endow $\mathbf{Gl}(F)$ with a class of display maps in such a way that $\mathbf{j}_*$ does not send display maps to display maps. Accordingly, we must make use of Theorem 7.2.21 to accommodate the fact that ν is a proper weak DRA.

Definition 7.2.23. The class $\mathscr{D}_{\mathbf{Gl}(F)} \subseteq \mathbf{Gl}(F)^{\to}$ consists of maps $f : X \longrightarrow Y$ such that the naturality square of the unit of $\mathbf{i}^* \dashv \mathbf{i}_*$ is cartesian and $\mathbf{i}^* f \in \mathscr{D}_{\mathbb{D}}$.

It is helpful to unfold this definition slightly. First, note that for a given object $X = D \longrightarrow F(C)$ we may compute:

$$\mathbf{i}_*\mathbf{i}^* X = D \longrightarrow F(\mathbf{1})$$

Accordingly, $\eta : \mathsf{id} \longrightarrow \mathbf{i}_* \circ \mathbf{i}^*$ becomes the identity when restricted to closed component and it sends the open component to the terminal object. Requiring the naturality square at $f : X \longrightarrow Y$ to be cartesian is equivalent, therefore, to requiring f to be an isomorphism on open components; $\mathbf{j}^* f$ must be invertible.

Moreover, this class is minimal in a certain sense:

Lemma 7.2.24. *$\mathscr{D}_{\mathbf{Gl}(F)}$ is the smallest pullback stable class containing $\mathbf{i}_*\mathscr{D}_{\mathbb{D}}$.*

Proof. By construction, any pullback stable class containing $\mathbf{i}_*\mathscr{D}_{\mathbb{D}}$ must contain $\mathscr{D}_{\mathbf{Gl}(F)}$. It remains only to argue that $\mathscr{D}_{\mathbf{Gl}(F)}$ is closed under pullbacks. To this end, fix $g \in \mathscr{D}_{\mathbf{Gl}(F)}$ and fix a cartesian square:

$$\begin{tikzcd}
A \arrow[r] \arrow[d, "f"'] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d, "g"] \\
B \arrow[r] & Y
\end{tikzcd}$$

We must show that the unit square for g is cartesian. First, note that as $f \in \mathscr{D}_{\mathbf{Gl}(F)}$ the square $g \longrightarrow \mathbf{i}_*\mathbf{i}^* f$ given by composing the above square with the unit at f is cartesian. Next since $\mathbf{i}^*$ preserves pullbacks, the out square and right-hand squares in the following are pullbacks:

$$\begin{tikzcd}
A \arrow[r] \arrow[d, "g"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{i}_*\mathbf{i}^* A \arrow[r] \arrow[d, "\mathbf{i}_*\mathbf{i}^* g"] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{i}_*\mathbf{i}^* X \arrow[d, "\mathbf{i}_*\mathbf{i}^* f"] \\
B \arrow[r, "\eta"'] & \mathbf{i}_*\mathbf{i}^* B \arrow[r] & \mathbf{i}_*\mathbf{i}^* Y
\end{tikzcd}$$

The conclusion now follows. □

Lemma 7.2.25. *The class $\mathscr{D}_{\mathbf{Gl}(F)}$ is closed under identities, diagonals, pushfoward and composition.*

Proof. All cases except that of pushforwards follow from elementary manipulation of pullbacks. Fix $f : A \longrightarrow X, g : X \longrightarrow Y \in \mathscr{D}_{\mathbf{Gl}(F)}$, we must show that $g_* f \in \mathscr{D}_{\mathbf{Gl}(F)}$. First, recall that $\mathbf{j}^*$ preserves pushforwards. Accordingly, $\mathbf{j}^*(g_* f)$ is invertible as the pushforward of an invertible map: $\mathbf{j}^* f$. Accordingly, the unit square associated with $g_* f$ is cartesian.

It remains to show that $\mathbf{i}^*(g_* f) \in \mathscr{D}_{\mathbb{D}}$. Through a standard reduction, we may pass to the slice over Y and reduce the case where $Y = \mathbf{1}$. In particular, this slice can also be realized by gluing along a lex functor between display map categories. In this case, $g_* f$—now a family over $\mathbf{1}$—can be defined through a slightly more direct pullback diagram:

$$\begin{tikzcd}
g_* f \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & A^X \arrow[d] \\
\mathbf{1} \arrow[r] & X^X
\end{tikzcd}$$

As $\mathbf{i}^*$ preserves pullbacks, to show $\mathbf{i}^*(g_* f) \longrightarrow \mathbf{1} \in \mathscr{D}_{\mathcal{D}}$, it suffices to argue that $\mathbf{i}^*(X^X) \longrightarrow \mathbf{i}^*(A^X)$ is a display map. Computing, this map arises by pulling back the map $\mathbf{i}^* X^{\mathbf{i}^* X} \longrightarrow \mathbf{i}^* A^{\mathbf{i}^* X}$. This latter map, in turn, is a display map through our assumption that $\mathscr{D}_{\mathcal{D}}$ is closed under pushforward. □

Lemma 7.2.26. *The universe $\upsilon_{\mathbf{Gl}(F)} = \mathbf{i}_* \upsilon_{\mathcal{D}}$ is closed under all the connectives of type theory.*

Proof. This follows immediately from the fact that $\mathbf{i}_*$ preserves locally cartesian closed structure and all the connectives of type theory are definable in this language. □

Theorem 7.2.27. *There is a model of MTT with the adjoint mode theory in $\mathcal{C}$ and $\mathbf{Gl}(F)$ arising from the pseudofunctor given by the adjunction $\mathbf{j}^* \dashv \mathbf{j}_*$ such that the right adjoint modality is a DRA, but there is no crisp induction principle for the left adjoint modality framed by the right.*

Proof. We will construct this model via Theorem 7.2.21. In this case, we are using extensional equality and thus Lemma 6.3.9 shows that it sufficient to construct ordinary products; we may use them and modal types to encode modal dependent products. Accordingly, in light of our assumptions and Lemmas 7.2.25 and 7.2.26, it remains only to discuss the prerequisites of Theorem 7.2.21 governing modalities. In fact, there is no need to explicitly argue for crisp induction principles for modalities: we will show the right adjoint is a DRA and thus automatically has all crisp induction principles, while the left adjoint is not required to have any; the only non-trivial case for the latter is the right adjoint frame, as $\mathbf{j}^*$ is fully faithful.

Let us write $\tau_{\mathcal{C}} : \mathcal{T}^{\bullet}_{\mathcal{C}} \longrightarrow \mathcal{T}_{\mathcal{C}}$ and $\tau_{\mathbf{Gl}(F)} : \mathcal{T}^{\bullet}_{\mathbf{Gl}(F)} \longrightarrow \mathcal{T}_{\mathbf{Gl}(F)}$ for the universes of type and terms in $\mathcal{C}$ and $\mathbf{Gl}(F)$, respectively. We similarly write $\upsilon_{\mathcal{C}}$ and $\upsilon_{\mathbf{Gl}(F)}$ for the small universes. We begin by constructing the interpretation of the right adjoint modality sending $\tau_{\mathbf{Gl}(F)}$ to $\tau_{\mathcal{C}}$.

The right adjoint While the preconditions of Theorem 7.2.21 only require us to produce a weak DRA, we will use Lemma 7.2.5 and construct a full DRA over $\mathbf{j}_*$.

To this end, note that right adjoint to $\mathbf{j}_*$ ($\mathbf{j}^!$) sends display maps to display maps. Indeed, by assumption G preserves display maps and $\mathbf{j}^!$ sends an arrow f in $\mathscr{D}_{\mathbf{Gl}(F)}$ to a pullback of $G\mathbf{i}^* f$. The result then follows because $\mathbf{i}^*$ sends display maps in $\mathbf{Gl}(F)$ to display maps in $\mathcal{D}$ by assumption. We therefore obtain a pre weak DRA over $\mathbf{j}^!$ by choosing a trivial factorization $i(f) = \mathsf{id}$.

Moreover, by assumption we have a cartesian square $G(\upsilon_{\mathcal{D}}) \longrightarrow \upsilon_{\mathcal{C}}$. It therefore suffices to construct a cartesian square $\mathbf{j}^{!}\mathbf{i}_{*}\upsilon_{\mathcal{D}} \longrightarrow G(\upsilon_{\mathcal{D}})$, but calculation shows that these two maps are in fact isomorphic.

Taken together, these two constructions exhibit the necessary modal types for the right adjoint and show that the universe of small types is closed under this modality.

The left adjoint The left adjoint is more complex, essentially because it is *not* a DRA. We begin again by constructing a pre weak DRA over $\mathbf{j}_{*}$ which sends a display map $f \in \mathscr{D}_{\mathcal{C}}$ to a factorization of $\mathbf{j}_{*}f$ into a stably anodyne map followed by a closed ètale map. Explicitly, given $f : X \longrightarrow Y$ we choose the following factorization:

$$\begin{tikzcd}
F(X) \arrow[r] \arrow[d] & F(X) \arrow[d] & & F(X) \arrow[r] \arrow[d] & F(Y) \arrow[d] \\
F(X) \arrow[r] & F(Y) & & F(Y) \arrow[r] & F(Y)
\end{tikzcd}$$

It is routine to verify that the left square $i(f)$ is stably anodyne and the right square $m(f)$ lies in $\mathscr{D}_{\mathbf{Gl}(F)}$.

It remains to show that $\upsilon_{\mathbf{Gl}(F)}$ is suitably closed under these modal types. That is, we must choose a morphism $\mathsf{ModCode}_{\nu} : \mathbf{j}_{*}\upsilon_{\mathcal{C}} \longrightarrow \upsilon_{\mathbf{Gl}(F)}$ such that given a display map $\pi : E \longrightarrow B \in \mathscr{D}_{\mathcal{C}}$ classified by $f : B \longrightarrow \mathcal{U}_{\mathcal{C}}$, the map $\mathsf{ModCode}_{\nu} \circ \mathbf{j}_{*}f$ classifies $m(f)$. Recall that $\upsilon_{\mathbf{Gl}(F)} = \mathbf{i}_{*}\upsilon_{\mathcal{D}}$ and so, transposing, it suffices to construct a map $\mathbf{i}^{*}\mathbf{j}_{*}\upsilon_{\mathcal{C}} \longrightarrow \upsilon_{\mathbf{Gl}(F)}$. Calculating, the domain of this morphism is $F(\upsilon_{\mathcal{C}})$ and so we choose the assumed cartesian square $\alpha : F(\upsilon_{\mathcal{C}}) \longrightarrow \upsilon_{\mathcal{D}}$.[1]

It remains to show that α has the required property. To this end, fix $\pi : E \longrightarrow B$ and suppose that it is classified by $f : B \longrightarrow \mathcal{U}_{\mathcal{C}}$. Chasing through the transposition, $\alpha \circ \mathbf{j}_{*}f$ is determined by the following square:

$$\begin{tikzcd}
F(B) \arrow[r, "\alpha \circ F(f)"] \arrow[d] & \mathcal{U}_{\mathcal{D}} \arrow[d] \\
F(B) \arrow[r, "!"'] & \mathbf{1}
\end{tikzcd}$$

Directly calculating, the display map classified by this arrow is given by the following:

$$\begin{tikzcd}
F(E) \arrow[r] \arrow[d] & F(B) \arrow[d] \\
F(B) \arrow[r] & F(B)
\end{tikzcd}$$

This is precisely $m(f)$, just as required. □

[1] Note that despite this last square being cartesian, the induced map $\mathsf{ModCode}_{\nu}$ will not typically be cartesian. Transposing a cartesian square does not generally yield another cartesian square.

Corollary 7.2.28. *If $F \dashv G$ presents an idempotent comonad so $\eta : \mathsf{id} \longrightarrow G \circ F$ is an isomorphism, the model constructed in Theorem 7.2.27 has all crisp induction principles.*

Proof. It suffices to consider the elimination principle for the left adjoint modality with the right adjoint as the frame. Recall the factorization $i(-), m(-)$ constructed in Theorem 7.2.27 to interpret the left adjoint modality. Following Theorem 7.2.21, it suffices to show that $\mathbf{j}^!(i(f))$ is stably anodyne for all display maps $f : X \longrightarrow Y$.

Calculating, we see that $\mathbf{j}^! i(f) = GFX \to GFX \times_{GFY} Y$. If the unit of $F \dashv G$ is an isomorphism, this map is an isomorphism and therefore stably anodyne. □

In particular, we obtain the following useful class of models of MTT:

Corollary 7.2.29. *If $F \dashv G$ presents an idempotent comonad, then Theorem 7.2.27 models all of MTT with extensional equality for the adjoint mode theory. In addition, the interpretation of $\langle \mu \mid - \rangle$ can be extended to a strict dependent right adjoint.*

Corollary 7.2.30. *Any lex idempotent comonad on a locally cartesian closed category induces a model of MTT with the mode theory from Example 6.1.4.*

7.3 Independence of various extensions of MTT

We conclude this chapter by using the machinery developed thus far to prove the independence of various extensions of MTT from the base theory. In particular, in Section 6.3, we introduced two classes of extensions for MTT:

- Crisp induction principles for proving $\langle \mu \mid \mathsf{Id}(a, b) \rangle \simeq \mathsf{Id}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b))$.
- Strict DRAs that presented an internal right adjoint modality by a type with an η law $\mu \Rightarrow -$.

In this section, we show construct models which refute the first principle along with a strengthening of the second.

In particular, we construct two models which refute the equivalence $\langle \mu \mid \mathsf{Id}(a, b) \rangle \simeq \mathsf{Id}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b))$. The second model is significantly more involved but shows that this holds even when μ is a (strict) internal right adjoint.

We also show that not every weak dependent right adjoint is a strict DRA, even in the presence of equality reflection. This shows that a natural strengthening of the second extension (making every modality a strict DRA) is independent of MTT. This model does not show, however, that wDRAs realizing internal right adjoints are not always strict dependent right adjoints. Constructing a countermodel for the second extension is more subtle and we defer it til Chapter 8 where one can argue by induction on normal forms (Theorem 8.6.12).

Recall that the presence of equality reflection trivializes crisp induction (Lemma 6.3.4). However, extensional equality is almost always validated by a model unless the model comes from syntax or from a homotopical model where identity types are interpreted by path objects. To find models which refute crisp induction, we therefore turn to syntactic and homotopical models. We now specify two such models of MTT—one syntactic, one homotopical.

Lemma 7.3.1. *There exists a model of MTT with a single mode m and modality $\mu : m \longrightarrow m$ such that $\langle \mu \mid \mathsf{Id}(a, b)\rangle$ is not equivalent to $\mathsf{Id}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b))$.*

Proof. Consider the syntactic model of MLTT. We will construct a model of MTT using this model to interpret mode m. We interpret $\langle \mu \mid A\rangle$ as $\prod_{\mathsf{Nat}} A$. In particular, as $\mathsf{Nat} \to -$ is a dependent right adjoint with left adjoint $\Gamma \mapsto \Gamma.\mathsf{Nat}$, Theorem 7.2.10 ensures that this assembles into a model of MTT.

Unfolding, the equivalence under consideration is precisely function extensionality, which is well-known to not hold in the syntactic model of MLTT. □

Lemma 7.3.2. *There exists a model of MTT with the adjoint mode theory with modalities $\nu \dashv \mu$ such that $\langle \mu \mid \mathsf{Id}(a, b)\rangle$ is not equivalent to $\mathsf{Id}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b))$*

Proof. Let us recall that $\mu : n \longrightarrow m$. We will interpret n in **Set** and m in simplicial sets **sSet**. Crucially, types in the latter will not be arbitrary small families but small simplicial sets as in the simplicial model of homotopy type theory [KL21]. In particular, display maps in **Set** are taken to be fiberwise-small families and display maps in **sSet** are fiberwise-small Kan complexes.

We will interpret the right adjoint modality as a DRA internalizing $\Gamma : \mathbf{sSet} \longrightarrow \mathbf{Set}$ while the left adjoint is interpreted by $\Delta : \mathbf{Set} \longrightarrow \mathbf{sSet}$. Both are adjoints: they form the rightmost half of $\Pi_0 \dashv \Delta \dashv \Gamma$. Finally, Theorem 7.2.10 then applies, as both right adjoints preserve display families.

Within the simplicial set model, identity types are interpreted by paths and therefore may be arbitrarily complex and proof-relevant types. On the other hand, identity types in **Set** are always propositional. It follows that $\langle \mu \mid - \rangle$ cannot preserve identity types as Γ takes the identity type of e.g., S^1 (the integers) to $\mathbf{1}$. □

Finally, we note that our model in $\mathbf{Gl}(F)$ provides a plentiful supply of semantically-natural weak dependent right adjoints which are *not* dependent right adjoints. In particular, even in the presence of equality reflection a weak DRA is not equivalent to a strong DRA.[2]

Lemma 7.3.3. *The left adjoint in Theorem 7.2.27 is never interpreted by a dependent right adjoint except in trivial cases.*

Proof. Inspecting the construction of the model, we see that the left adjoint is implemented by a DRA if for any $f : X \longrightarrow Y \in \mathscr{D}_{\mathfrak{C}}$ the factorization of $\mathbf{j}_* f$ into an anodyne map $i(f)$ and a display map $m(f)$ is trivial; that is, when $i(f)$ is an isomorphism. Recall that $i(f)$ is given by the following commuting square, which is not invertible unless f was invertible:

$$\begin{tikzcd}
F(X) \arrow[r] \arrow[d] & F(X) \arrow[d] \\
F(X) \arrow[r] & F(Y)
\end{tikzcd}$$

In particular, the interpretation of $\langle \nu \mid - \rangle$ is not a DRA. □

[2] This is in contrast to “weak coproducts” or similar, which do collapse to coproducts in the presence of equality reflection.

Corollary 7.3.4. *Extending extensional MTT with an elimination principle akin to Rule 5.21 is not a conservative extension.*

7.4 Relating modal type theories to MTT

Having developed the semantics of MTT in this chapter, we are now well-positioned to discuss and prove relationships between MTT and several of the type theories discussed in Chapter 5. In particular, we argue that MTT should be seen as a unifying modal type theory. The basis for this comparison is the following series of results, each of which follows from the results previously established or inspection on the definition of models for the relevant theories.

Definition 7.4.1. We write $\mathcal{M}_{\mathsf{icom}}$ for the refinement of the walking adjunction $\mathcal{M}_{\mathsf{adj}}$ which further requires that the unit 2-cell η is invertible. This is the *walking colocalization* which presents an idempotent comonad.

Theorem 7.4.2. *The syntax of AdjTT is a model of MTT with $\mathcal{M}_{\mathsf{icom}}$ without μ-crisp ν-induction.*

Proof. Comparing the definition of a model of AdjTT in Section 5.4 with MTT instantiated with $\mathcal{M}_{\mathsf{icom}}$, we see that the former supports all the required structure of the latter except the necessary lifting structure for μ-crisp ν-induction.

We may unfold this interpretation slightly to obtain the following:

1. A mode m context in MTT is realized by a single-context in AdjTT.

2. A mode n context in MTT is realized by a dual-context in AdjTT.

3. The restriction $-.\{\mu\}$ is interpreted by $\Delta \mapsto \Delta; \mathbf{1}$.

4. The restriction $-.\{\nu\}$ is interpreted by $\Delta; \Gamma \mapsto \Delta$.

5. The modal types $\langle \mu \mid - \rangle$ and $\langle \nu \mid - \rangle$ are realized by $\mathcal{R}$ and $\mathcal{L}$, respectively.

We note in particular that $-.\{\nu\} \circ -.\{\mu\} = \mathsf{id}$ and, in particular, the realization of the unit 2-cell is the identity.[3]

More subtle perhaps is the realization of modal context extensions. As $\langle \mu \mid - \rangle$ is interpreted by $\mathcal{R}$, a dependent right adjoint, extending a context by an element of A with a μ-annotated extension is realized by ordinary context extension by $\mathcal{R}[\![A]\!]$. The same is not true for ν, as $\mathcal{L}$ is not a dependent right adjoint. Extending $[\![\Gamma]\!] = [\![\Gamma]\!]_0; [\![\Gamma]\!]_1$ by A with a ν-annotation is interpreted by $[\![\Gamma]\!]_0.[\![A]\!]; [\![\Gamma]\!]_1$ (see Lemma 5.4.8). Inspecting universal properties, extension by a $\nu \circ \mu$-annotation variable is realized by combining the above two procedures, so extending $[\![\Gamma.(\nu \circ \mu \mid A)]\!]$ becomes $[\![\Gamma]\!]_0.\mathcal{R}[\![A]\!]; [\![\Gamma]\!]_1$. Finally, we note that since $[\![-.\{\mu \circ \nu\}]\!] = \mathsf{id}$, extension by a $\mu \circ \nu$-annotated variable is necessarily realized by ordinary extension. □

[3] We remind the reader that the interpretation of contexts in a model of MTT must form a pseudo-functor out of $\mathcal{M}^{\mathsf{coop}}$ rather than $\mathcal{M}$. This accounts for the reversal of μ and ν in the above formula when compared with $\mathcal{M}_{\mathsf{icom}}$.

Corollary 7.4.3. *The syntax of* AdjTT *specialized to a colocalization is a model of* MTT *with the adjoint mode theory.*

Proof. Specializing AdjTT to a colocalization means assuming an inverse ι to the canonical map $\eta : A \to \mathcal{R}\,\mathcal{L}\,A$. In particular, we assume that $\iota(\mathsf{mod}_{\mathcal{R}}(\mathsf{mod}_{\mathcal{L}}(M))) = M$. Such an inverse can then be used to *define* the relevant crisp induction principle using the dual-context equivalent of Lemma 6.3.5. Concretely:

$$[\![\mathsf{let}_{\mu}\ \mathsf{mod}_{\nu}(x) \leftarrow M_0\ \mathsf{in}\ M_1]\!] = M_1[\iota^{-1}(\mathsf{mod}_{\mathcal{R}}([\![M_0]\!]))/x]$$

The computation principle for this rule follows directly from the assumption that ι is an inverse to η. □

Theorem 7.4.4. DRA *is a model of* MTT *with a the single endomodality (Example 6.1.1).*

Proof. This is an immediate consequence of the definition of a model of DRA given in Section 5.5 along with Theorem 7.2.10. □

Theorem 7.4.5. FitchTT *with mode theory* $\mathcal{M}$ *is a model of* MTT *with the same mode.*

Proof. This is again a consequence of Theorem 7.2.10. □

Recall that an ordinary adjunction between categories with display maps induces a dependent adjunction when the right adjoint preserves display maps (Lemma 5.5.10). Consider a pseudofunctor $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ such that $F(m)$ is a category with display maps and each $F(\mu)$ is a right adjoint preserving those display maps. If we further assume that the left adjoint to $F(\mu)$ is a parametric adjunction, this forms the basis of a model of FitchTT with mode theory $\mathcal{M}$. The above result shows that it also constitutes a model of MTT with the same mode theory, but often more is true:

Theorem 7.4.6. *Considering* $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ *as described above, if the left adjoint to* $F(\mu)$ *preserves the terminal object and display maps for each* μ *then* F *induces a model of* MTT *instantiated with the 2-category* $\mathcal{M}[\mathcal{M}^*]$*—the 2-category which freely adds left adjoints to each 1-cell in* $\mathcal{M}$*. This model also supports strict dependent right adjoints.*

Proof. This is an immediate consequence of Theorem 7.2.10 after noting that a parametric right adjoint that preserves terminal objects is precisely a right adjoint. □

In particular, these theorems show that MTT can be applied in any of the situations presently addressed by DRA (or variant Fitch-style calculi), AdjTT, or FitchTT. Whether switching from one of these calculi to MTT is advantageous depends on the particulars of the situation.

Adjoint modalities Either AdjTT or MTT may be used to reason about a pair of display-map-preserving adjoint functors. It appears that little is gained or lost passing between MTT and AdjTT. The category of models is nearly identical, with the only point of divergence being whether or not the modal restriction $-.\{\nu\}$ has cartesian lifts for display maps. In practice, this appears to matter very little and so the better developed metatheory of MTT (Chapter 8) points to using it over AdjTT.

A single dependent right adjoint When considering a single dependent right adjoint, one may choose between MTT or any of the myriad Fitch-style type theories refining DRA to better capture the properties of the dependent adjunctions. In this case, the situation is less clear-cut: Fitch-style type theories offer additional definitional equalities and while it is possible to encode $\mathsf{let}\ \mathsf{mod}_{\mu}(-) \leftarrow - \ \mathsf{in}\ -$ in terms of $\mathsf{unmod}(-)$, the reverse is untrue.

Accordingly, while one still benefits from the uniformity of MTT, in some circumstances pen-and-paper calculations may be simpler with a specialized calculus with a more powerful elimination rule [GSB19a; HP23; VRT22]. While it may require additional work to adapt Fitch-style type theories to a new situation, this work has been done for a select few mode theories already and there is little reason not to take advantage of it.

The lack of uniformity does mean that it would be difficult to develop a proof assistant based around Fitch-style type theories which offers the same flexibility as e.g., what is proposed by Stassen et al. [SGB23a]. It remains unclear whether the convenience offered by the strong elimination rule balances out the more complex syntax, but this question can only be addressed with further experience and experimentation.

Multiple dependent right adjoints When one considers multiple dependent right adjoints, it is nearly always better to work with MTT and, frequently, it is the only option. The only potential exception to this rule arises when reasoning about a pseudofunctor $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ where each $F(\mu)$ has a left adjoint which is a PRA which *does not* preserve the terminal object. In this situation, FitchTT may fit better as it contains a more powerful collection of elimination rules. However, even in such a specialized situation, the decision is not completely clear-cut.

FitchTT does not have the same suite of metatheorems available as MTT or several other Fitch-style type theories and it remains unknown whether it admits e.g., a normalization algorithm. Accordingly, if one is aiming to build build proof assistant to mechanize certain arguments, MTT is still the better option. Even when working on pen-and-paper, the additional context operations in FitchTT are complex and no equivalent to the “informal MTT discipline” offered in Chapter 6 has been put forth.

Finally, and most compelling, asking for a parametric adjunction is a substantial requirement and it is somewhat uncommon that this PRA is not simply an ordinary adjunction—the major exception is $A \times -$ as discussed in Section 5.6.1. This usually means that one is better served by MTT instantiated with $\mathcal{M}[\mathcal{M}^*]$ as described in Theorem 7.4.6 and using the extensions discussed in Section 6.3 to recover all the advantages of FitchTT along with additional modalities.

8 Normalization for MTT

> [T]he purpose of categorical algebra is to show that which is formal is formally formal.
>
> J. P. May
> *Picard groups, Grothendieck rings, and Burnside rings of categories.*

Remark 8.0.1. The material of this section is largely drawn from Gratzer [Gra23], an extended version of Gratzer [Gra22]. The introductory material on MTT has been removed—it is redundant in light of Chapters 6 and 7. Furthermore, Section 8.7 has been extended to include a discussion of strict dependent right adjoints. Numerous small changes to the exposition and text have also been made to ensure cohesion with the surrounding material. ◇

In this chapter we construct a normalization algorithm for MTT equipped with the full suite of connectives: dependent sums, products, booleans, intensional identity types, a universe, and modal types. In addition to the usual corollaries of normalization (decidability of type checking, injectivity of type constructors, etc.), this sharpens the canonicity result of Gratzer et al. [Gra+20a]. This algorithm applies to any choice of mode theory and therefore simultaneously establishes normalization results for all the instantiations of MTT discussed those far as well as those in Part III.

In order to prove this result, we advance modern gluing techniques to apply to modal type theories and demonstrate that extensional MTT itself is a suitable metalanguage for carrying out the proof of normalization-by-gluing. We further argue that these techniques scale by extending the proof to accommodate both with crisp induction principles and strict dependent right adjoints for internal right adjoint modalities.

8.1 Introduction

The central complexity in this proof is the generality of MTT. MTT can be instantiated with an arbitrary collection of modalities and transformations between them and these instantiations may behave quite differently.

While this flexibility allows MTT to accommodate many interesting calculi, it becomes proportionally more challenging to prove metatheoretic results about MTT. In particular, the rich substitution structure inherited from the mode theory can introduce subtle equations between terms. The proof that the crisp induction principles can be reconstructed in MTT given in Section 6.4, for instance, exemplifies this and hinges on many such calculations. In fact, the metatheoretic results established by Gratzer et al. [Gra+20a] (soundness and canonicity) or the proof given in Corollary 7.2.15 are results on closed terms in MTT, allowing their proofs to avoid the majority of the substitution apparatus.

Crucially, it remained open whether MTT admitted a normalization algorithm and, consequently, whether type checking was decidable. Even in the presence of a normalization algorithm MTT cannot admit an unconditional type checking algorithm: it is not only necessary to have a decision procedure for terms in the language, but also for modalities and 2-cells as both appear in terms for MTT.

In this paper we show the best possible result holds: MTT admits an unconditional normalization algorithm and conversion of normal forms is decidable if and only if conversion is decidable in the mode theory. As corollaries, we show that type constructors in MTT are always injective and that type checking is decidable when the mode theory is decidable.[1]

8.1.1 Normalization-by-evaluation

A normalization algorithm must begin by defining *normal forms*. Their precise formulation varies depends on the situation but they always satisfy two crucial properties. First, the equality of normal forms $u = v$ is clearly decidable—often no more than structural equality—and there is a function $\mathbf{dec}(u)$ decoding a normal form to a term of the same type.

Relative to a notion of normal form, a normalization algorithm sends a term $\Gamma \vdash M : A$ to a normal form $\mathbf{nf}_\Gamma(M, A)$ such that $(\mathbf{nf}_\Gamma(-, A), \mathbf{dec}(-))$ lifts to an isomorphism between equivalence classes of terms of A and normal forms [Abe13]. Typically one breaks the condition that $(\mathbf{nf}_\Gamma(-, A), \mathbf{dec}(-))$ forms an isomorphism into three conditions:

1. *Completeness*: if $\Gamma \vdash M = N : A$ then $\mathbf{nf}_\Gamma(M, A) = \mathbf{nf}_\Gamma(N, A)$.

2. *Soundness*: $\Gamma \vdash \mathbf{dec}(\mathbf{nf}_\Gamma(M, A)) = M : A$.

3. *Idempotence*: $u = \mathbf{nf}_\Gamma(\mathbf{dec}(u), A)$.

Proving normalization is an involved affair. Traditionally, one begins by fixing a strongly normalizing confluent rewriting system presenting the equational theory of the type theory. The normal forms are then exactly the terms of the theory which cannot be further reduced. This approach does not scale, however, to type theories with *type-directed* equations such as the unicity principles of dependent sums and the unit type. These equations defy attempts to present them in a rewriting system and require type-directed algorithms.

The main type-directed technique for normalization is *normalization-by-evaluation (NbE)* [Abe13]. Proving that an NbE algorithm works, however, is an extremely intricate affair involving a variety of complex constructions. After the algorithm is defined, the proof of correctness typically proceeds by establishing properties (1)-(3) in order. Each property, moreover, requires a separate argument. Completeness is established through

[1] This requirement is potentially nontrivial e.g., the word problem for groups is known to be undecidable and is subsumed by the problem for 2-categories.

a PER model, soundness through a cross-language logical relation, and idempotence through a final inductive argument. The first two properties in particular are time-consuming to verify; recent work by Gratzer et al. [GSB19a] extended NbE to a type theory with an idempotent comonad but even in this minimal case the correctness proof occupied a 90 page technical report [GSB19b].

These difficulties are not unique to modal type theories, and a long line of research focuses on taming the complexity of NbE through *gluing* [AHS95; AK16; Coq19; Fio02; Ste21; Str98]. This line of work recasts normalization algorithms as the construction of models of type theory in categories defined by Artin gluing.

8.1.2 Normalization-by-gluing

Stepping back from type theory and normalization, fix a functor $F : \mathcal{C} \longrightarrow \mathcal{D}$ between a pair of categories. The *gluing* of F (written $\mathbf{Gl}(F)$) is a category whose objects triples $\big(C : \mathcal{C}, D : \mathcal{D}, f : D \longrightarrow F(D)\big)$. Morphisms in this category are given by pairs of morphisms (x_0, x_1) fitting into a commuting square, e.g.:

$$\begin{tikzcd}
D_0 \arrow[r, "x_1"] \arrow[d, "f_0"'] & D_1 \arrow[d, "f_1"] \\
F(C_0) \arrow[r, "F(x_0)"'] & F(C_1)
\end{tikzcd}$$

We note that there are evident projection functors $\pi_0 : \mathbf{Gl}(F) \longrightarrow \mathcal{C}$ and $\pi_1 : \mathbf{Gl}(F) \longrightarrow \mathcal{D}$.

We will view $\mathbf{Gl}(F)$ as a category of proof-relevant predicates on $\mathcal{C}$. To illustrate this, consider $\mathcal{E} = \mathbf{Gl}(\Gamma)$ where $\Gamma = \hom(\mathbf{1}, -) : \mathcal{C} \longrightarrow \mathbf{Set}$ is the global sections map on a cartesian closed category $\mathcal{C}$ sending each object to the set of its global points. Objects in $\mathcal{E}$ then correspond to an object $C : \mathcal{C}$ equipped with a map of sets $\pi : X \longrightarrow \hom(\mathbf{1}, C)$. Shifting perspective, we can view π as a (proof-relevant) predicate on the global points of C by setting $\Phi(c) = \pi^{-1}(c)$.

Remarkably, $\mathcal{E}$ inherits much of the structure of $\mathcal{C}$ so that $\mathcal{E}$ is also a Cartesian closed category and π_0 preserves finite products and exponentials. This is a recurrent pattern with Artin gluing; if $F : \mathcal{C} \longrightarrow \mathcal{D}$ is a nice functor between categories closed under (co)limits, exponentials, etc., then $\mathbf{Gl}(F)$ will be closed under the same operations in such a way that π_0 preserves them. In fact, unfolding the construction of e.g. binary products and exponentials in $\mathcal{E}$ yields, we the definition familiar from logical relations.

Example 8.1.1. *Viewing objects of $\mathcal{E}$ as proof-relevant predicates as described above, the exponential $(C, \Phi)^{(D,\Psi)}$ is given by the following pair (C^D, Ξ) where Ξ is defined as follows (writing ϵ for the evaluation map associated with C^D):*

$$\Xi(f) = \textstyle\prod_{d \in \hom(\mathbf{1}, D)} \Psi(d) \to \Phi(\epsilon\langle f, d\rangle)$$

Informally, therefore, we view $\mathbf{Gl}(F : \mathcal{C} \longrightarrow \mathcal{D})$ as the category of $\mathcal{D}$-valued predicated on $\mathcal{C}$ and the construction of exponentials, products, etc. within $\mathbf{Gl}(F)$ corresponds to defining a logical relation on $\mathcal{C}$. See Mitchell and Scedrov [MS93] for an exposition on this perspective.

Carrying out a normalization-by-gluing proof, therefore, turns the classical approach on its head. Originally one defined the normalization algorithm then showed it to be sound, complete, and idempotent. When carrying out the proof by gluing, the algorithm is not defined up front. Instead, one carefully one constructs a gluing category $\mathbf{Gl}(F)$ built on a functor out of the category of contexts of the initial model $\mathcal{I}$. Concretely, this is the category of syntactic contexts and simultaneous substitutions between them up to definitional equality. The heart of the argument then breaks down into three steps:

1. We show that $\mathbf{Gl}(F)$ supports a particular model of type theory $\mathcal{G}$.
2. We define a *reify* operation which sends terms from $\mathcal{G}$ to normal forms.
3. We show that the projection π_0 induces a morphism of models $\mathcal{G} \longrightarrow \mathcal{I}$ and that for a given term x in $\mathcal{G}$ reifying x yields a normal form for $\pi_0(x)$.

In particular, types in $\mathcal{G}$ will be chosen such that they consist of a type from the initial model along with a proof-relevant predicate carving out those terms which have (suitably hereditary) normal forms. A term in this model is then a term from the syntactic model together with a witness for the proof-relevant predicate associated with the type.

The first step and the universal property of the initial model produces a morphism of models $i : \mathcal{I} \longrightarrow \mathcal{G}$ and the second step ensures that $\pi_0 \circ i = \mathsf{id}$. Remarkably, this already defines a sound and complete normalization algorithm. The algorithm simply takes a syntactic term $M : A$, regards it as an element of the initial model, and then reifies $i(M)$ to obtain the normal form. Moreover, because $\pi_0 \circ i = \mathsf{id}$ we conclude that this yields a normal form for the supplied M.

To a coarse approximation, the construction of $\mathcal{G}$ and reification specifies the normalization algorithm and proves its soundness in a single step. The attentive reader will notice, however, that the completeness requirement from Section 8.1.1 seems to be absent from this new story. In fact, in this approach completeness is automatic and no proof is required. Indeed, terms and types within the initial model are realized by equivalences classes of syntactic terms and types taken up to definitional equality. Accordingly, the morphism i—and therefore the normalization algorithm—cannot distinguish between definitional equal terms.

One might suspect that working with equivalence classes of terms when defining $\mathcal{G}$ simply causes the burden to shift so that—while there is no need to prove completeness separately—the work of such a proof is spread throughout the construction of $\mathcal{G}$. In fact the opposite is the case: working with terms up to definitional equality substantially simplifies the construction of $\mathcal{G}$. Connectives in type theory only have universal properties up to definitional equality. Only when working with equivalences classes therefore, can we use these universal properties and benefit from existing results. For instance, we shall see that our construction of dependent products in our gluing model is essentially mechanical.

The gluing approach yields other unexpected advantages. Recall that $\mathbf{Gl}(F)$ intuitively consists of *proof-relevant* predicates. This proof relevance is crucial to an elegant treatement of universes in the model [Coq19]. We are able to define the predicate associated with an element of a universe to consist not only of an appropriate normal form but to also contain the data of the type it encodes within the model. In proof-irrelevant settings, universes were a frequent source of difficulty which necessitated laborious techniques to encode [All87].

8.1.3 Synthetic Tait computability

Using gluing to prove normalization is certainly an improvement over 'free-hand' proofs of normalization-by-evaluation, but the picture is not as rosy at may first appear. Models of type theory are subject to a variety of strict equations (see Item 3) which often force external constructions, where naturality obligations can be prohibitive. Worse, the passage between between mathematics internal to the gluing category and external constructions is difficult and the boundary frequently raises mismatches.

We follow Sterling and Harper [SH21] and adopt a synthetic approach to gluing. We have already discussed synthetic Tait computability in Chapter 4 in the context of *canonicity* but the same principles apply for proving normalization. In particular, Sterling and collaborators have then shown that it is possible to work exclusively within the internal language of $\mathbf{Gl}(F)$ to construct the normalization model just as was done with the canonicity model.

Unlike with canonicity, the functor F is more complex; it can no longer be chosen to be just the global sections functor but the resulting category is still a presheaf topos and still equipped with the expected model of extensional type theory and lex idempotent monads.

Just as with canonicity, the heart of the normalization proof is realized by a series of programming exercises in extensional type theory. The crucial distinction separating a proof of normalization from that of canonicity is in the structure of the semantic types. In Chapter 4, semantic types were given by a syntactic type together with a particular predicate upon them. This is insufficient for normalization, we must account for normal forms by equipping each semantic type with a reify operation sending an element to a corresponding normal form. Just as with classical gluing arguments, this in turn necessitates an operation promoting a neutral form into an element of the semantic type.

Synthetic Tait computability for MTT

Unlike Martin-Löf type theory or cubical type theory, a model of MTT is not a single category equipped with additional structure. Rather, a model is a network of categories, each supporting their own individual model of type theory which are then connected by various adjoints and natural transformations. The internal language of any of these categories is insufficient to construct the gluing model, so it is necessary to generalize from working in the extensional type theory of a topos to working in all topoi simultaneously using extensional MTT. Each topos then comes equipped with the structure of STC: a pair of lex monads and a strictification axiom. We prove that this mode-local structure is respected by the MTT modalities between topoi and call the resulting language *multimodal synthetic Tait computability*. The smooth interaction between MTT modalities and the lex monads $\bigcirc$ and $\bullet$ ensures that the key techniques of STC proofs can be generalized to multimodal STC.

With this machinery, we are able to give a concise and conceptual construction of the gluing model and extract the first normalization algorithm for multimodal type theory. In practice, this internal proof is necessary; removing the simplifying assumption on substitutions used in the canonicity proof given by Gratzer et al. [Gra+21] is already nearly intractable.

8.1.4 Related work

We have built on top of a long line of research systematically structuring logical relations as gluing models [AHS95; Coq19; Fio02; KHS19; MS93; Shu15b; Ste21; SA21; Str98]. In particular, Altenkirch et al. [AHS95] and Fiore [Fio02] recast NbE into the construction of a gluing model. In the case of Altenkirch et al. [AHS95], semantic types are realized as triples $(A, \downarrow, \uparrow)$. Generalizing from this work to dependent type theory has proven a considerable challenge [AK16]. The final ingredient for Martin-Löf type theory was provided by Coquand [Coq19]: a construction of a universe in this gluing model similar to that of Shulman [Shu15b].

Gluing for modal type theory Gratzer et al. [GSB19a] gave a classical normalization-by-evaluation proof for a Fitch-style type theory. The complexity of this proof, however, makes it intractable to extend to a general modal type theory like MTT. Unfortunately, extending gluing techniques to modal type theories has proven challenging. In particular, Gratzer et al. [Gra+20a] used gluing to prove canonicity for MTT, but they were forced to add an additional equality to MTT ($\mathbf{1}.\{\mu\} = \mathbf{1}$) to tame the construction of the gluing model. The challenge lies in fitting the glued category of contexts into a CwF-style model of type theory; the natural definition of glued types and terms fails to admit modalities. While there have been some attempts to systematize the construction of glued CwFs [KHS19], they do not apply to MTT.

Recently, Hu and Pientka [HP23] gave a proof of normalization for a simply-typed Fitch-style type theory (Kripke-style in their parlance) with one modality. They give two separate proofs of normalization; one through both an untyped PER model similar to Gratzer et al. [GSB19a] and one using a gluing model. Their gluing proof is closely related to the argument above. For instance, their theory of unified substitutions and modal transformations corresponds to a specialization of MTT's substitution calculus to one modality and, accordingly, their category of renamings offers a strict presentation of the category of renamings described above. Their proof, however, is done using external constructions on the gluing category which may make it difficult to scale to either multiple modalities or dependent types.

Synthetic Tait computability The introduction of representable map categories [Uem19] and LCCCs [GS20] for modeling the syntax of (non-modal) type theory offered an alternative approach. Crucially, they show that syntax can be given a universal property among structured categories with better behavior than CwFs. Sterling and collaborators [Ste21; SA21; SH21] have built on this idea and introduced synthetic Tait computability to prove syntactic metatheorems via gluing together LCCCs rather than CwFs. Unlike other approaches to gluing, STC generalizes well to a multimodal setting and by extending STC to MSTC normalization for MTT becomes tractable.

MTT as a metalanguage In a parallel line of work, Bocquet et al. [BKS21] have also used MTT as a metalanguage in the construction of models of type theory. They, however, do not work with a modal object type theory and instead use MTT to internalize a functor F rather than working internally to $\mathbf{Gl}(F)$. As a result, while both proofs use MTT modalities, the modalities used by op. cit. are encoded in our proof by fibered lex monads ($\circ$, $\bullet$) which prove easier to manipulate.

8.1.5 Structure

In Section 8.3, we discuss the models of MTT and relax the definition of a model of MTT to obtain *MTT cosmoi*. We prove that the syntactic cosmos enjoys a privileged position among MTT cosmoi (Theorem 8.3.5). Section 8.4 introduces *multimodal synthetic Tait computability* and shows that gluing together a network of topoi results in a model of extensional MTT equipped with STC structure in each mode (Theorem 8.4.15). Finally, in Section 8.5 we construct the normalization cosmos (Theorem 8.5.11) and extract the normalization function in Section 8.6 (Theorem 8.6.4). Section 8.7 discusses an extension of this proof to support crisp induction.

8.2 Normal and neutral forms in MTT

As mentioned in Section 8.1.1, the starting point for normalization is the definition of normal form. In MTT—as in other type theories—normal forms are presented together with a class of neutral forms. Intuitively, normal forms capture terms in β-normal and η-long form while neutrals are chains of eliminations applied to a variable.

We define normal and neutral forms as separate syntactic classes, equipped with their own family of typing judgments and decoding functions sending them to terms. Dependency complicates this definition as various typing rules require substitution in the types of premises or the conclusion. Unfortunately, it is just as hard to define substitution on normal forms as it is to define normalization in general [Wat+04]. Accordingly, a normal form (resp. neutral, normal type) is typed by the judgment $\Gamma \vdash^{\mathsf{nf}} u : A \mathbin{@} m$ (resp. $\Gamma \vdash^{\mathsf{ne}} e : A \mathbin{@} m$, $\Gamma \vdash^{\mathsf{nf}} \tau \mathbin{@} m$) where A is not required to be any sort of normal form. Furthermore, these judgments are defined inductive-recursively with decoding functions $|u|$ (resp. $|e|$, $|\tau|$) which send a normal form (resp. neutral, normal type) to its corresponding piece of syntax. Normal and neutral forms for mode-local connectives are unchanged from their standard presentation in type theory:

$$\begin{array}{llll} \textit{(Normals)} & u & ::= & \lambda(u) \mid \mathsf{up}(e) \mid \mathsf{mod}_{\mu}(u) \mid \dots \\ \textit{(Neutral)} & e & ::= & \mathbf{v}^{\alpha}_{k} \mid e(u) \mid \mathsf{letmod}(\mu; \nu; \tau; e; u) \mid \dots \\ \textit{(Normal types)} & \tau & ::= & \tau \to \sigma \mid \langle \mu \mid \tau \rangle \mid \mathsf{El}(u) \mid \dots \end{array}$$

We defer a more complete presentation of the judgments and decoding function to Fig. 8.2, but remark that the neutral form for variables is annotated with a 2-cell and index, decoding to $\mathbf{v}$ together with a combination of weakening and 2-cell substitutions $\mathbf{p}$ and $\{\alpha\}$.

To ensure that normal forms are η-long, neutrals can only be 'injected' into normals by $\mathsf{up}(-)$ for types without an η law e.g., at modal types but not at dependent products. Finally, we emphasize that normal forms are freely generated, so their equality is decidable if and only if equality of modalities and 2-cells is decidable.

Renamings While normal and neutral forms are not stable under substitution, they are stable under the restricted class of *renamings*. The formal definition of renamings

$$\frac{}{\Gamma \vdash {!} : \mathbf{1} \mathbin{@} m \qquad |{!}| = {!}} \qquad \frac{}{\Gamma.(\mu \mid A) \vdash {\uparrow} : \Gamma \mathbin{@} m \qquad |{\uparrow}| = \mathbf{p}}$$

$$\frac{}{\Gamma \vdash \mathsf{id} : \Gamma \mathbin{@} m \qquad |\mathsf{id}| = \mathsf{id}} \qquad \frac{\Gamma_0 \vdash r : \Gamma_1 \mathbin{@} m \qquad \Gamma_1 \vdash s : \Gamma_2 \mathbin{@} m}{\Gamma_0 \vdash s \circ r : \Gamma_2 \mathbin{@} m \qquad |s \circ r| = |s| \circ |r|}$$

$$\frac{\Gamma \vdash r : \Delta \mathbin{@} m}{\Gamma.\{\mu\} \vdash r.\{\mu\} : \Delta.\{\mu\} \mathbin{@} n \qquad |r.\{\mu\}| = |r|.\{\mu\}}$$

$$\frac{\mu, \nu : n \longrightarrow m \qquad \alpha : \nu \longrightarrow \mu}{\Gamma.\{\mu\} \vdash \{\alpha\}_\Gamma : \Gamma.\{\nu\} \mathbin{@} n \qquad |\{\alpha\}_\Gamma| = \Gamma.\{\alpha\}}$$

$$\frac{\Gamma \vdash r : \Delta \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash^{\mathsf{ne}} \mathbf{v}_k^\alpha : A[|r|.\{\mu\}] \mathbin{@} n}{\Gamma \vdash r.\mathbf{v}_k^\alpha : \Delta.(\mu \mid A) \mathbin{@} m \qquad |r.\mathbf{v}_k^\alpha| = |r|.|\mathbf{v}_k^\alpha|}$$

Figure 8.1: Complete definition of renamings

is presented in Fig. 8.1. Intuitively, they are the smallest class of substitutions closed under weakening, composition, identity, modal substitutions ($-.\{\mu\}$,$\{\alpha\}$), and extension by variables $\mathbf{v}_k^\alpha$.

Renamings are easily seen to act on normal forms, neutral forms, and normal types. Unlike normals and neutrals, however, renamings are taken up to a definitional equality which ensures that e.g., composition is associative and that modal substitutions organize into a 2-functor. This poses no issue as the action of renamings on normals and neutrals send definitionally equal renamings to identical normals and neutrals, ensuring that the action lifts to equivalences classes.

A nontrivial definitional equality on renamings is essential, however, as it ensures that the class of contexts of mode m and renamings between them organizes into a category Ren_m and that the assignments $m \mapsto \mathsf{Ren}_m$, $\mu \mapsto -.\{\mu\}$, and $\alpha \mapsto \{\alpha\}$ define a 2-functor $\mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$.

Lemma 8.2.1. *The decoding of renamings to substitutions gives a 2-natural transformation* $\mathbf{i}[-] : \mathsf{Ren}_{-} \longrightarrow \mathsf{Cx}_{-}$*.*

8.3 Models and cosmoi

Gratzer et al. [Gra+21] introduced MTT as a generalized algebraic theory so that MTT is automatically equipped with a category of models. A standard result of GATs ensures that that the syntax of MTT organizes into an initial model which opens the possibility of semantic methods for proving results about syntax. Gratzer et al. [Gra+21] then repackages the definition of models in the language of natural models [Awo18].

8.3.1 MTT cosmoi

As mentioned in Section 8.1, normalization is proven through the construction of a model of MTT together with a map from this model to syntax. Models of MTT and

$$\frac{}{\Gamma \vdash^{\mathsf{nf}} \mathsf{bool} \mathrel{@} m \qquad \Gamma \vdash^{\mathsf{nf}} \mathsf{U} \mathrel{@} m} \qquad \frac{\Gamma \vdash^{\mathsf{nf}} \tau \mathrel{@} m \qquad \Gamma.(\mathsf{id} \mid A) \vdash^{\mathsf{nf}} \sigma \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \tau \to \sigma \mathrel{@} m}$$

$$\frac{\Gamma \vdash^{\mathsf{nf}} \tau \mathrel{@} m \qquad \Gamma.(\mathsf{id} \mid A) \vdash^{\mathsf{nf}} \sigma \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \tau \times \sigma \mathrel{@} m} \qquad \frac{\Gamma \vdash^{\mathsf{nf}} \tau \mathrel{@} m \qquad \Gamma \vdash^{\mathsf{nf}} u, v : A \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{Id}_A(u, v) \mathrel{@} m}$$

$$\frac{\Gamma.\{\mu\} \vdash^{\mathsf{nf}} \tau \mathrel{@} n}{\Gamma \vdash^{\mathsf{nf}} \langle \mu \mid \tau \rangle \mathrel{@} m} \qquad \frac{\Gamma \vdash^{\mathsf{nf}} u : \mathsf{U} \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{El}(u) \mathrel{@} m}$$

$$\frac{\Gamma(k) = (\mu \mid A) \qquad \mathsf{mods}(\Gamma, k) = \nu \qquad \alpha : \mu \longrightarrow \nu}{\Gamma \vdash^{\mathsf{ne}} \mathbf{v}_k^{\alpha} : A[\{\alpha\} \circ (\mathbf{p}.\{\nu_{k-1}\}) \cdots \circ (\mathbf{p}.\{\nu_0\})] \mathrel{@} m}$$

$$\frac{}{\Gamma \vdash^{\mathsf{nf}} \mathsf{tt} : \mathsf{Bool} \mathrel{@} m \qquad \Gamma \vdash^{\mathsf{nf}} \mathsf{ff} : \mathsf{Bool} \mathrel{@} m} \qquad \frac{\Gamma \vdash^{\mathsf{ne}} e : \mathsf{Bool} \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{up}(e) : \mathsf{Bool} \mathrel{@} m}$$

$$\frac{\begin{array}{c}\Gamma.(\mathsf{id}_m \mid \mathsf{Bool}) \vdash^{\mathsf{nf}} \tau \mathrel{@} m \\ \Gamma \vdash^{\mathsf{ne}} e : \mathsf{Bool} \mathrel{@} m \qquad \Gamma \vdash^{\mathsf{nf}} v_1 : A[\mathsf{id}.\mathsf{tt}] \mathrel{@} m \qquad \Gamma \vdash^{\mathsf{nf}} v_2 : A[\mathsf{id}.\mathsf{ff}] \mathrel{@} m\end{array}}{\Gamma \vdash^{\mathsf{ne}} \mathsf{if}(\tau; e; v_1; v_2) : A[\mathsf{id}.|e|] \mathrel{@} m}$$

$$\frac{\Gamma \vdash^{\mathsf{nf}} u : A \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{refl}(u) : \mathsf{Id}(A, M, M) \mathrel{@} m} \qquad \frac{\Gamma \vdash M_0, M_1 : A \qquad \Gamma \vdash^{\mathsf{ne}} e : \mathsf{Id}(A, M_0, M_1) \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{up}(e) : \mathsf{Id}(A, M_0, M_1) \mathrel{@} m}$$

$$\frac{\begin{array}{c}\Gamma \vdash M_0, M_1 : A \\ \Gamma \vdash^{\mathsf{ne}} e : \mathsf{Id}(A, M_0, M_1) \mathrel{@} m \qquad \Gamma.(\mathsf{id}_m \mid A).(\mathsf{id}_m \mid A).(\mathsf{id}_m \mid \mathsf{Id}(A[\mathbf{p}^2], \mathbf{v}[\mathbf{p}], \mathbf{v})) \vdash^{\mathsf{nf}} \tau \mathrel{@} m \\ \Gamma.(\mathsf{id} \mid A) \vdash^{\mathsf{nf}} u : C[\mathsf{id}.\mathbf{v}.\mathbf{v}.\mathsf{refl}\mathbf{v}] \mathrel{@} m\end{array}}{\Gamma \vdash^{\mathsf{ne}} \mathsf{J}(\tau; u; e) : C[\mathsf{id}.M_0.M_1.P] \mathrel{@} m}$$

$$\frac{\Gamma.(\mu \mid A) \vdash^{\mathsf{nf}} u : B \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \lambda(u) : (\mu \mid A) \to B \mathrel{@} m} \qquad \frac{\Gamma \vdash^{\mathsf{ne}} e : (\mu \mid A) \to B \mathrel{@} m \qquad \Gamma \vdash^{\mathsf{nf}} u : A \mathrel{@} m}{\Gamma \vdash^{\mathsf{ne}} e(u) : B[\mathsf{id}.|u|] \mathrel{@} m}$$

$$\frac{\Gamma.\{\mu\} \vdash^{\mathsf{nf}} u : A \mathrel{@} n}{\Gamma \vdash^{\mathsf{nf}} \mathsf{mod}_{\mu}(u) : \langle \mu \mid A \rangle \mathrel{@} m} \qquad \frac{\Gamma \vdash^{\mathsf{ne}} e : \langle \mu \mid A \rangle \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{up}(e) : \langle \mu \mid A \rangle \mathrel{@} m}$$

$$\frac{\begin{array}{c}\Gamma.\{\mu\} \vdash^{\mathsf{ne}} u : \langle \nu \mid A \rangle \mathrel{@} n \\ \Gamma.(\mu \mid \langle \nu \mid A \rangle) \vdash^{\mathsf{nf}} \tau \mathrel{@} m \qquad \Gamma.(\mu \circ \nu \mid A) \vdash^{\mathsf{nf}} u : B[\mathbf{p}.\mathsf{mod}_{\nu}(\mathbf{v})] \mathrel{@} m\end{array}}{\Gamma \vdash^{\mathsf{ne}} \mathsf{letmod}(\mu; \nu; \tau; e; u) : B[\mathsf{id}.|e|] \mathrel{@} m}$$

$$\frac{\Gamma \vdash^{\mathsf{ne}} e : \mathcal{U} \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{up}(e) : \mathcal{U} \mathrel{@} m} \qquad \frac{\Gamma.\{\mu\} \vdash^{\mathsf{nf}} u : \mathcal{U} \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \widehat{\langle \mu \mid u \rangle} : \mathcal{U} \mathrel{@} m} \qquad \frac{\Gamma \vdash^{\mathsf{ne}} e : \mathcal{U} \mathrel{@} m \qquad \Gamma \vdash^{\mathsf{ne}} f : \mathsf{El}(|e|) \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{up}(f) : \mathsf{El}(|e|) \mathrel{@} m}$$

$$\frac{\Gamma.\{\mu\} \vdash A : \mathcal{U} \mathrel{@} n \qquad \Gamma \vdash^{\mathsf{ne}} e : \mathsf{El}(\mathsf{Mod}_{\mu}(A)) \mathrel{@} m}{\Gamma \vdash^{\mathsf{ne}} \mathsf{dec}^{\rhd}(e) : \langle \mu \mid \mathsf{El}(A) \rangle \mathrel{@} m} \qquad \frac{\Gamma \vdash^{\mathsf{nf}} u : \langle \mu \mid \mathsf{El}(A) \rangle \mathrel{@} m}{\Gamma \vdash^{\mathsf{nf}} \mathsf{dec}^{\lhd}(u) : \mathsf{El}(\mathsf{Mod}_{\mu}(A)) \mathrel{@} m}$$

Figure 8.2: Definition of selected normals, neutrals, and normal types

morphisms between them are difficult to construct, however, because of the extreme strictness of morphisms and the requirement that each τ_m be a representable natural transformation. Prior to normalization, therefore, we introduce a weakened notion of model: an MTT cosmos. An MTT cosmos is an axiomatization of a natural model of MTT, but rather than working in presheaf topoi and requiring that τ_m is a representable natural transformation a cosmos requires only that τ_m be a morphism in a locally cartesian closed category equipped with structure such as Components 7.1.6 and 7.1.7.

Definition 8.3.1. A *cosmos* is a pseudofunctor $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ such that each $F(m)$ is a locally cartesian closed category and each $F(\mu)$ has a left adjoint $F_!(\mu) \dashv F(\mu)$.

Example 8.3.2. *A model of MTT F assembles into a cosmos G by taking $G(m) = \mathbf{PSh}(F(m))$ and $G(\mu) = F(\mu)^*$. In particular, we write $\mathcal{S} : \mathcal{M} \longrightarrow \mathbf{Cat}$ for the cosmos induced by the initial model of MTT specified by Theorem 7.1.8.*

The additional requirements imposed by natural models of MTT to encode various connectives can be transferred *mutatis mutandis* to a cosmos; they are all stated within the language of locally cartesian closed categories.

Definition 8.3.3. An cosmos F is an MTT cosmos when equipped with the following structure:

1. In $F(m)$, there is a universe $\tau_m : \mathcal{T}_m \longrightarrow \mathcal{T}_m$ with a choice of codes witnessing its closure under dependent sums and products, identity types, and booleans. For instance, a choice of pullback square of the following shape:

$$\begin{tikzcd}
\sum_{A:F(\mu)(\mathcal{T}_m)} \sum_{B:F(\mu)(\tau_n[A])\to\mathcal{T}_m} \prod_{a:F(\mu)(\tau_n[A])} \tau_m[B(a)] \arrow[r, "\mathbf{lam}"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{T}_m \arrow[d] \\
\sum_{A:F(\mu)(\mathcal{T}_n)} F(\mu)(\tau_n[A]) \to \mathcal{T}_m \arrow[r, "\mathbf{Prod}"'] & \mathcal{T}_m
\end{tikzcd}$$

2. For each μ, there exists a chosen commuting square

$$\begin{tikzcd}
F(\mu)(\mathcal{T}_n) \arrow[r] \arrow[d] & \mathcal{T}_m \arrow[d] \\
F(\mu)(\mathcal{T}_n) \arrow[r, "\mathbf{Mod}"'] & \mathcal{T}_m
\end{tikzcd} \tag{8.1}$$

3. For each $\mu : n \longrightarrow m$ and $\nu : o \longrightarrow n$, there is a chosen lifting structure $F(\mu)(m) \pitchfork F(\mu \circ \nu)(\mathcal{T}_o) \times \tau_m$, where $m : F(\nu)(\mathcal{T}_o) \longrightarrow F(\nu)(\mathcal{T}_o) \times_{\mathcal{T}_n} \mathcal{T}_n$ is the comparison map induced by Eq. (8.1).

4. τ_m contains a subuniverse also closed under all these connectives.

Definition 8.3.4. A morphism between MTT cosmoi $\alpha : F \longrightarrow G$ is a 2-natural transformation α such that α_m is an LCCC functor and preserves all connectives strictly.

Furthermore, we require that α satisfies the Beck-Chevalley condition so that there is a natural isomorphism $\beta_\mu : \alpha_n \circ F(\mu)_! \cong G(\mu)_! \circ \alpha_m$ commuting with transposition. Precisely, if $a : X \longrightarrow F(\mu)(Y) : F(m)$ the transposition of $\alpha_\mu \circ \alpha_m(a)$ is $\alpha_n(\widehat{a}) \circ \beta_\mu^{-1}$.

A morphism of MTT cosmoi is both more and less restrictive than a morphism of MTT models. While a morphism of models need not induce an LCC functor between the relevant presheaf categories, a morphism of cosmoi is not required to strictly preserve context extension or the choice of terminal context. It so happens that the only map of consequence in this proof is locally cartesian closed, so the additional structure of morphisms of cosmoi poses no issue. Not requiring the strict preservation of context extension and dropping the representability requirements from MTT cosmoi, however, ensures that cosmoi are far easier to construct.

Merely defining a normalization cosmos $\mathcal{G}$ and projection $\pi : \mathcal{G} \longrightarrow \mathcal{S}$, however, is not enough to prove normalization; we also need a section to π. In the category of models, this section would exist as a consequence of initiality, but $\mathcal{S}$ is not initial in the category of MTT cosmoi.[2] Accordingly, we cannot easily obtain a section of a map into $\mathcal{S}$ and in fact sections rarely exist. Any such map, however, is surjective on definable terms and this 'quasi-projectivity' is sufficient:

Theorem 8.3.5. *Fix an MTT cosmos G and $\pi : G \longrightarrow \mathcal{S}$.*

1. *For $\vdash \Gamma\ \mathsf{cx} \mathbin{@} m$, there exists $[\![\Gamma]\!] : G(m)$ and a canonical isomorphism $\alpha_\Gamma : \pi([\![\Gamma]\!]) \cong \mathbf{y}(\Gamma)$.*
2. *For every $\Gamma \vdash A\ \mathsf{type} \mathbin{@} m$, there exists $[\![A]\!] : [\![\Gamma]\!] \longrightarrow \mathcal{T}_m$ such that $\pi([\![A]\!]) \circ \alpha_\Gamma = \lfloor A \rfloor$.*
3. *For every $\Gamma \vdash M : A \mathbin{@} m$, there exists $[\![M]\!] : [\![\Gamma]\!] \longrightarrow \mathcal{T}_m$ lying over $[\![A]\!]$ such that $\pi([\![M]\!]) \circ \alpha_\Gamma = \lfloor M \rfloor$.*

Here $\lfloor - \rfloor$ is the isomorphism induced by the Yoneda lemma.

Remark 8.3.6. While we have proven this result at quite generally, we will apply it only in the special case where π is a 2-natural transformation between strict 2-functors and required isomorphisms of left adjoints are likewise identities. The reader may accordingly safely ignore these coherences when reading the proof without consequence. ◇

Remark 8.3.7. Both Theorems 7.1.8 and 8.3.5 are categorical abstractions of *rule induction*. Indeed, Theorem 7.1.8 is used to prove Theorem 8.3.5—via the construction of an appropriate displayed model [KKA19]—and the latter takes the place of rule induction in the proof of normalization (see Theorem 8.6.4). ◇

Proof. We write $\mathbf{El}_m$, $\mathbf{Ty}_m$ and $\mathbf{Tm}_m$ instead of τ_m, $\mathcal{T}_m$, and $\mathcal{T}_m$ in the syntactic model, reserving the latter exclusively for G. We write $[\![\mu]\!]$ for the functor sending Γ to $\Gamma.\{\mu\}$. We begin by replacing G by an equivalent strict 2-functor so that π becomes strictly 2-natural.

[2] 2-monad theory [GS20; KPT99] yields an initial cosmos $\mathcal{I}$ but we work with $\mathcal{S}$ because—unlike $\mathcal{I}$—it is known to adequately represent syntax.

We construct a displayed model of MTT [KKA19] which lies over the syntactic model. Using the existing coherence result for MTT [Gra+20b], we only ensure that $\Gamma.\{\mu\}.\{\nu\}$ and $\Gamma.\{\mu \circ \nu\}$ agree up to pseudonatural isomorphism.

- A context in m is a triple $X : G(m)$, $\vdash \Gamma\,\mathsf{cx} \mathbin{@} m$, and $\alpha : \pi(X) \cong \mathbf{y}(\Gamma)$.
- A type in a context (X, Γ, α) is a pair of $\bar{A} : X \longrightarrow \mathcal{T}_m$ and $\Gamma \vdash A\,\mathsf{type} \mathbin{@} m$ such that $\pi(\bar{A}) = \lfloor A \rfloor \circ \alpha$.
- A term in a context (X, Γ, α) of type (A^*, A) is a pair $M^* : X \longrightarrow \tau_m[A^*]$ and $\Gamma \vdash M : A \mathbin{@} m$ such that $\pi(M^*) = \lfloor M \rfloor \circ \alpha$.
- A substitution $(X, \Gamma, \alpha) \longrightarrow (Y, \Delta, \beta)$ is a pair $f : X \longrightarrow Y$ and $\Gamma \vdash \delta : \Delta \mathbin{@} m$ satisfying $\beta \circ \pi(f) = \mathbf{y}(\delta) \circ \alpha$

Once this model is constructed, the result is follows from Theorem 7.1.8. The construction of contexts, substitutions, terms, and types is straightforward as π is a 2-natural transformation which preserves finite limits, and commutes with all connectives. We show two cases.

The action of a modality on a context Given a triple (X, Γ, α) at mode n and a modality $\mu : n \longrightarrow m$, we define the 'locked' context to be the following:

$$(G(\mu)_!(X), \Gamma.\{\mu\}, \gamma \circ [\![\mu]\!]_! \alpha \circ \beta)$$

Here $\beta : \pi(G(\mu)_! X) \cong [\![\mu]\!]_! \pi(X)$ and $\gamma : [\![\mu]\!]_! \mathbf{y}(\Gamma) \cong \mathbf{y}(\Gamma.\{\mu\})$ are the canonical isomorphisms.

Modal types Suppose we are given a context (X, Γ, α) and a type (A^*, A) in the context $(G(\mu)_!(\mu)(X), \Gamma.\{\mu\}, \gamma \circ [\![\mu]\!]^*(\alpha) \circ \beta_\mu)$. We form the modal type as

$$(\mathbf{Mod}_\mu(\widehat{A^*}), \langle \mu \mid A \rangle)$$

It remains to check that these types are coherent i.e.:

$$\pi(\mathbf{Mod}_\mu(\widehat{A^*})) = \lfloor \langle \mu \mid A \rangle \rfloor \circ \alpha$$

By assumption, $\pi(A^*) = \lfloor A \rfloor \circ \gamma \circ [\![\mu]\!]^*(\alpha) \circ \beta$. By our assumption that π satisfies Beck-Chevalley $\pi(\widehat{A^*}) = \widehat{\lfloor A \rfloor \circ} \gamma \circ \alpha$. The result follows from the fact that π preserves $\mathbf{Mod}_\mu$. □

8.3.2 *Presheaf cosmoi*

Example 8.3.2 shows that each model of MTT induces an MTT cosmos. In fact, such cosmoi are particularly well-behaved as they are comprised of presheaf topoi connected by adjoint triples. These cosmoi enjoy a privileged role in our proof and we observe some of their unique behavior.

Definition 8.3.8. A presheaf cosmos F is a cosmos where each $F(m)$ is a presheaf topos and each right adjoint $F(\mu)$ sends small families to small families.

What distinguishes presheaf cosmoi from other cosmoi is the rich internal language they offer. Gratzer et al. [Gra+21] have proven that such a cosmos F supports a model of *extensional* MTT with the same mode theory where $\langle \mu \mid - \rangle$ is interpreted by $F(\mu)$. We will now use extensional MTT as a *multimodal metalanguage* to specify the structure of an MTT cosmos as a sequence of constants, thereby reducing its construction to a series of programming exercises. It is this characterization of MTT-cosmoi that we will use in Section 8.5 to construct the normalization cosmos.

Remark 8.3.9. Some caution is required here, as a presheaf cosmos will frequently host more than one interpretation of MTT. A presheaf cosmos is always equipped with this modal metalanguage (extensional MTT) which can then be used to specify a model of (intensional) MTT. This is comparable to some of the diagrams used in e.g. Awodey [Awo18] or Chapter 3, where type theory is used to describe a model of type theory. ⋄

Within this internal language, the universe $\tau_m : \mathcal{T}_m \longrightarrow \mathcal{T}_m$ is encoded by a pair of types:

$$\mathsf{Ty}_m : \mathcal{U}_0 \qquad \mathsf{Tm}_m : (A : \mathsf{Ty}_m) \to \mathcal{U}_0$$

Each of the diagrams discussed in Section 8.3.1 can then be translated into constants within this language with the use of dependent types automatically encoding commutativity. For instance, Eq. (8.1) becomes the following pair of constants:

$$\mathsf{Mod}_\mu : (\mu \mid \mathsf{Ty}_n) \to \mathsf{Ty}_m \qquad \mathsf{m}_\mu : (A : (\mu \mid \mu))\mathsf{Ty}_n(\mu \mid \mathsf{Tm}_n(A)) \to \mathsf{Tm}_m(\mathsf{Mod}_\mu(A))$$

In this language it is far easier to specify the modal elimination principle:

$$\begin{aligned}
&\mathsf{letmod}_{\mu;\nu} : \\
&\quad (A : (\nu \circ \mu \mid \mathsf{Ty}_n))\,(B : (\mu \mid \mathsf{Tm}_n(\mathsf{Mod}_\mu(A))) \to \mathsf{Ty}_o) \\
&\quad \big(b : \big(x : \big(\nu \circ \mu \mid \mathsf{Tm}_n(A)\big)\big) \to \mathsf{Tm}_o\big(B(\mathsf{m}_\mu(A, x))\big)\big) \\
&\quad \to (a : (\nu \mid \mathsf{Tm}_m(\mathsf{Mod}_\mu(A)))) \to \mathsf{Tm}_o(B(a))
\end{aligned}$$

Each argument to $\mathsf{letmod}_{\mu;\nu}$ corresponds directly to a premise of the rule given in Section 6.2. The hypothetical judgment is encoded by the dependent products in the language and each occurrence of $-.\{-\}$ is replaced with an occurrence of the corresponding modal type within the metalanguage. The β-rule for this elimination principle is encoded by another constant inhabiting the equality type:

$$\begin{aligned}
&\mathsf{Mod/beta}_{\mu;\nu} : \\
&\quad (A : (\nu \circ \mu \mid \mathsf{Ty}_n))\,(B : (\mu \mid \mathsf{Tm}_n(\mathsf{Mod}_\mu(A))) \to \mathsf{Ty}_o) \\
&\quad \big(b : \big(x : \big(\nu \circ \mu \mid \mathsf{Tm}_n(A)\big)\big) \to \mathsf{Tm}_o\big(B(\mathsf{m}_\mu(A, x))\big)\big) \\
&\quad \to (a : (\nu \circ \mu \mid \mathsf{Tm}_m(A))) \to \mathsf{letmod}_{\mu;\nu}(A, B, b, \mathsf{m}_\mu(A, a)) = b(a)
\end{aligned}$$

The remaining connectives are detailed in Fig. 8.3.

8.4 Multimodal Synthetic Tait computability

In light of Section 8.3, we revise the proof outlined in Section 8.1: instead of constructing a glued *model* of MTT, we will construct a glued MTT *cosmos*. In fact, we will construct

$$\begin{aligned}
&\mathsf{Prod} : (A : (\mu \mid \mathsf{Ty}_m))\,(B : (\mu \mid \mathsf{Tm}_m(A)) \to \mathsf{Ty}_m) \to \mathsf{Ty}_m\\
&\alpha_{\mathsf{Prod}} : (A : (\mu \mid \mathsf{Ty}_m))\,(B : (\mu \mid \mathsf{Tm}_m(A)) \to \mathsf{Ty}_m)\\
&\quad \to \mathsf{Tm}_m(\mathsf{Prod}(A, B)) \cong [(a : (\mu \mid \mathsf{Tm}_m(A))) \to \mathsf{Tm}_m(B(a))]
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Sig} : (A : \mathsf{Ty}_m) \to \mathsf{Tm}_m(A) \to \mathsf{Ty}_m \to \mathsf{Ty}_m\\
&\alpha_{\mathsf{Sig}} : (A : \mathsf{Ty}_m)(B : \mathsf{Tm}_m(A) \to \mathsf{Ty}_m)\\
&\quad \to \mathsf{Tm}_m(\mathsf{Sig}(A, B)) \cong \left[\textstyle\sum_{a:\mathsf{Tm}_m(A)} \mathsf{Tm}_m(B(a))\right]
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Bool} : \mathsf{Ty}_m\\
&\mathsf{true}, \mathsf{false} : \mathsf{Tm}_m(\mathsf{Bool})\\
&\mathsf{if} : (A : \mathsf{Tm}_m(\mathsf{Bool}) \to \mathsf{Ty}_m)\\
&\quad \to \mathsf{Tm}_m(A(\mathsf{true})) \to \mathsf{Tm}_m(A(\mathsf{false})) \to (b : \mathsf{Tm}_m(\mathsf{Bool})) \to \mathsf{Tm}_m(A(b))\\
&\_ : (A : \mathsf{Tm}_m(\mathsf{Bool}) \to \mathsf{Ty}_m)\,(t : \mathsf{Tm}_m(A(\mathsf{true})))\,(f : \mathsf{Tm}_m(A(\mathsf{false})))\\
&\quad \to (\mathsf{if}(A, t, f, \mathsf{true}) = t) \times (\mathsf{if}(A, t, f, \mathsf{false}) = f)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Id} : (A : \mathsf{Ty}_m)(a_0, a_1 : \mathsf{Tm}_m(A)) \to \mathsf{Ty}_m\\
&\mathsf{refl} : (A : \mathsf{Ty}_m)(a : \mathsf{Tm}_m(A)) \to \mathsf{Tm}_m(\mathsf{Id}(A, a, a))\\
&\mathsf{J} : (A : \mathsf{Ty}_m)\,(B : (a_0, a_1 : \mathsf{Tm}_m(A))(p : \mathsf{Tm}_m(\mathsf{Id}(A, a_0, a_1))) \to \mathsf{Ty}_m)\\
&\quad \to ((a : \mathsf{Tm}_m(A)) \to \mathsf{Tm}_m(B(a, a, \mathsf{refl}(a))))\\
&\quad \to (a_0, a_1 : \mathsf{Tm}_m(A))(p : \mathsf{Tm}_m(\mathsf{Id}(A, a_0 a_1))) \to \mathsf{Tm}_m(B(a_0, a_1, p))\\
&\_ : (A : \mathsf{Ty}_m)\,(B : (a_0, a_1 : \mathsf{Tm}_m(A))(p : \mathsf{Tm}_m(\mathsf{Id}(A, a_0, a_1))) \to \mathsf{Ty}_m)\\
&\quad \to (b : (a : \mathsf{Tm}_m(A)) \to \mathsf{Tm}_m(B(a, a, \mathsf{refl}(a))))\\
&\quad \to (a : \mathsf{Tm}_m(A)) \to \mathsf{J}(A, B, b, a, a, \mathsf{refl}(a)) = b(a)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{Uni} : \mathsf{Ty}_m\\
&\mathsf{El} : \mathsf{Tm}_m(\mathsf{Uni}) \to \mathsf{Ty}_m\\
&\widehat{\mathsf{Sig}} : (A : \mathsf{Tm}_m(\mathsf{Uni})) \to (\mathsf{Tm}_m(\mathsf{El}(A)) \to \mathsf{Tm}_m(\mathsf{Uni})) \to \mathsf{Tm}_m(\mathsf{Uni})\\
&\widehat{\mathsf{Prod}} : (A : \mathsf{Tm}_m(\mathsf{Uni})) \to (\mathsf{Tm}_m(\mathsf{El}(A)) \to \mathsf{Ty}_m) \to \mathsf{Tm}_m(\mathsf{Uni})\\
&\widehat{\mathsf{Bool}} : \mathsf{Tm}_m(\mathsf{Uni})\\
&\widehat{\mathsf{Mod}} : (\mu \mid \mathsf{Tm}_n(\mathsf{Uni})) \to \mathsf{Tm}_m(\mathsf{Uni})\\
&\mathsf{dec}_{\widehat{\mathsf{Sig}}} : (A : \mathsf{Tm}_m(\mathsf{Uni}))(B : \mathsf{Tm}_m(\mathsf{El}(A)) \to \mathsf{Tm}_m(\mathsf{Uni}))\\
&\quad \to \mathsf{Tm}_m(\mathsf{El}(\widehat{\mathsf{Sig}}(A, B,))) \cong \mathsf{Tm}_m(\mathsf{Sig}(\mathsf{El}(A), \mathsf{El} \circ B))\\
&\mathsf{dec}_{\widehat{\mathsf{Prod}}} : (A : \mathsf{Tm}_m(\mathsf{Uni}))(B : \mathsf{Tm}_m(\mathsf{El}(A)) \to \mathsf{Tm}_m(\mathsf{Uni}))\\
&\quad \to \mathsf{Tm}_m(\mathsf{El}(\widehat{\mathsf{Prod}}(A, B))) \cong \mathsf{Tm}_m(\mathsf{Prod}(\mathsf{El}(A), \mathsf{El} \circ B))\\
&\mathsf{dec}_{\widehat{\mathsf{Bool}}} : \mathsf{Tm}_m(\mathsf{El}(\widehat{\mathsf{Bool}})) \cong \mathsf{Tm}_m(\mathsf{Bool})\\
&\mathsf{dec}_{\widehat{\mathsf{Mod}}} : (A : (\mu \mid \mathsf{Tm}_m(\mathsf{Uni}))) \to \mathsf{Tm}_m(\mathsf{El}(\widehat{\mathsf{Mod}}(A))) \cong \mathsf{Tm}_m(\mathsf{Mod}_\mu(\mathsf{El}(A)))
\end{aligned}$$

Figure 8.3: Internal presentation of an MTT cosmos

a glued presheaf cosmos, and take advantage of the internal language discussed in Section 8.3.2 to upgrade it to an MTT cosmos with a projection onto $\mathcal{S}$. Prior to this, however, we must show that (1) a pair of cosmoi can be glued together and (2) that each mode of the internal language of the resulting cosmos can be extended with synthetic Tait computability primitives compatible with the already-present MTT modalities.

8.4.1 Synthetic Tait computability

We briefly recall the language of synthetic Tait computability to fix notation. For this subsection, fix two presheaf topoi $\mathcal{E}$ and $\mathcal{F}$ along with a continuous functor $\rho : \mathcal{E} \longrightarrow \mathcal{F}$.

We will work with $\mathbf{Gl}(\rho)$. Intuitively $\mathbf{Gl}(\rho)$ is a category of proof-relevant $\mathcal{F}$-predicates on ρ-elements of $\mathcal{E}$. To cultivate this intuition, consider $\mathcal{F} = \mathbf{Set}$ and $\rho = \mathrm{hom}(\mathbf{1}, -)$. An object of $\mathbf{Gl}(\mathrm{hom}(\mathbf{1}, -))$ is a triple of (S, E, f) which induces a proof-relevant predicate $\Phi(e) = f^{-1}(e)$ on the global points of E. Following Tait [Tai67], we refer to elements in the image of f as *computable elements*. Morphisms are then morphisms of $\mathcal{E}$ equipped with additional structure ensuring that computable elements are sent to computable elements.

We now reap the first reward from considering proof-*relevant* predicates: $\mathbf{Gl}(\rho)$ is extremely well-behaved.

Theorem 8.4.1 (Artin et al. [AGV72] and Carboni and Johnstone [CJ95])**.** *$\mathbf{Gl}(\rho)$ is a presheaf topos and π_0 is a logical functor with left and right adjoints.*

As a presheaf topos, $\mathbf{Gl}(\rho)$ enjoys a model of extensional type theory with a strictly cumulative hierarchy of universes and a universe of propositions Ω. We can use this language to *synthetically* build logical relations models [SH21]. In order to effectively construct such models, however, we must supplement type theory with primitives specific to $\mathbf{Gl}(\rho)$. The most fundamental of these is a proposition:

Definition 8.4.2. The *syntactic proposition* $\mathbf{syn} : \Omega$ is interpreted in $\mathbf{Gl}(\rho)$ as the subterminal object $(\mathbf{1}_{\mathcal{E}}, \mathbf{0}_{\mathcal{F}}, !)$.

Recalling the correspondence between objects of $\mathbf{Gl}(\rho)$ and predicates, $\mathbf{syn}$ is the predicate on $\mathbf{1}_{\mathcal{E}}$ with no computable elements. What makes this proposition useful is its ability to wipe out the obligation to track computable elements. A morphism $f : \mathbf{syn} \times A \longrightarrow B$ must contain a morphism $\pi_0(f) : \pi_0(\mathbf{syn} \times A) \cong \pi_0(A) \longrightarrow \pi_0(B)$, but there are no computable elements of $\mathbf{syn} \times A$ so $\pi_0(f)$ entirely determines f; there is a bijection $\mathrm{hom}_{\mathbf{Gl}(\rho)}(\mathbf{syn} \times A, B) \cong \mathrm{hom}_{\mathcal{E}}(\pi_0(A), \pi_0(B))$. Internally, hypothesizing $\mathbf{syn}$ collapses the category to $\mathcal{E}$:

Lemma 8.4.3. *There is an equivalence $\mathcal{E} \simeq \mathbf{Gl}(\rho)/\mathbf{syn}$.*

In topos-theoretic terms, $\mathcal{E}$ is an open subtopos of $\mathbf{Gl}(\rho)$. As an open subtopos, we can present $\mathcal{E}$ internally to $\mathbf{Gl}(\rho)$ through a lex idempotent monad $\circ A = \mathbf{syn} \to A$ [RSS20]. This modality has a strongly disjoint lex idempotent modality, $\bullet A$ [RSS20, Section 3.4]. While we could work with $\bullet$ entirely through this characterization, it is helpful to fix a definition:

$$\begin{tikzcd}
\mathbf{syn} \times A \arrow[r] \arrow[d] & A \arrow[d] \\
\mathbf{syn} \arrow[r] & \bullet A \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd} \tag{8.2}$$

Intuitively, $\bullet A$ is the portion of A with a trivial $\mathcal{E}$ component. This is even clearer if one calculates the behavior of $\bullet$ on a closed type $A = (E, F, f)$ as $\bullet A = (\mathbf{1}, F, !)$. Just as hypothesizing **syn** i.e., working under $\circ$, recovers $\mathcal{E}$ internally to $\mathbf{Gl}(\rho)$, working under $\bullet$ recovers $\mathcal{F}$. Phrased in topos-theoretic terms, $\mathcal{F}$ is a *closed* subtopos of $\mathbf{Gl}(\rho)$.

The final ingredient we must add to our type theory is the *realignment axiom* [Bir+19; OP18; SH21], stating that the following canonical map has an inverse re for any $B : \mathcal{U}$:

$$\left(\textstyle\sum_{A:\mathcal{U}}[A \cong B]\right) \to \left(\textstyle\sum_{A:\mathbf{syn}\to\mathcal{U}} \prod_{z:\mathbf{syn}} A(z) \cong B\right) \tag{8.3}$$

Unfolding these conditions yields the following:

Definition 8.4.4. Fix $B : \mathcal{U}$, $A : \circ\mathcal{U}$, and $\alpha : \prod_{z:\mathbf{syn}} A(z) \cong B$. The *realignment* $\mathsf{re}(B, A, \alpha)$ of B along α is a term of type $\sum_{A^*:\mathcal{U}} A^* \cong B$ satisfying the following condition:

$$\textstyle\prod_{z:\mathbf{syn}} \mathsf{re}(B, A, \alpha) = (A(z), \alpha(z))$$

More intuitively, realignment states that a predicate lying over an object in $\mathcal{E}$ can be shifted to lie over an isomorphic object. A proper motivation of realignment is deferred to its use in Section 8.5, but broadly realignment will be used to satisfy the strict equalities demanded by Definition 8.3.4 where a priori two constants might agree only up to isomorphism.

Theorem 8.4 of Orton and Pitts [OP18] shows that a Hofmann–Streicher universe satisfies realignment for levelwise decidable propositions. Using the presentation of $\mathbf{Gl}(\rho)$ as a presheaf topos [CJ95], **syn** is clearly levelwise decidable and so realignment at **syn** is constructively valid.

Definition 8.4.5. The language of synthetic Tait computability is extensional type theory with a cumulative hierarchy of universes and a universe of propositions equipped with a distinguished proposition $\mathbf{syn} : \Omega$ such that each universe satisfies the realignment axiom for **syn**.

This subsection is summarized by the following result, which might be termed the 'fundamental lemma' of STC:

Theorem 8.4.6. $\mathbf{Gl}(\rho)$ *is a model of STC.*

8.4.2 Gluing together cosmoi

While a model in $\mathbf{Gl}(\rho)$ for a carefully chosen $\mathcal{E}$, $\mathcal{F}$, and ρ is sufficient to prove many results of MLTT [Coq19] the situation for MTT is more complex. Rather than gluing along a single functor, it is necessary to glue along an entire 2-natural transformation of continuous functors between 2-functors of presheaf topoi. We begin by considering a pair of presheaf cosmoi for the mode theory $\{\mu : n \longrightarrow m\}$ and a 2-natural transformation of right adjoints between them:

$$\begin{tikzcd} \mathcal{E}_n \arrow[r, "\rho_n"] \arrow[d, "f"'] & \mathcal{F}_n \arrow[d, "g"] \\ \mathcal{E}_m \arrow[r, "\rho_m"'] & \mathcal{F}_m \end{tikzcd} \tag{8.4}$$

Let us further assume that f and g preserve finite colimits.

Gluing 'horizontally', we obtain a pair of categories $\mathbf{Gl}(\rho_n)$ and $\mathbf{Gl}(\rho_m)$ and by Theorems 8.4.1 and 8.4.6 both are presheaf topoi and models of STC. Artin gluing is functorial, and Eq. (8.4) induce a functor $\mathbf{Gl}(f,g) : \mathbf{Gl}(\rho_n) \longrightarrow \mathbf{Gl}(\rho_m)$ sending (E_n, F_n, x) to $(f(E_n), g(F_n), g(x))$.

Lemma 8.4.7. $\mathbf{Gl}(f,g) : \mathbf{Gl}(\rho_n) \longrightarrow \mathbf{Gl}(\rho_m)$ *is a right adjoint.*

Proof. While this follows classically from the special adjoint functor theorem, an explicit construction is useful. There is a comparison $\beta : g_! \circ \rho_m \longrightarrow \rho_n \circ f_!$ induced by transposition and the unit of the $f_! \dashv f$. The left adjoint $\mathbf{Gl}(f,g)_!$ sends $f : F \longrightarrow \rho_m(E)$ to $\beta \circ g_!(f) : g_!(F) \longrightarrow \rho_n(f_!(E))$. The isomorphism $\hom([f,g]_!(X), Y) \cong \hom(X, [f,g](Y))$ is given component-wise by the isomorphisms associated with $f_! \dashv f$ and $g_! \dashv g$. □

Remark 8.4.8. This calculation show that $\pi_n : \mathbf{Gl}(n) \longrightarrow \mathcal{E}_n$ and $\pi_n : \mathbf{Gl}(m) \longrightarrow \mathcal{E}_m$ assemble into a natural transformation which satisfies Beck-Chevalley. ◇

Lemma 8.4.9. *The adjunction* $\mathbf{Gl}(f,g)_! \dashv \mathbf{Gl}(f,g)$ *induces a dependent right adjoint with respect to sufficiently large Hofmann-Streicher universe* $\mathcal{U}$.

Proof. It suffices to argue that $\mathbf{Gl}(f,g)$ sends a $\mathcal{U}$-small family in $\mathbf{Gl}(\rho_n)$ to a $\mathcal{U}$-small in $\mathbf{Gl}(\rho_m)$. This is proven by e.g., Gratzer et al. [GSS22, Lemma 3.3.7]. □

As a consequence of Lemma 8.4.9, we obtain a model of MTT with the mode theory $\{\mu : n \longrightarrow m\}$ which interprets n, m, and μ as $\mathbf{Gl}(\rho_n)$, $\mathbf{Gl}(\rho_m)$, and $\mathbf{Gl}(f,g)$ respectively. This model of MTT is particularly well-behaved: equality is extensional and $\mathbf{Gl}(f,g)$ validates the strong transposition-style elimination rules specified by Birkedal et al. [Bir+20].

Lemma 8.4.10. *In this model of MTT,* $\langle \mu \mid \mathbf{syn}_n \rangle \cong \mathbf{syn}_m$

Proof. Externally, $\mathbf{syn}_n = (\mathbf{1}, \mathbf{0}, !)$ but g preserves $\mathbf{0}$ and f preserves $\mathbf{1}$, so we have the following:

$$\mathbf{Gl}(f,g)(\mathbf{syn}_n) \cong (\mathbf{1}, \mathbf{0}, !) = \mathbf{syn}_m$$

□

Lemma 8.4.11. *In this model of MTT,* $\bigcirc\langle \mu \mid A \rangle \cong \langle \mu \mid \bigcirc A \rangle$ *and* $\bullet\langle \mu \mid A \rangle \cong \langle \mu \mid \bullet A \rangle$.

Proof. We consider the only case of $\bigcirc$, as the argument for $\bullet$ is identical. First, we observe that $\mathbf{Gl}(f,g)$ preserves $\bigcirc$ *externally*. That is, there is an isomorphism $\alpha : \mathbf{Gl}(f,g) \circ \bigcirc \cong \bigcirc \circ \mathbf{Gl}(f,g)$. It remains to show that this isomorphism can be internalized. Let us write $\tau_m : \mathcal{T}_m \longrightarrow \mathcal{T}_m$ for the universe of types in $\mathbf{Gl}(\rho_m)$ and write τ_n for its counterpart in $\mathbf{Gl}(\rho_n)$. Let us further write i, $\hat{\bigcirc}_m$, and $\hat{\bigcirc}_n$ for the cartesian natural transformations $\mathbf{Gl}(f,g)(\tau_n) \longrightarrow \tau_m$, $\bigcirc\tau_m \longrightarrow \tau_m$, and $\bigcirc\tau_n \longrightarrow \tau_n$ that are used to interpret $\langle \mu \mid - \rangle$ and $\bigcirc$ in both $\mathbf{Gl}(\rho_n)$ and $\mathbf{Gl}(\rho_m)$, respectively.

Unfolding this statement into the model, we must argue that the following pair of maps classify isomorphic families:

$$\begin{tikzcd}
\mathbf{Gl}(f,g)(\bigcirc\mathcal{T}_n) \arrow[r, "\mathbf{Gl}(f,g)(\hat{\bigcirc})"] & \mathbf{Gl}(f,g)(\mathcal{T}_n) \arrow[r, "i"] & \mathcal{T}_m
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbf{Gl}(f,g)(\bigcirc\mathcal{T}_n) \arrow[r, "\bigcirc i \circ \alpha"] & \bigcirc\mathcal{T}_m \arrow[r, "\hat{\bigcirc}"] & \mathcal{T}_m
\end{tikzcd}$$

We directly check that both classify $\mathbf{Gl}(f,g)(\bigcirc\tau_n)$ using the fact that both $\mathbf{Gl}(f,g)$ and $\bigcirc$ preserve finite limits. □

Remark 8.4.12. Technically, **syn**, $\bigcirc$, and $\bullet$ should be always annotated with a mode. In light of these results, however, we shall omit this annotation and systematically *identify* $\mathbf{syn}_m$ and $\langle \mu \mid \mathbf{syn}_n \rangle$. As both are subterminal, there are no coherence issues in this identification. ◇

Definition 8.4.13. The language of *multimodal STC* (MSTC) is extensional MTT with a cumulative hierarchy of universes and a universe of propositions such that

- Each mode is equipped with a proposition **syn**.
- Each universe satisfies the realignment axiom for **syn**.
- MTT modalities commute with **syn**, $\bigcirc$, and $\bullet$.

Summarizing the preceding discussion:

Theorem 8.4.14. $\mathbf{Gl}(\rho_n)$*,* $\mathbf{Gl}(\rho_m)$*, and* $\mathbf{Gl}(f,g)$ *assemble into a presheaf cosmos and a model of MSTC.*

In fact, it is only a small step from this result to the full fundamental lemma of multimodal STC:

Theorem 8.4.15. *Given a pair of presheaf cosmoi* $F, G : \mathcal{M} \longrightarrow \mathbf{Cat}$ *and a 2-natural transformation* $\rho : F \longrightarrow G$ *such that each* $F(\mu), G(\mu)$ *preserves finite colimits and each* ρ_m *is continuous,* $\mathbf{Gl}(\rho) : \mathcal{M} \longrightarrow \mathbf{Cat}$ *both a presheaf cosmos and a model of MSTC. Furthermore* $\pi_0 : \mathbf{Gl}(\rho) \longrightarrow F$ *is a morphism of cosmoi.*

8.5 The normalization cosmos

Recall from Section 8.2 the 2-functor of categories of renamings Ren_{-}. By an identical construction to Example 8.3.2, we obtain the cosmos of renamings $\mathcal{R}(-) = \mathbf{PSh}(\mathsf{Ren}_{-})$ and the 2-natural transformation $\mathbf{i}[-] : \mathsf{Ren}_{-} \longrightarrow \mathsf{Cx}_{-}$ acts by precomposition to yield a 2-natural transformation $\mathbf{i}[-]^* : \mathcal{S} \longrightarrow \mathcal{R}$. Theorem 8.4.15 then yields the following:

Definition 8.5.1. The normalization cosmos $\mathcal{G}$ is a presheaf cosmos and model of MSTC where $\mathcal{G}(m) = \mathbf{Gl}(\mathbf{i}[m]^*)$.

As a further consequence of Theorem 8.4.15, the projection map $\pi_0 : \mathcal{G} \longrightarrow \mathcal{S}$ is a morphism of cosmoi. In this section, we equip $\mathcal{G}$ with the structure of an MTT cosmos and show that π_0 extends to a morphism of MTT cosmoi.

8.5.1 Prerequisites for the normalization cosmos

Before we extend $\mathcal{G}$ to an MTT cosmos, we import features of $\mathcal{G}$ into the language of MSTC to specialize the latter to this situation. In this section, we begin using the interpretation of MTT to work internally to $\mathcal{G}$ and explicitly record the extensions to MSTC required for the normalization proof.

Notation 8.5.2 (Dependent open modality)*.* As $\bigcirc A = \mathbf{syn} \to A$, we will write $\bigcirc_z A(z) = (z : \mathbf{syn}) \to A(z)$ for the *dependent* version of the open modality.

Notation 8.5.3 (Extension types)*.* Given a type A, a proposition ϕ, and an element $a : \phi \to A$, we write $\{A \mid x : \phi \mapsto a(x)\}$ for subtype of A of elements equal to a under ϕ. Formally:

$$\{A \mid x : \phi \mapsto a(x)\} = \textstyle\sum_{a':A} (x : \phi) \to a' = a(x)$$

We treat the coercion $\{A \mid x : \phi \mapsto a(x)\} \to A$ as silent and refer to the equation $a' = a(x)$ as a *boundary condition.*

Recall from Example 8.3.2 that $\mathcal{S}$ already contains the structure of an MTT cosmos. As a presheaf cosmos, this manifests through a series of constants in the internal language of $\mathcal{S}$. Using Lemma 8.4.3 we import these constants into $\mathcal{G}$.

Extension 8.5.1. *For each $m : \mathcal{M}$, there is a pair of constants $z : \mathbf{syn} \vdash \mathsf{Ty}_m(z) : \mathcal{U}_0$ and $z : \mathbf{syn}, A : \mathsf{Ty}_m(z) \vdash \mathsf{Tm}_m(z, A) : \mathcal{U}_0$. These constants are further equipped with operations à la Fig. 8.3 closing them under dependent sums, dependent products, modal types, etc.*

Next, observe that normals, neutrals, and normal types are equipped with an action by renamings, so that they can be structured as presheaves over Ren_{-}. The decoding operations further organize them into proof-relevant predicates over terms and types e.g., the presheaf of normal types as an object of $\mathcal{G}$ lying over the presheaf of types from $\mathcal{S}(m)$. In fact, because renamings map variables to variables, the collection of variables of a given type organizes into a presheaf over Ren_{-} and part of an object in $\mathcal{G}$. We import these objects into the internal language as additional constants:

Extension 8.5.2. *Given $m : \mathcal{M}$ and $A : \bigcirc_z \mathsf{Ty}_m(z)$, we have constants*

$$\mathsf{Nf}_m(A), \mathsf{Ne}_m(A), \mathsf{V}_m(A) : \{\mathcal{U}_0 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}_m(z, A(z))\}$$
$$\mathsf{NfTy}_m : \{\mathcal{U}_0 \mid z : \mathbf{syn} \mapsto \mathsf{Ty}_m(z)\}$$

We treat the coercion from $\mathsf{V}_m(A)$ to $\mathsf{Ne}_m(A)$ as silent.

Notation 8.5.4*.* We frequently omit $z : \mathbf{syn}$ as an argument to $M : \bigcirc X$. For instance, given $A, B : \bigcirc\mathsf{Ty}_m$ we write $\mathsf{Nf}_m(\mathsf{Prod}(A, B))$ not $\mathsf{Nf}_m(\lambda z.\ \mathsf{Prod}(z, A(z), B(z)))$.

The normals and neutrals themselves lift to constants of type $\mathsf{Nf}_m(A)$, $\mathsf{Ne}_m(A)$, and NfTy_m using a form of higher-order abstract syntax [Hof99]. These operations collapse to the corresponding syntactic constants specified by Extension 8.5.1 under $z : \mathbf{syn}$—recall from Extension 8.5.2 that here e.g. $\mathsf{Nf}_m(A) = \mathsf{Tm}_m(z, A)$. The full collection of constants is specified in Fig. 8.4.

Extension 8.5.3. *There are constants internalizing normals, neutrals, and normal types.*

Finally, inspecting Definition 8.5.1 reveals that modalities are interpreted by functors which are both left and right adjoints. As a result, modalities preserve coproducts:

Extension 8.5.4. $\langle \mu \mid A + B \rangle \cong \langle \mu \mid A \rangle + \langle \mu \mid B \rangle$

8.5.2 The MTT cosmos

We now extend $\mathcal{G}$ to an MTT cosmos. To ensure that π_0 induces a morphism of MTT cosmoi, it suffices to ensure that each constant we add to $\mathcal{G}$ is equal to the corresponding piece of $\mathcal{S}$ as internalized by Extension 8.5.1 under $z : \mathbf{syn}$.

The universe of computable types and terms We begin with the definition of types and terms in this cosmos. Concretely, we require the following for each $m : \mathcal{M}$:

$$\begin{aligned} &\mathsf{Ty}^*_m : \{\mathcal{U}_2 \mid z : \mathbf{syn} \mapsto \mathsf{Ty}_m(z)\} \\ &\mathsf{Tm}^*_m : (A : \mathsf{Ty}^*_m) \to \{\mathcal{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}_m(z, A)\} \end{aligned}$$

We start with the following putative definition of types:

$$\begin{aligned} &\mathbf{record}\ T : \mathcal{U}_2\ \mathbf{where} \\ &\quad \mathsf{code} : \mathsf{NfTy}_m \\ &\quad \mathsf{pred} : \{\mathcal{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}_m(z, \mathsf{code})\} \\ &\quad \mathsf{reflect} : \{\mathsf{Ne}_m(\mathsf{code}) \to \mathsf{pred} \mid \mathbf{syn} \mapsto \mathsf{id}\} \\ &\quad \mathsf{reify} : \{\mathsf{pred} \to \mathsf{Nf}_m(\mathsf{code}) \mid \mathbf{syn} \mapsto \mathsf{id}\} \end{aligned} \tag{8.5}$$

In prose, $A : T$ contains the code of a normal type $A.\mathsf{code}$ as well as a proof-relevant predicate on the elements of $A.\mathsf{code}$.

The last two fields ensure that (1) all elements tracked by this predicate can be assigned normal forms, and (2) all neutrals lie within the predicate. We write $\downarrow_A$ and $\uparrow_A$ for $A.\mathsf{reify}$ and $A.\mathsf{reflect}$. Of the two, the reify is the crucial operation needed for the normalization algorithm: it ensures that computable elements can be given normal forms. Tait [Tai67], however, has shown that the pair of operations is necessary to close all type formers under just reify.

We cannot simply define $\mathsf{Ty}^*_m = T$, as T does not satisfy the equation $z : \mathbf{syn} \vdash T = \mathsf{Ty}_m(z)$. It does, however, satisfy this condition up to isomorphism: under $z : \mathbf{syn}$, the types of pred, $\mathsf{reflect}$, and reify collapse to singletons, while the type of code collapses to $\mathsf{Ty}_m(z)$ by Extension 8.5.2:

$$\alpha_{\bigcirc}(z, A) = A.\mathsf{code} : \textstyle\prod_{z:\mathbf{syn}} T \cong \mathsf{Ty}^*_m(z)$$

$$\begin{aligned}
&\mathbf{Prod} : (A : \mathsf{NfTy}_m)(B : \mathsf{V}_m(A) \to \mathsf{NfTy}_m) \to \mathsf{NfTy}_m\\
&\mathbf{Sum} : (A : \mathsf{NfTy}_m)(B : \mathsf{V}_m(A) \to \mathsf{NfTy}_m) \to \mathsf{NfTy}_m\\
&\mathbf{Bool} : \mathsf{NfTy}_m\\
&\mathbf{Mod}_\mu : (\mu \mid \mathsf{NfTy}_n) \to \mathsf{NfTy}_m
\end{aligned}$$

$$\begin{aligned}
&\mathbf{lam} : (A : \bigcirc\mathsf{Ty}_m)(B : \bigcirc\mathsf{Tm}_m(A) \to \bigcirc\mathsf{Ty}_m)\\
&\quad \to ((a : \mathsf{V}_m(A)) \to \mathsf{Nf}_m(B(a))) \to \mathsf{Nf}_m(\mathsf{Prod}(A, B))\\
&\mathbf{app} : (A : (\mu \mid \bigcirc\mathsf{Ty}_m))(B : \bigcirc\mathsf{Tm}_m(A) \to \bigcirc\mathsf{Ty}_m)\\
&\quad \to \mathsf{Ne}_m(\mathsf{Prod}(A, B)) \to (a : \mathsf{Nf}_m(A)) \to \mathsf{Ne}_m(B(a))
\end{aligned}$$

$$\begin{aligned}
&\mathbf{up} : \mathsf{Ne}_m(\mathsf{Bool}) \to \mathsf{Nf}_m(\mathsf{Bool})\\
&\mathbf{tt}, \mathbf{ff} : \mathsf{Nf}_m(\mathsf{Bool})\\
&\mathbf{if} : (A : \mathsf{V}_m(\mathsf{Bool}) \to \mathsf{NfTy}_m)\\
&\quad \to \mathsf{Nf}_m(A(\mathsf{true})) \to \mathsf{Nf}_m(A(\mathsf{false})) \to (b : \mathsf{Ne}_m(\mathsf{Bool})) \to \mathsf{Ne}_m(A(b))
\end{aligned}$$

$$\begin{aligned}
&\mathbf{up} : (A : \bigcirc\mathsf{Ty}_m)(a_0, a_1 : \bigcirc\mathsf{Tm}_m(A))\\
&\quad \to \mathsf{Ne}_m(\mathsf{Id}(A, a_0, a_1)) \to \mathsf{Nf}_m(\mathsf{Id}(A, a_0, a_1))\\
&\mathbf{refl} : (A : \bigcirc_z\mathsf{Ty}_m(z))(a : \bigcirc_z\mathsf{Tm}_m(z, A(z))) \to \mathsf{Nf}_m(\mathsf{Id}(A, a, a))\\
&\mathbf{J} : (A : \bigcirc\mathsf{Ty}_m)\,(B : (a_0, a_1 : \mathsf{V}_m(A))(p : \mathsf{V}_m(\mathsf{Id}(A, a_0, a_1))) \to \mathsf{NfTy}_m)\\
&\quad \to ((a : \mathsf{V}_m(A)) \to \mathsf{Nf}_m(B(a, a, \mathsf{refl}(a))))\,(a_0, a_1 : \bigcirc_z\mathsf{Tm}_m(A))(p : \mathsf{Ne}_m(\mathsf{Id}(A, a_0, a_1)))\\
&\quad \to \mathsf{Ne}_m(B(a_0, a_1, \eta(p)))
\end{aligned}$$

$$\begin{aligned}
&\mathbf{up} : (A : (\mu \mid \mathsf{Ty}_n)) \to \mathsf{Ne}_m(\mathsf{Mod}_\mu(A)) \to \mathsf{Nf}_m(\mathsf{Mod}_\mu(A))\\
&\mathbf{mod}_\mu : (A : (\mu \mid \bigcirc\mathsf{Ty}_n))(\mu \mid \mathsf{Nf}_n(A)) \to \mathsf{Nf}_m(\lambda z.\ \mathsf{Mod}_\mu(z, A(z)))\\
&\mathbf{letmod}_{\mu;\nu} : (A : (\nu \circ \mu \mid \bigcirc\mathsf{Ty}_n))\,(B : (a : (\nu \mid \mathsf{V}_m(\mathsf{Mod}_\mu(A)))) \to \mathsf{NfTy}_o)\\
&\quad \to ((a : (\nu \circ \mu \mid \mathsf{V}_n(A))) \to \mathsf{Nf}_o(B(\mathsf{m}_\mu(a))))\\
&\quad \to (a : (\nu \mid \mathsf{Ne}_m(\mathsf{Mod}_\mu(A)))) \to \mathsf{Ne}_o(B(a))
\end{aligned}$$

$$\begin{aligned}
&\mathbf{Uni} : \mathsf{NfTy}_m\\
&\mathbf{El} : \mathsf{Nf}_m(\mathsf{Uni}) \to \mathsf{NfTy}_m\\
&\mathbf{up} : \mathsf{Ne}_m(\mathsf{Uni}) \to \mathsf{Nf}_m(\mathsf{Uni})\\
&\widehat{\mathbf{Mod}}_\mu : (\mu \mid \mathsf{Nf}_n(\mathsf{Uni})) \to \mathsf{Nf}_m(\mathsf{Uni})\\
&\mathbf{dec}^{\triangleright}_{\widehat{\mathbf{Mod}}_\mu} : (A : (\mu \mid \mathsf{Nf}_n(\mathsf{Uni}))) \to \mathsf{Nf}_m(\mathsf{Mod}_\mu(A)) \to \mathsf{Nf}_m(\mathsf{El}(\widehat{\mathsf{Mod}}(A)))\\
&\mathbf{dec}^{\triangleleft}_{\widehat{\mathbf{Mod}}_\mu} : (A : (\mu \mid \mathsf{Nf}_n(\mathsf{Uni}))) \to \mathsf{Ne}_m(\mathsf{El}(\widehat{\mathsf{Mod}}(A))) \to \mathsf{Ne}_m(\mathsf{Mod}_\mu(A))
\end{aligned}$$

Figure 8.4: Neutral and normal forms, internally

Observe $(\mathsf{Ty}_m, \alpha_{\bigcirc}) : \sum_{A:\bigcirc\mathcal{U}} \prod_{z:\mathbf{syn}} A(z) \cong T$, so the realignment axiom of Definition 8.4.4 applies and we can define

$$(\mathsf{Ty}^*_m, \alpha) = \mathsf{re}(T, \mathsf{Ty}_m, \alpha_{\bigcirc}) \tag{8.6}$$

The equation $z : \mathbf{syn} \vdash \mathsf{Ty}^*_m = \mathsf{Ty}_m(z)$ follows immediately from the second half of Definition 8.4.4. On elements $A : \mathsf{Ty}^*_m$, this implies $z : \mathbf{syn} \vdash A = \alpha(A).\mathsf{code}$. For readability, we continue to use record notation to manipulate Ty^*_m.

Given $A : \mathsf{Ty}^*_m$, we define $\mathsf{Tm}^*_m(A)$:

$$\mathsf{Tm}^*_m(A) = A.\mathsf{pred} : \{\mathcal{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}^*_m(z, A)\} \tag{8.7}$$

To see that this is well-typed, we must show $\mathsf{Tm}^*_m(A) = \mathsf{Tm}_m(z, A)$ given $z : \mathbf{syn}$. The type of $A.\mathsf{code}$ in Eq. (8.5) ensures $\mathsf{Tm}^*_m(A) = \mathsf{Tm}_m(z, A.\mathsf{code})$. We have observed that $A = A.\mathsf{code}$ under $z : \mathbf{syn}$ so $\mathsf{Tm}^*_m(A) = \mathsf{Tm}_m(z, A)$.

Type connectives It remains only to close $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ under all connectives. For mode-local connectives, these constructions are very similar to those given by Sterling [Ste21] (Lemmas 8.5.7 to 8.5.10). Modal types and dependent products, however, are involve modalities and thus are different than the other connectives (Lemmas 8.5.5 and 8.5.6).

Lemma 8.5.5. $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ *is closed under dependent products.*

Proof. We must define two constants:

$$\begin{aligned}
&\mathsf{Prod}^* : (A : (\mu \mid \mathsf{Ty}^*_n))(B : (\mu \mid \mathsf{Tm}^*_n(A)) \to \mathsf{Ty}^*_m) \to \mathsf{Ty}^*_m \\
&\alpha_{\mathsf{Prod}^*} : (A : (\mu \mid \mathsf{Ty}^*_n))(B : (\mu \mid \mathsf{Tm}^*_n(A)) \to \mathsf{Ty}^*_m) \to \mathsf{Ty}^*_m \\
&\quad \to \mathsf{Tm}^*_m(\mathsf{Prod}^*(A, B)) \cong [(a : (\mu \mid \mathsf{Tm}^*_n(A)))\mathsf{Tm}^*_m(B(a))]
\end{aligned}$$

Additionally, we must show that if $z : \mathbf{syn}$ then $\mathsf{Prod}^* = \mathsf{Prod}(z)$ and $\alpha_{\mathsf{Prod}^*} = \alpha_{\mathsf{Prod}}(z)$.

We begin by fixing $(A : (\mu \mid \mathsf{Ty}^*_m))$ and $B : (\mu \mid \mathsf{Tm}^*_n(A)) \to \mathsf{Ty}^*_m$. Define $\Phi = (a : (\mu \mid \mathsf{Tm}^*_n(A))) \to \mathsf{Tm}^*_m(B(a))$ and observe under $z : \mathbf{syn}$, the following equality holds:

$$\Phi = (a : (\mu \mid \mathsf{Tm}_n(z, A))) \to \mathsf{Tm}_m(B(z, a))$$

We may apply realignment using $\alpha_{\mathsf{Prod}}(z) : \mathsf{Tm}_m(z, \mathsf{Prod}(z, A, B)) \cong \Phi$. This realignment yields a type Ψ and isomorphism $\beta : \Psi \cong \Phi$. Under $z : \mathbf{syn}$, these restrict to $\mathsf{Tm}_m(z, \mathsf{Prod}(z, A, B))$ and $\alpha_{\mathsf{Prod}}(z)$ respectively.

With these to hand we define Prod^* and α_{Prod^*} as follows:

$$\begin{aligned}
&\mathsf{Prod}^*(A, B).\mathsf{code} = \mathbf{Prod}(A.\mathsf{code}, \lambda v.\ B(\downarrow_A v).\mathsf{code}) \\
&\mathsf{Prod}^*(A, B).\mathsf{pred} = \Psi \\
&\mathsf{Prod}^*(A, B).\mathsf{reflect} = \lambda e.\ \beta^{-1}(\lambda a.\ \mathbf{app}(e, \downarrow_A a)) \\
&\mathsf{Prod}^*(A, B).\mathsf{reify} = \lambda f.\ \mathbf{lam}(\lambda v.\ \downarrow_{B(\uparrow_A v)} \beta(f)(\uparrow_A v)) \\
&\alpha_{\mathsf{Prod}^*} = \beta
\end{aligned}$$

It remains to check a variety of boundary conditions under $z : \mathbf{syn}$. In particular, we must show that $\mathsf{Prod}^*(A, B) = \mathsf{Prod}(z, A, B)$ and that $\mathsf{reflect}$ and reify become the identity. These follow directly from assumptions about A, B, and the boundaries of various constructors. For instance

$$\begin{aligned} \mathsf{Prod}^*(A,B) &= \mathsf{Prod}^*(A,B).\mathsf{code} \\ &= \mathbf{Prod}(A.\mathsf{code}, \lambda v.\ B(\downarrow_A v).\mathsf{code}) \\ &= \mathsf{Prod}(z, A.\mathsf{code}, \lambda v.\ B(\downarrow_A v).\mathsf{code}) \\ &= \mathsf{Prod}(z, A, \lambda v.\ B(\downarrow_A v)) \\ &= \mathsf{Prod}(z, A, B) \end{aligned}$$

□

Lemma 8.5.6. $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ *is closed under modal types.*

Proof. Fix a modality $\mu : n \longrightarrow m$. In this case we define the four constants specified by Fig. 8.3, subject to the expected boundary conditions. Fix a variable $A : \mathsf{Ty}^*_n$ under the modal annotation μ i.e., $A : (\mu \mid \mathsf{Ty}^*_n)$. We define the unaligned predicate as follows:

$$\begin{array}{l} \textbf{record}\ \Phi : \mathfrak{U}_1\ \textbf{where} \\ \quad \mathsf{tm} : \mathsf{Nf}_m(\mathsf{Mod}_\mu(A)) \\ \quad \mathsf{prf} : \bullet\left(\begin{array}{l} \quad\sum_{e:\mathsf{Ne}_m(\mathsf{Mod}_\mu(A))} \mathsf{tm} = \mathbf{up}(e) \\ + \sum_{a:\langle\mu \mid A.\mathsf{pred}\rangle} \mathsf{tm} = \mathbf{mod}_\mu(\downarrow_A a) \end{array}\right) \end{array}$$

For the first time, we have used the closed modality $\bullet$ to explicitly tweak the proof-relevant predicate. Intuitively, Φ is a predicate on $\mathsf{Tm}_m(z, \mathsf{Mod}_\mu(z, A))$ and tm ensures that this predicate tracks elements with normals forms. The second field, moreover, ensures that these normal are either neutral or $\mathsf{mod}_\mu(a)$ where a is computable. Without the closed modality shielding the second field of Φ, however, this could never have the correct extent along $z : \mathbf{syn}$. Using $\circ\bullet X \cong \mathbf{1}$ and the boundary of $\mathsf{Nf}_m(\mathsf{Mod}_\mu(A))$, we can now define the following isomorphism:

$$\alpha_\circ(z, p) = p.\mathsf{tm} : \textstyle\prod_{z:\mathbf{syn}} \Phi \cong \mathsf{Tm}_m(z, \mathsf{Mod}_\mu(z, A))$$

Realigning Φ along $\alpha_\circ$, we obtain Ψ and $\alpha : \Psi \cong \Phi$ which under $z : \mathbf{syn}$ become $\mathsf{Tm}_m(z, \mathsf{Mod}_\mu(z, A))$ and $\alpha_\circ$.

We now define Mod^*_μ:

$$\begin{aligned} &\mathsf{Mod}^*_\mu(A).\mathsf{code} = \mathbf{Mod}_\mu(A.\mathsf{code}) \\ &\mathsf{Mod}^*_\mu(A).\mathsf{pred} = \Psi \\ &\mathsf{Mod}^*_\mu(A).\mathsf{reflect} = \lambda e.\ \alpha^{-1}\langle \mathbf{up}(e), \eta_\bullet \mathsf{in}_1\langle e, \star\rangle\rangle \\ &\mathsf{Mod}^*_\mu(A).\mathsf{reify} = \lambda m.\ \alpha(m).\mathsf{tm} \end{aligned}$$

Unlike Lemma 8.5.5, the introduction and elimination principles are not automatically obtained from α and they must be constructed separately:

$$\mathsf{m}^*_\mu(A, a) = \alpha^{-1}\langle \downarrow_A a, \eta_\bullet \mathsf{in}_2\langle a, \star\rangle\rangle$$

It remains to define the elimination principle $\mathsf{letmod}^*_{\mu;\nu}$. This is an involved affair and we describe it step-by-step. Begin by fixing $\nu : m \longrightarrow o$ along with the following:

$$B : (\nu \mid \mathsf{Tm}^*_m(\mathsf{Mod}^*_\mu(A))) \to \mathsf{Ty}_o$$

$$\begin{aligned}
&b : (x : (\nu \circ \mu \mid \mathsf{Tm}^*_n(A))) \to \mathsf{Tm}^*_o(B(\mathsf{m}^*_\mu(A, x))) \\
&m : (\nu \mid \mathsf{Tm}^*_m(\mathsf{Mod}^*_\mu(A)))
\end{aligned}$$

We must construct an element of $\mathsf{Tm}^*_o(B(a))$. We begin by inspecting m. As MTT modalities in extensional MTT commute with dependent sums, equality, $\bullet$, and—by Extension 8.5.4—with finite coproducts, m can be decomposed into the following:

$$\begin{aligned}
&\mathsf{tm} : (\nu \mid \mathsf{Nf}_m(\mathsf{Mod}_\mu(A))) \\
&\mathsf{prf} : \bullet\begin{pmatrix} \sum_{e:\langle \nu \mid \mathsf{Ne}_m(\mathsf{Mod}_\mu(A))\rangle} \mathsf{mod}_\nu(\mathsf{tm}) = \mathbf{up} \circledast e \\ + \sum_{a:\langle \nu\circ\mu \mid A.\mathsf{pred}\rangle} \mathsf{mod}_\nu(\mathsf{tm}) = (\mathbf{mod}_\mu \circ {\downarrow_A}) \circledast a \end{pmatrix}
\end{aligned}$$

Recall from Eq. (8.2) that $\bullet X$ is a pushout of $\mathbf{syn}$ and X. To define a map out of $\bullet X$, therefore, it suffices to define a map out of X which is constant assuming $z : \mathbf{syn}$. We conclude by scrutinizing prf:

$$\begin{cases} \mathsf{in}_1(\mathsf{mod}_\nu(e), \_) \mapsto {\uparrow}\mathbf{letmod}_{\mu;\nu}(A, \lambda v.\ B(\uparrow v).\mathsf{code}, \lambda x.\ {\downarrow} b(\uparrow x), e) \\ \mathsf{in}_2(\mathsf{mod}_\nu(a), \_) \mapsto b(a) \end{cases}$$

Given $z : \mathbf{syn}$, both branches collapse to $\mathsf{letmod}_{\mu;\nu}(z, A, B, b, a)$ so this yields a well-defined map. The boundary conditions follow from routine computations. □

Lemma 8.5.7. $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ *is closed under dependent sums.*

Proof. Fixing $A : \mathsf{Ty}^*_m$ and $B : \mathsf{Tm}^*_m(A) \to \mathsf{Ty}^*_m$. We must construct the following pair of constants:

$$\begin{aligned}
&\mathsf{Sig}^*(A, B) : \mathsf{Ty}^*_m \\
&\qquad \alpha_{\mathsf{Sig}^*} : \mathsf{Tm}_m(\mathsf{Sig}^*(A, B)) \cong \textstyle\sum_{a:\mathsf{Tm}^*_m(A)} \mathsf{Tm}^*_m(B(a))
\end{aligned}$$

Such that they lie over Sig and α_{Sig} respectively.

We begin by applying realignment to the following:

$$\left(\textstyle\sum_{a:A.\mathsf{pred}} B(a).\mathsf{pred}, \alpha_{\mathsf{Sig}(z)}\right)$$

This produces $\Psi : \mathcal{U}_1$ and $\alpha_{\mathsf{Sig}^*} : \Psi \cong \sum_{a:A.\mathsf{pred}} B(a).\mathsf{pred}$ such that under the assumption $z : \mathbf{syn}$ the following holds:

$$\Psi = \mathsf{Sig}(z, A, B) \qquad \alpha_{\mathsf{Sig}^*} = \alpha_{\mathsf{Sig}}(z)$$

We now define $\mathsf{Sig}^*(A, B)$ as follows:

$$\begin{aligned}
&\mathsf{Sig}^*(A, B).\mathsf{code} = \mathbf{Sum}(\mathsf{code}A, \lambda v.\ B.\mathsf{code}({\uparrow_A} v)) \\
&\mathsf{Sig}^*(A, B).\mathsf{pred} = \Psi \\
&\mathsf{Sig}^*(A, B).\mathsf{reflect} = \lambda e.\ \alpha^{-1}_{\mathsf{Sig}^*}\langle {\uparrow_A}(\mathbf{proj}_0(e)), {\uparrow_{B({\uparrow_A}(\mathbf{proj}_0(e)))}}(\mathbf{proj}_1(e))\rangle \\
&\mathsf{Sig}^*(A, B).\mathsf{reify} = \lambda p.\ \mathbf{pair}({\downarrow_A}(\alpha_{\mathsf{Sig}^*} p.0), {\downarrow_{B(\alpha_{\mathsf{Sig}^*} p.0)}}(\alpha_{\mathsf{Sig}^*} p.1))
\end{aligned}$$

The fact that $\downarrow$ and $\uparrow$ lie over the identity follows directly from the β and η laws of dependent sums in MTT. We show the calculations for $\uparrow$. Fix $z : \mathbf{syn}$:

$${\uparrow_{\mathsf{Sig}^*(A,B)}}(e) = \alpha^{-1}_{\mathsf{Sig}^*}\langle {\uparrow_A}(\mathbf{proj}_0(e)), {\uparrow_{B({\uparrow_A}(\mathbf{proj}_0(e)))}}(\mathbf{proj}_1(e))\rangle$$

$$
\begin{aligned}
&= \alpha_{\mathsf{Sig}}^{-1}\langle \mathbf{proj}_0(e), \mathbf{proj}_1(e)\rangle \\
&= \alpha_{\mathsf{Sig}}^{-1}\langle \alpha_{\mathsf{Sig}(A,B)}(e)_0, \alpha_{\mathsf{Sig}(A,B)}(e)_1\rangle \\
&= e
\end{aligned}
$$

Finally, $\mathsf{Sig}^*(A,B).\mathsf{code}$ and $\mathsf{Sig}^*(A,B).\mathsf{pred}$ lie over $\mathsf{Sig}(A,B)$ and $\mathsf{Tm}_m(z,\mathsf{Sig}(z,A,B))$ by inspecting their definition and realignment. □

Lemma 8.5.8. $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ *is closed under booleans.*

Proof. We must implement the following constants:

$$
\begin{aligned}
&\mathsf{Bool}^* : \{\mathsf{Ty}^*_m \mid z : \mathbf{syn} \mapsto \mathsf{Bool}(z)\} \\
&\mathsf{true}^* : \{\mathsf{Tm}^*_m(\mathsf{Bool}) \mid z : \mathbf{syn} \mapsto \mathsf{true}\} \\
&\mathsf{false}^* : \{\mathsf{Tm}^*_m(\mathsf{Bool}) \mid z : \mathbf{syn} \mapsto \mathsf{false}\} \\
&\mathsf{if}^* : (A : \mathsf{Tm}^*_m(\mathsf{Bool}(z)) \to \mathsf{Ty}^*_m) \\
&\quad \to \mathsf{Tm}^*_m(A(\mathsf{true}^*)) \\
&\quad \to \mathsf{Tm}^*_m(A(\mathsf{false}^*)) \\
&\quad \to (b : \mathsf{Tm}^*_m(\mathsf{Bool}^*)) \\
&\quad \to \{\mathsf{Tm}^*_m(A(b)) \mid z : \mathbf{syn} \mapsto \mathsf{if}(A,t,f,b)\} \\
&\_ : (A : \mathsf{Tm}^*_m(\mathsf{Bool}^*) \to \mathsf{Ty}^*_m) \\
&\quad \to (t : \mathsf{Tm}^*_m(A(\mathsf{true}^*))) \\
&\quad \to (f : \mathsf{Tm}^*_m(A(\mathsf{false}^*))) \\
&\quad \to (\mathsf{if}^*(A,t,f,\mathsf{true}^*) = t) \times (\mathsf{if}^*(A,t,f,\mathsf{false}^*) = f)
\end{aligned}
$$

First, we define Φ by realignment:

$$
\begin{aligned}
&\mathbf{record}\ \Phi : \{\mathcal{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}_m(z,\mathsf{Bool})\}\ \mathbf{where} \\
&\quad \mathsf{tm} : \mathsf{Nf}_m(\mathsf{Bool}) \\
&\quad \mathsf{prf} : \bullet\left(\begin{array}{r} \sum_{e:\mathsf{Ne}_m(\mathsf{Bool})} \mathsf{tm} = \mathbf{up}(e) \\ + \sum_{b:\mathbf{2}} \mathsf{tm} = \mathsf{rec}_{\mathbf{2}}(b;\mathsf{tt};\mathsf{ff})) \end{array}\right)
\end{aligned}
$$

We may now define Bool^*:

$$
\begin{aligned}
&\mathsf{Bool}^*.\mathsf{code} = \mathbf{Bool} \\
&\mathsf{Bool}^*.\mathsf{pred} = \Phi \\
&\mathsf{Bool}^*.\mathsf{reflect} = \lambda e.\langle \mathbf{up}(e), \eta(\mathsf{in}_1(e,\star))\rangle \\
&\mathsf{Bool}^*.\mathsf{reify} = \lambda b.\ b.\mathsf{tm}
\end{aligned}
$$

It remains to define the introduction and elimination forms.

$$
\begin{aligned}
\mathsf{true}^* &= \langle \mathbf{tt}, \eta(\mathsf{in}_2(0,\star))\rangle \\
\mathsf{false}^* &= \langle \mathbf{ff}, \eta(\mathsf{in}_2(1,\star))\rangle
\end{aligned}
$$

The elimination form is defined by constructing a map out of $\bullet X$, by taking advantage of its definition as a pushout (Eq. (8.2)):

$$
\mathsf{if}^*(A, t_0, t_1, b = \langle \mathsf{tm}, \mathsf{prf}\rangle) =
$$

$$\begin{cases} \mathsf{if}(z, T_m, t_0, t_1, s) & \mathsf{prf} = \mathsf{in}_1(z) \\ \downarrow_{A(b)}\mathbf{if}(\lambda v.\ A(\uparrow v).\mathsf{code}, \downarrow t_0, \downarrow t_1, e) & \mathsf{prf} = \mathsf{in}_2(\mathsf{in}_1(e, \_)) \\ t_i & \mathsf{prf} = \mathsf{in}_2(\mathsf{in}_2(i, \_)) \end{cases}$$

□

Lemma 8.5.9. $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ *is closed under intensional identity types.*

Proof. We must implement the following constants:

$$\begin{aligned}
&\mathsf{Id}^* : (A : \mathsf{Ty}^*_m)(a_0, a_1 : \mathsf{Tm}^*_m(A)) \\
&\quad \to \{\mathsf{Ty}^*_m \mid z : \mathbf{syn} \mapsto \mathsf{Id}(z, A, a_0, a_1)\} \\
&\mathsf{refl}^* : (A : \mathsf{Ty}^*_m)(a : \mathsf{Tm}^*_m(A)) \\
&\quad \to \{\mathsf{Tm}^*_m(\mathsf{Id}(A, a, a)) \mid z : \mathbf{syn} \mapsto \mathsf{refl}(z, A, a)\} \\
&\mathsf{J}^* : (A : \mathsf{Ty}^*_m) \\
&\quad \to (B : (a_0, a_1 : \mathsf{Tm}^*_m(A)) \to \mathsf{Tm}^*_m(\mathsf{Id}^*(A, a_0, a_1)) \to \mathsf{Ty}^*_m) \\
&\quad \to (b : (a : \mathsf{Tm}^*_m(A)) \to \mathsf{Tm}^*_m(B(a, a, \mathsf{refl}(a)))) \\
&\quad \to (a_0, a_1 : \mathsf{Tm}^*_m(A))(p : \mathsf{Tm}^*_m(\mathsf{Id}^*(A, a_0, a_1))) \\
&\quad \to \{\mathsf{Tm}^*_m(B(a_0, a_1, p)) \mid z : \mathbf{syn} \mapsto \mathsf{J}(z, B, b, p)\} \\
&\_ : (A : \mathsf{Ty}^*_m) \\
&\quad \to (B : (a_0, a_1 : \mathsf{Tm}^*_m(A)) \to \mathsf{Tm}^*_m(\mathsf{Id}^*(A, a_0, a_1)) \to \mathsf{Ty}^*_m) \\
&\quad \to (b : (a : \mathsf{Tm}^*_m(A)) \to \mathsf{Tm}^*_m(B(a, a, \mathsf{refl}(a)))) \\
&\quad \to (a : \mathsf{Tm}^*_m(A)) \to \mathsf{J}^*(A, B, b, \mathsf{refl}^*(a)) = b(a)
\end{aligned}$$

Fix $A : \mathsf{Ty}^*_m$ and $a_0, a_1 : \mathsf{Tm}^*_m(A)$. Just as with the normalization structure for booleans, we begin by defining Φ by realignment:

$$\begin{aligned}
&\mathbf{record}\ \Phi : \{\mathcal{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}_m(z, \mathsf{Id}(A, a_0, a_1))\}\ \mathbf{where} \\
&\quad \mathsf{tm} : \mathsf{Nf}_m(\mathsf{Id}(A, a_0, a_1)) \\
&\quad \mathsf{prf} : \bullet\begin{pmatrix} \sum_{e:\mathsf{Ne}_m(\mathsf{Id}(A, a_0, a_1))} \mathsf{tm} = \mathbf{up}(e) \\ + \sum_{a:A.\mathsf{pred}} \mathsf{tm} = \mathsf{refl}(\downarrow_A a) \end{pmatrix}
\end{aligned}$$

We now define Id^*:

$$\begin{aligned}
\mathsf{Id}^*(A, a_0, a_1).\mathsf{code} &= \mathsf{Id}_{\mathsf{code}A}(\uparrow_A a_0, \uparrow_A a_1) \\
\mathsf{Id}^*(A, a_0, a_1).\mathsf{pred} &= \Phi \\
\mathsf{Id}^*(A, a_0, a_1).\mathsf{reflect} &= \lambda e.\langle \mathbf{up}(e), \eta(\mathsf{in}_1(e, \star))\rangle \\
\mathsf{Id}^*(A, a_0, a_1).\mathsf{reify} &= \lambda p.\ p.\mathsf{tm}
\end{aligned}$$

We define reflexivity by $\mathsf{refl}^* = \langle \mathsf{refl}, \eta(\mathsf{in}_2(\star, \star))\rangle$. Finally, the elimination principle is defined using the induction principle for $\bullet X$.

$$\begin{aligned}
&\mathsf{J}^*(B, b, a_0, a_1, p = \langle \mathsf{tm}, \mathsf{prf}\rangle) = \\
&\begin{cases} \mathsf{J}(z, B, b, a_0, a_1, p) & \mathsf{prf} = \mathsf{in}_1(z) \\ \downarrow\mathbf{J}(\lambda l, r, p.B(\uparrow l, \uparrow r, \uparrow p).\mathsf{code}, \lambda a.\downarrow b(\uparrow a), e) & \mathsf{prf} = \mathsf{in}_2(\mathsf{in}_1(e, \_)) \\ b(a_0) & q = \mathsf{in}_2(\mathsf{in}_2(\_, \_)) \end{cases}
\end{aligned}$$

□

Lemma 8.5.10. $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ *is closed under a universe.*

Proof. We begin by constructing the two constants for the universe and the decoding family:

$$\begin{aligned}&\mathsf{Uni}^* : \{\mathsf{Ty}^*_m \mid z : \mathbf{syn} \mapsto \mathsf{Uni}\}\\&\mathsf{El}^* : (A : \mathsf{Tm}^*_m(\mathsf{Uni}^*)) \to \{\mathsf{Ty}^*_m \mid z : \mathbf{syn} \mapsto \mathsf{El}(A)\}\end{aligned}$$

At this point we take advantage of the fact that pred is an element of $\mathcal{U}_1$; in particular, we use the fact that is a universe $\mathcal{U}_0$ small enough to fit inside $\mathcal{U}_1$.

We may then define Ψ by realigning the following element of $\mathcal{U}_1$ along the evident isomorphism to $\mathsf{Tm}^*_m(z, \mathsf{Uni}(z))$:

$$\begin{aligned}&\mathbf{record}\ \Psi : \{\mathcal{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}^*_m(z, \mathsf{Uni})\}\ \mathbf{where}\\&\quad \mathsf{code} : \mathsf{Nf}_m(\mathsf{Uni})\\&\quad \mathsf{pred} : \{\mathcal{U}_0 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}_m(z, \mathsf{El}(\mathsf{code}))\}\\&\quad \mathsf{reflect} : \{\mathsf{Ne}_m(\mathsf{El}(\mathsf{code})) \to \mathsf{pred} \mid z : \mathbf{syn} \mapsto \mathsf{id}\}\\&\quad \mathsf{reify} : \{\mathsf{pred} \to \mathsf{Nf}_m(\mathsf{El}(\mathsf{code})) \mid z : \mathbf{syn} \mapsto \mathsf{id}\}\end{aligned}$$

With Ψ in hand, we may define Uni^*:

$$\begin{aligned}&\mathsf{Uni}^*.\mathsf{code} = \mathbf{Uni}\\&\mathsf{Uni}^*.\mathsf{pred} = \Psi\\&\mathsf{Uni}^*.\mathsf{reflect} = \lambda e.\ \langle \mathbf{up}(e); \mathsf{Ne}_m; \mathsf{id}; \lambda e.\ \mathbf{up}(e)\rangle\\&\mathsf{Uni}^*.\mathsf{reify} = \lambda A.\ A.\mathsf{code}\end{aligned}$$

The definition of El^* is essentially cumulativity:

$$\mathsf{El}^*(\langle \mathsf{code}; \mathsf{pred}; \mathsf{reify}; \mathsf{reflect}\rangle) = \langle \mathsf{El}(\mathsf{code}); \mathsf{pred}; \mathsf{reify}; \mathsf{reflect}\rangle$$

It remains to show that $(\mathsf{Uni}^*, \mathsf{El}^*)$ is closed under various type formers. We show a representative cases: modal types. This concretely entails implementing the following constants:

$$\begin{aligned}&\widehat{\mathsf{Mod}}^* : (A : (\mu \mid \mathsf{Tm}^*_n(\mathsf{Uni}^*))) \to \{\mathsf{Tm}^*_m(\mathsf{Uni}^*) \mid z : \mathbf{syn} \mapsto \widehat{\mathsf{Mod}}(z, A)\}\\&\mathsf{dec}^*_{\widehat{\mathsf{Mod}}} : (A : (\mu \mid \mathsf{Tm}^*_n(\mathsf{Uni}^*)))\\&\quad \to \{\mathsf{Tm}^*_m(\mathsf{El}^*(\widehat{\mathsf{Mod}}^*(A))) \cong \mathsf{Tm}^*_m(\mathsf{Mod}^*_\mu(\mathsf{El}^*(A))) \mid z : \mathbf{syn} \mapsto \mathsf{dec}_{\widehat{\mathsf{Mod}}}(z, A)\}\end{aligned}$$

Fix $A : (\mu \mid \mathsf{Tm}^*_n(\mathsf{Uni}^*))$. We realign $\mathsf{Tm}^*_m(\mathsf{Mod}^*_\mu(\mathsf{El}^*(A)))$ along the isomorphism $\mathsf{dec}_{\widehat{\mathsf{Mod}}}$ to obtain a type Ψ and an isomorphism:

$$\mathsf{dec}^*_{\mathsf{Mod}_\mu} : \{\mathsf{Tm}^*_m(\mathsf{El}^*(\widehat{\mathsf{Mod}}^*(A))) \cong \mathsf{Tm}^*_m(\mathsf{Mod}^*_\mu(\mathsf{El}^*(A))) \mid z : \mathbf{syn} \mapsto \mathsf{dec}_{\widehat{\mathsf{Mod}}}(z, A)\}$$

It remains only to define $\widehat{\mathsf{Mod}}^*(A)$ such that $\widehat{\mathsf{Mod}}^*(A).\mathsf{pred} = \Psi$:

$$\begin{aligned}&\widehat{\mathsf{Mod}}^*(A).\mathsf{code} = \mathsf{Mod}_\mu(A.\mathsf{code})\\&\widehat{\mathsf{Mod}}^*(A).\mathsf{pred} = \Psi\end{aligned}$$

$$\widehat{\mathsf{Mod}}^{*}(A).\mathsf{reflect} = \lambda e.\ (\mathsf{dec}^{*}_{\widehat{\mathsf{Mod}}})^{-1}(\uparrow_{\mathsf{Mod}^{*}_{\mu}(\mathsf{El}^{*}(A))}\mathsf{dec}^{\rhd}(e))$$
$$\widehat{\mathsf{Mod}}^{*}(A).\mathsf{reify} = \lambda m.\ \mathsf{dec}^{\lhd}(\downarrow_{\mathsf{Mod}^{*}_{\mu}(\mathsf{El}^{*}(A))}\mathsf{dec}^{*}_{\widehat{\mathsf{Mod}}}(m))$$

The checks that all constructions lie over their syntactic counterparts follow immediately from the conclusions of realignment. □

Theorem 8.5.11. *$\mathcal{G}$ supports an MTT cosmos built around* $(\mathsf{Ty}^{*}_{m}, \mathsf{Tm}^{*}_{m})$ *and* $\pi_0 : \mathcal{G} \longrightarrow \mathcal{S}$ *is a map of MTT cosmoi.*

8.6 The normalization algorithm

After Theorem 8.5.11, it remains only to parlay the existence of the normalization cosmos into a normalization function.

8.6.1 The normalization function

At this point, it becomes necessary to shift from working purely internally to $\mathcal{G}$ to inspecting some constructions externally. Accordingly, we will have use for the *total* spaces of terms and normal forms e.g. $\mathsf{Tm}^{*}_{m} = \sum_{A:\mathsf{Ty}^{*}_{m}} \mathsf{Tm}^{*}_{m}(A)$. We write $\mathcal{T}_m$ and $\mathcal{T}_m$ for the presheaves of types and terms in $\mathcal{S}(m)$ to disambiguate them from Ty^{*}_{m} and Tm^{*}_{m}.

Lemma 8.6.1. *There is a morphism* $\downarrow : \mathsf{Tm}^{*}_{m} \longrightarrow \mathsf{Nf}_{m}$ *which restricts to* id *under* $\mathbf{syn}$.

Proof. Working internally, $\downarrow(A, M) = (A, \downarrow_A M)$. □

Fix a term $\Gamma \vdash M : A$. Theorems 8.3.5 and 8.5.11 define a map $[\![M]\!] : [\![\Gamma]\!] \longrightarrow \mathsf{Tm}^{*}_{m}$ in $\mathcal{G}(m)$ along with an isomorphism $\alpha : \pi_0([\![\Gamma]\!]) \cong \mathbf{y}(\Gamma)$ such that $\pi_0([\![M]\!]) = \lfloor M \rfloor \circ \alpha$.

We would like to obtain a normal form for M from $[\![M]\!]$. To this end, we can unfold $[\![M]\!]$ along with $\downarrow$ from Lemma 8.6.1 to obtain a commuting diagram:

$$\begin{tikzcd}
\pi_1([\![\Gamma]\!]) \arrow[rr] \arrow[d, "\alpha \circ [\![\Gamma]\!]"'] & & \pi_1(\mathsf{Tm}^{*}_{m}) \arrow[r] \arrow[d] & \pi_1(\mathsf{Nf}_{m}) \arrow[dl] \\
\mathbf{i}[m]^{*}(\mathbf{y}(\Gamma)) \arrow[rr, "\mathbf{i}[m]^{*}(\lfloor M \rfloor)"'] & & \mathbf{i}[m]^{*}(\mathcal{T}_m) &
\end{tikzcd}$$

To normalize M, it suffice to construct $\mathsf{atoms}_{\Gamma} : \pi_1([\![\Gamma]\!])_{\Gamma}$ such that $\alpha([\![\Gamma]\!](\mathsf{atoms}_{\Gamma})) = \mathsf{id} : \mathbf{i}[m]^{*}(\mathbf{y}(\Gamma))_{\Gamma}$: pushing atoms_{Γ} along the top of the diagram would yield a normal form (an element of $\pi_1(\mathsf{Nf}_{m})$) which decodes to M by Yoneda.

Lemma 8.6.2. *For any* $\vdash \Gamma\ \mathsf{cx}$ *there exists* $\mathsf{atoms}_{\Gamma} : \pi_1([\![\Gamma]\!])_{\Gamma}$ *lying over* $\mathsf{id} : \mathbf{i}[m]^{*}(\mathbf{y}(\Gamma))$.

Proof. This proof proceeds by induction on Γ.

Case $\Gamma = \mathbf{1}$

Here $[\![\Gamma]\!]$ is terminal, so $\mathsf{atoms}_{\mathbf{1}}$ is its unique element.

Case $\Gamma = \Delta.(\mu \mid A)$

In this case $[\![\Gamma]\!] = [\![\Delta]\!] \times_{\mathcal{G}(\mu)(\mathsf{Ty}^*_n)} \mathcal{G}(\mu)(\mathsf{Tm}^*_n)$. First, we reindex atoms_Δ by $\Gamma \vdash \mathbf{p} : \Delta$ to obtain $\delta \in [\![\Delta]\!]_\Gamma$. Next, using the element $\mathbf{v} \in \mathcal{G}(\mu)(\mathsf{Ne}_n(A))_\Gamma$ we define $\mathsf{atoms}_\Gamma = (\delta, \uparrow_A \mathbf{v})$.

Case $\Gamma = \Delta.\{\mu\}$

We define $\mathsf{atoms}_\Gamma = \mathcal{G}(\mu)_!(\mathsf{atoms}_\Delta)$

□

Remark 8.6.3. atoms_Γ is analogous to the initial environment used in classical NbE proofs to kick off normalization. Abel [Abe13], for instance, denotes the environment $\uparrow^\Gamma$. ◇

Combining Lemma 8.6.2 with the argument above, we conclude that for term $\Gamma \vdash M : A$, there exists $\Gamma \vdash^{\mathsf{nf}} u : A \mathbin{@} m$ such that $|u| = M$. Moreover, because we have consistently worked with equivalences class of terms, this function automatically respects definitional equality. Summarizing:

Theorem 8.6.4. *There is a function* $\mathbf{nf}_\Gamma(-, A)$ *sending terms of type* $\Gamma \vdash A\,\mathsf{type}$ *to normal forms such that*

1. *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash |\mathbf{nf}_\Gamma(M, A)| = M : A$.
2. *If* $\Gamma \vdash M = N : A$ *then* $\mathbf{nf}_\Gamma(M, A) = \mathbf{nf}_\Gamma(N, A)$.

We can repeat this process to normalize types instead of terms. Given $\Gamma \vdash A\,\mathsf{type}$, we obtain $[\![A]\!] : [\![\Gamma]\!] \longrightarrow \mathsf{Ty}^*_m$ which unfolds to an analogous diagram with only a small change: rather than using $\uparrow$ to pass from $\pi_1(\mathsf{Tm}^*_m)$ to normal forms, we use code to shift from Ty^*_m to normal types:

$$\begin{tikzcd}
\pi_1([\![\Gamma]\!]) \arrow[r] \arrow[d, "\alpha \circ [\![\Gamma]\!]"'] & \pi_1(\mathsf{Ty}^*_m) \arrow[r] \arrow[d] & \pi_1(\mathsf{NfTy}_m) \arrow[dl] \\
\mathbf{i}[m]^*(\mathbf{y}(\Gamma)) \arrow[r, "\mathbf{i}[m]^*(\lfloor A \rfloor)"'] & \mathbf{i}[m]^*(\mathcal{T}_m) &
\end{tikzcd}$$

By again pushing atoms_Γ along the top of this diagram, we obtain a normalization function for types.

Theorem 8.6.5. *There is a function* $\mathbf{nfty}_\Gamma(-)$ *sending types to normal types such that*

1. *If* $\Gamma \vdash A\,\mathsf{type}$ *then* $\Gamma \vdash |\mathbf{nfty}_\Gamma(A)| = A\,\mathsf{type}$.
2. *If* $\Gamma \vdash A = B\,\mathsf{type}$ *then* $\mathbf{nfty}_\Gamma(A) = \mathbf{nfty}_\Gamma(B)$.

8.6.2 Corollaries of normalization

A number of important theorems follow as corollaries of Theorems 8.6.4 and 8.6.5. For instance, we can reduce the decidability of conversion to the decidability of the mode theory.

Corollary 8.6.6 (Decidability of conversion)**.**

1. $\Gamma \vdash M = N : A$ *iff* $\mathbf{nf}_\Gamma(M, A) = \mathbf{nf}_\Gamma(N, A)$.
2. $\Gamma \vdash A = B\,\mathsf{type}$ *iff* $\mathbf{nfty}_\Gamma(A) = \mathbf{nfty}_\Gamma(B)$.

Proof. We show only the proof for this first claim. The 'only if' direction is established by the second point of Theorem 8.6.4. Suppose instead $\mathbf{nf}_\Gamma(M, A) = \mathbf{nf}_\Gamma(N, A)$, so $|\mathbf{nf}_\Gamma(M, A)| = |\mathbf{nf}_\Gamma(N, A)|$. By the first point of Theorem 8.6.4, $|\mathbf{nf}_\Gamma(M, A)| = M$ and $|\mathbf{nf}_\Gamma(M, A)| = N$, so the conclusion follows. □

Equality of normal forms and normal types is evidently decidable if equality in $\mathcal{M}$ is decidable, so this proves the promised sharp bound on the decidability of conversion in MTT. While we have not developed a bidirectional syntax for MTT, the fully annotated presentation of its syntax is decidable precisely when conversion is decidable:

Corollary 8.6.7. *If $\mathcal{M}$ is decidable, type checking is decidable.*

A priori, however, a given term could have multiple normal forms which complicates further analysis. We therefore strengthen Theorem 8.6.4 with the following:

Theorem 8.6.8 (Tightness)**.**

1. *If $\Gamma \vdash^{\mathsf{nf}} u : A \mathbin{@} m$, then $\mathbf{nf}_\Gamma(|u|, A) = u$.*
2. *If $\Gamma \vdash^{\mathsf{nf}} \tau \mathbin{@} m$, then $\mathbf{nfty}_\Gamma(|\tau|) = \tau$.*

Proof. Recall that Theorems 8.3.5 and 8.5.11 induce a function $[\![-]\!]$ sending a piece of syntax to its interpretation in the normalization model. Furthermore, recall the $(\!|\Theta|\!)$-element $\mathsf{atoms}_\Theta : [\![\Gamma]\!]$ constructed in Lemma 8.6.2.

We begin by strengthening the statement to make it more amenable to induction:

1. If $\Gamma \vdash^{\mathsf{ne}} e : A \mathbin{@} m$, then $[\![|M|]\!](\mathsf{atoms}_\Theta) = \uparrow_{[\![A]\!](\mathsf{atoms}_\Theta)} e$
2. If $\Gamma \vdash^{\mathsf{nf}} u : A \mathbin{@} m$, then $\uparrow_{[\![A]\!](\mathsf{atoms}_\Theta)} [\![|u|]\!](\mathsf{atoms}_\Theta) = u$.
3. If $\Gamma \vdash^{\mathsf{nf}} \tau \mathbin{@} m$, then $\mathsf{code}[\![|A|]\!](\mathsf{atoms}_\Theta) = \tau$.

Here we have identified a code u (resp. e) as an $(\!|\Theta|\!)$ element of Nf_A (resp. Ne_A). All three follow straightforwardly from mutual induction and the relevant definitions. □

Corollary 8.6.9. *Normalization is an isomorphism between equivalence classes of terms (resp. types) and normal forms (resp. normal types).*

Proof. Corollary 8.6.6 already shows that normalization is injective and Theorem 8.6.8 provides a section. □

These results imply the injectivity of type constructors, an essential property for implementation.

Corollary 8.6.10. *If* $\Gamma \vdash A_0 \to B_0 = A_1 \to B_1\,\mathsf{type}$ *then* $\Gamma \vdash A_0 = A_1\,\mathsf{type}$ *and* $\Gamma.(\mathsf{id} \mid A_0) \vdash B_0 = B_1\,\mathsf{type}$.

Proof. Set $\tau_i = \mathbf{nfty}_\Gamma(A_i)$ and $\sigma_i = \mathbf{nfty}_{\Gamma.(\mathsf{id}|A_0)}(B_i)$. Unfolding definitions shows that $|\tau_i \to \sigma_i| = |\tau_i| \to |\sigma_i| = A_i \to B_i$. By Corollary 8.6.9, $\mathbf{nfty}_\Gamma(A_i \to B_i) = \tau_i \to \sigma_i$.

Next, we recall that $\Gamma \vdash A_0 \to B_0 = A_1 \to B_1\,\mathsf{type}$ by assumption, so $\tau_0 \to \sigma_0 = \tau_1 \to \sigma_1$. As an operation on normal forms, however, $- \to -$ is clearly injective, so $\tau_0 = \tau_1$ and $\sigma_0 = \sigma_1$. The result now follows from Corollary 8.6.6. □

Gratzer et al. [Gra+20a] show canonicity for MTT extended with the equality $\mathbf{1}.\{\mu\} = \mathbf{1}$. Normalization provides a (heavy-handed) proof of canonicity without this equation by scrutinizing the definition of normal forms:

Corollary 8.6.11. *If* $\mathbf{1}.\{\mu\} \vdash M : \mathsf{Bool}$ *then* $M \in \{\mathsf{tt}, \mathsf{ff}\}$.

Finally, we are now in a position to show the independence of strict dependent right adjoints from ordinary MTT.

Theorem 8.6.12. *In MTT with the adjoint mode theory (Example 6.1.6), the right adjoint is not a strict dependent right adjoint.*

Proof. The argument is similar to the one used in Corollary 8.6.10. Let us write μ for the internal right adjoint and suppose that $\langle \mu \mid - \rangle$ was a strict dependent right adjoint. In this case, there would exist a term $\Gamma.(\mathsf{id} \mid \langle \mu \mid A \rangle).\{\mu\} \vdash \mathsf{unmod}(\mathbf{v}) : A[\mathbf{p}.\{\mu\}]$ such that $\mathsf{mod}_\mu(\mathsf{unmod}(\mathbf{v})) = \mathbf{v}$; the existence of $\mathsf{unmod}(\mathbf{v})$ is one formulation of the stronger elimination rule.

Normalization shows that no such term can be defined in MTT; we may compute the normal forms of the left- and right-hand sides of the equation $\mathsf{mod}_\mu(\mathsf{unmod}(\mathbf{v})) = \mathbf{v}$ and note that the left-hand side must begin $\mathsf{mod}_\mu(\dots)$ while the right-hand side will be $\mathsf{up}(\mathbf{v}_0^{\mathsf{id}})$. The completeness of normalization implies that $\mathsf{mod}_\mu(\mathsf{unmod}(\mathbf{v}))$ can never be equal to $\mathbf{v}$ which contradicts our assumptions on $\mathsf{unmod}(-)$. □

8.7 Normalization in the presence of extensions

To demonstrate the flexibility of the normalization argument given in Sections 8.5 and 8.6, we now show how it may be extended to accommodate modal principles not included in MTT discussed in Section 6.3.

8.7.1 Crisp identity induction principles

The modularity of our proof of normalization ensures that only local changes to the construction of identity types in $\mathcal{G}$ are needed to adapt the entire proof to support crisp induction. Concretely, two changes to primitive constants added to MSTC by Section 8.5.1. One alteration to the definition of cosmoi and one to the definition of neutral forms:

$$\mathsf{J}_\mu : (A : (\mu \mid \mathsf{Ty}_n))\,(B : (a_0, a_1 : (\mu \mid \mathsf{Tm}_n(A)))(p : (\mu \mid \mathsf{Tm}_n(\mathsf{Id}(A, a_0, a_1)))) \to \mathsf{Ty}_m)$$

$$
\begin{aligned}
&\quad \to ((a : (\mu \mid \mathsf{Tm}_n(A))) \to \mathsf{Tm}_m(B(a, a, \mathsf{refl}(a))))\\
&\quad \to (a_0, a_1 : (\mu \mid \mathsf{Tm}_n(A)))(p : (\mu \mid \mathsf{Tm}_n(\mathsf{Id}(A, a_0, a_1))))\\
&\quad \to \mathsf{Tm}_m(B(a_0, a_1, p))\\
&\mathbf{J}_\mu : (A : (\mu \mid \bigcirc\mathsf{Ty}_n))\,(B : (a_0, a_1 : (\mu \mid \mathsf{V}_n(A)))(p : (\mu \mid \mathsf{V}_m(\mathsf{Id}(A, a_0, a_1)))) \to \mathsf{NfTy}_m)\\
&\quad \to ((a : (\mu \mid \mathsf{V}_n(A))) \to \mathsf{Nf}_m(B(a, a, \mathsf{refl}(a))))\\
&\quad \to (a_0, a_1 : (\mu \mid \bigcirc_z \mathsf{Tm}_n(z, A(z))))(p : (\mu \mid \mathsf{Ne}_n(\mathsf{Id}(A, a_0, a_1))))\\
&\quad \to \mathsf{Ne}_m(B(a_0, a_1, \eta(p)))
\end{aligned}
$$

These changes simply reflect the change to the elimination principle of the identity type.

After having made this change, only one portion of Section 8.5.2 must change: Lemma 8.5.9 which shows that the gluing cosmos is closed under identity types. We must show that $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ is closed under crisp induction.

Lemma 8.7.1. $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ *supports crisp identity induction.*

Proof. This argument is similar to Lemma 8.5.6, as the induction principle for modal types is always 'crisp' in MTT. We must implement the following constant.

$$
\begin{aligned}
&\mathsf{J}^*_\mu : (A : (\mu \mid \mathsf{Ty}^*_n))\,(B : (a_0, a_1 : (\mu \mid \mathsf{Tm}^*_n(A)))(p : (\mu \mid \mathsf{Tm}^*_n(\mathsf{Id}^*(A, a_0, a_1)))) \to \mathsf{Ty}^*_m)\\
&\quad \to (b : (a : (\mu \mid \mathsf{Tm}^*_n(A))) \to \mathsf{Tm}^*_m(B(a, a, \mathsf{refl}^*(a))))\\
&\quad \to (a_0, a_1 : (\mu \mid \mathsf{Tm}^*_n(A)))(p : (\mu \mid \mathsf{Tm}^*_n(\mathsf{Id}(A, a_0, a_1)))) \to\\
&\quad \to \{\mathsf{Tm}^*_m(B(a_0, a_1, p)) \mid z : \mathbf{syn} \mapsto \mathsf{J}_\mu(A, B, b, p)\}
\end{aligned}
$$

Let us fix A, B, b, a_0, a_1, and p with the types described above. Recalling the definition of $\mathsf{Id}^*(A, a_0, a_1).\mathsf{pred}$ from Lemma 8.5.9, we can commute $\langle \mu \mid - \rangle$ past the dependent sum, closed modalities, equality types, and coproducts to decompose p into a pair of the following:

$$
\begin{aligned}
&(\mathsf{tm} : (\mu \mid \mathsf{Nf}_n(\mathsf{Id}(A, a_0, a_1))))\\
&\mathsf{prf} : \bullet \begin{bmatrix} \sum_{e : \langle \mu \mid \mathsf{Ne}_n(\mathsf{Id}(A, a_0, a_1)) \rangle} \mathbf{up} \circledast e = \mathsf{mod}_\mu(m) \\ + \,\mathsf{mod}_\mu(m) = \mathsf{mod}_\mu(\mathsf{refl}(a_0)) \end{bmatrix}
\end{aligned}
$$

We then define $\mathsf{J}^*_\mu(B, b, a_0, a_1, p)$ by analyzing prf:

$$
\begin{cases}
\mathsf{J}(z, B, b, a_0, a_1, p) & \mathsf{prf} = \mathsf{in}_1(z)\\
\downarrow\mathbf{J}(\lambda a_0, a_1, p.\ B(\uparrow a_0, \uparrow a_1, \uparrow p).\mathsf{code}, \lambda a.\ \downarrow b(\uparrow a), e) & q = \mathsf{in}_2(\mathsf{in}_1(e, \_))\\
b(a_0) & q = \mathsf{in}_2(\mathsf{in}_2(\_))
\end{cases}
$$

□

Having made this alteration, the remainder of Section 8.5 and Section 8.6 are unchanged. In particular, all the results of Section 8.6 continue to hold in the presence of crisp induction.

8.7.2 Strict dependent right adjoints

For this subsection, suppose we are given an internal right adjoint $\mu : n \longrightarrow m \in \mathcal{M}$ i.e. we suppose that there exists a modality $\nu : m \longrightarrow n$ together with 2-cells $\eta : \mathsf{id} \longrightarrow \mu \circ \nu$ and $\epsilon : \nu \circ \mu \longrightarrow \mathsf{id}$ satisfying the triangle identity. In Section 6.3, we noted that the existence of the left adjoint enables us to present a modal type for μ with an η law without sacrificing good syntactic properties:

$$\frac{\vdash \Gamma\ \mathsf{cx} \ @\ m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type}\ @\ n}{\Gamma \vdash \mu \Rightarrow A\ \mathsf{type}\ @\ m}$$

$$\frac{\vdash \Gamma\ \mathsf{cx}\ @\ m \qquad \Gamma.\{\mu\} \vdash A\ \mathsf{type}\ @\ n \qquad \Gamma.\{\mu\} \vdash M : A\ @\ n}{\Gamma \vdash \{M\}_{\mu} : \mu \Rightarrow A\ @\ m}$$

$$\frac{\vdash \Gamma\ \mathsf{cx}\ @\ n \qquad \Gamma.\{\nu \circ \mu\} \vdash A\ \mathsf{type}\ @\ n \qquad \Gamma.\{\nu\} \vdash M : \langle \mu \mid A \rangle\ @\ m}{\Gamma \vdash\ !_{\mu}\, M : A[\Gamma.\{\epsilon\}]\ @\ n}$$

We now show that the normalization proof can extend to support these strict modal types while retaining all the corollaries derived in Section 8.6.

Just as with crisp identity induction principles, the changes are essentially local. We must extend normal and neutral forms to account for this new connective, specify how $\mu \Rightarrow -$ is represented internally in a presheaf cosmos and then show that the normalization cosmos can be extended to support these types. In fact, all of these tasks are *easier* than the corresponding exercises for standard modal types owing to the presence of the η law.

The following constants suffice to represent $\mu \Rightarrow -$ internally:

$$\begin{aligned}&\mathsf{SMod}_{\mu} : (\mu \mid \mathsf{Ty}_n) \to \mathsf{Ty}_m\\ &\alpha_{\mathsf{SMod}_{\mu}} : (A : (\mu \mid \mathsf{Ty}_n)) \to \langle \mu \mid \mathsf{Tm}_n(A) \rangle \cong \mathsf{Tm}_m(\mathsf{SMod}_{\mu}(A))\end{aligned}$$

In particular, the presence of an η law ensures that the introduction, elimination, β and η laws can all be bundled up into a single isomorphism. The constants encoding normal and neutral forms for this type former are internalized as follows:

$$\begin{aligned}&\mathbf{SMod}_{\mu} : (\mu \mid \mathsf{NfTy}_n) \to \mathsf{NfTy}_m\\ &\mathbf{smod}_{\mu} : (A : (\mu \mid \bigcirc\mathsf{Ty}_n))(a : (\mu \mid \mathsf{Nf}_n(A))) \to \{\mathsf{Nf}_m(\mathsf{SMod}_{\mu}(A)) \mid z : \mathbf{syn} \mapsto \alpha_{\mathsf{SMod}_{\mu}}(a)\}\\ &\mathbf{unmod}_{\mu} : (A : (\mu \mid \bigcirc\mathsf{Ty}_n))(e : \mathsf{Ne}_m(\mathsf{SMod}_{\mu}(A))) \to \{\mathsf{Ne}_n(A) \mid z : \mathbf{syn} \mapsto \alpha^{-1}_{\mathsf{SMod}_{\mu}}(e)\}\end{aligned}$$

It remains only to adapt the normalization cosmos to include this additional type.

Lemma 8.7.2. $(\mathsf{Ty}^*_m, \mathsf{Tm}^*_m)$ *supports a strict dependent right adjoint type for* μ.

Proof. We must construct the following constants:

$$\begin{aligned}&\mathsf{SMod}^*_{\mu} : (\mu \mid \mathsf{Ty}^*_n) \to \mathsf{Ty}^*_m\\ &\alpha_{\mathsf{SMod}^*_{\mu}} : (A : (\mu \mid \mathsf{Ty}^*_n)) \to \langle \mu \mid \mathsf{Tm}^*_n(A) \rangle \cong \mathsf{Tm}^*_m(\mathsf{SMod}^*_{\mu}(A))\end{aligned}$$

We must further arrange that $\mathsf{SMod}^*_{\mu} = \mathsf{SMod}_{\mu}(z)$ and $\alpha_{\mathsf{SMod}^*_{\mu}} = \alpha_{\mathsf{SMod}_{\mu}}(z)$ after assuming $z : \mathbf{syn}$.

Assuming $z : \mathbf{syn}$, we have the following:

$$\alpha_{\mathsf{SMod}_\mu} : \langle \mu \mid \mathsf{Tm}^*_n(A) \rangle = \langle \mu \mid \mathsf{Tm}_n(A) \rangle \cong \mathsf{Tm}_m(z, \mathsf{SMod}_\mu(A))$$

Accordingly, we realign along $\alpha_{\mathsf{SMod}_\mu}$ to obtain Φ along with the following isomorphism:

$$\alpha_{\mathsf{SMod}^*_\mu} : \{ \langle \mu \mid \mathsf{Tm}^*_n(A) \rangle \cong \Phi \mid z : \mathbf{syn} \mapsto \alpha_{\mathsf{SMod}_\mu}(z) \}$$

It remains to parlay Φ to SMod^*_μ:

$$\begin{aligned}
&\mathsf{SMod}^*_\mu(A).\mathsf{code} = \mathbf{SMod}_\mu(A.\mathsf{code}) \\
&\mathsf{SMod}^*_\mu(A).\mathsf{pred} = \Phi \\
&\mathsf{SMod}^*_\mu(A).\!\uparrow\, = \mathbf{unmod}_\mu(A) \\
&\mathsf{SMod}^*_\mu(A).\!\downarrow\, = \lambda a.\ \mathbf{smod}_\mu(A, \downarrow_A(a))
\end{aligned}$$

Familiar calculations ensure that all relevant boundary conditions are satisfied. □

Part III

Applications of multimodal type theory

9 Guarded recursion

One good example is worth a host of generalities

Martin Hyland
The Effective Topos

In this chapter, we investigate various formulations of guarded recursion within MTT. While it was briefly touched upon in Section 5.1, we begin with a description and motivation of guarded recursion broadly.

9.1 Introduction

The original motivation for guarded recursion is relatively concrete [Nak00]. Suppose one is designing a programming language with infinite data but where programs that loop forever without producing output are forbidden. To ensure that recursive definitions do not introduce such divergent programs, we must ensure that each definition is productive; it should be the case that producing a finite prefix of the output stream takes only a finite amount of time. For instance, consider the following programs computing infinite streams:

$$\begin{aligned}&\mathsf{ones} = \mathsf{cons}(1, \mathsf{ones})\\ &\mathsf{uhoh} = \mathbf{let}\ (h, t) = \mathsf{uhoh}\ \mathbf{in}\ (h + 1, t)\end{aligned}$$

The first definition defines an infinite stream, but any finite prefix of this infinite stream can be obtained after unfolding ones a finite number of times. The second definition is different: we are not even able to produce the first element of uhoh.

In these examples there are syntactic checks one could impose to rule out uhoh while permitting ones. Indeed, syntactic checks of this form are presently implemented in systems like Coq or Agda to ensure the soundness of their coinductive mechanisms. By their nature, however, syntactic checks are brittle. Changing a program—even while ensuring that it remains the same up to definitional equality—can cause an algorithm to go from passing to failing a syntactic check.

Guarded recursion promises to reduce the syntactic check to a type-checking problem. Rather than providing unrestricted recursive definitions and attempting to check after the fact that a definition was well-formed, guarded-recursive systems limit the recursion that is available from the start. They add a new type former $\blacktriangleright -$ along with the following principle:

$$\mathsf{loeb} : (\blacktriangleright A \to A) \to A$$

This combinator mirrors the standard fixed-point combinator but with the recursive argument replaced by $\blacktriangleright A$. Intuitively, $\blacktriangleright A$ classifies computations which will eventually compute to A but which are only safe to access after some computation has taken place. This intuition is codified by shaping $\blacktriangleright$ into an applicative functor [MP08] and inserting it into the types of constructs like $\mathsf{cons} : A \to \blacktriangleright\mathsf{Str}(A) \to \mathsf{Str}(A)$. Indeed, we can see that while ones can be elaborated to a definition using loeb, the same is not true of uhoh.

Guarded domain theory While coinductive definitions are certainly useful, much of the focus on guarded recursion has come from its use in the theory of programming languages. Analysis of programming languages with features like recursive definitions or general store is fraught with recursive equations, some of which cannot be handled by traditional domain-theoretic technology.

In the context logical relations, for instance, these recursive equations are handled through *step-indexing* [Ahm04; AM01]. While originally step-indexing was motivated very operationally—it ensured that certain constructions were well-defined—later work isolated a certain logical substrate of the technique by connecting it with guarded recursion [DAB11] and the canonical model of guarded recursion $\mathbf{PSh}(\omega)$ [Bir+12].

Remark 9.1.1. For those familiar with step-indexed logical relations but not guarded recursion, we provide a sketch of the connection. The key idea of step-indexed logical relations is to avoid defining a single relation for a type $[\![\tau]\!]$ but to instead define a family of relations indexed by the natural numbers $[\![\tau]\!]_n$. In particular, one defines $[\![\tau]\!]_{n+1}$ using the definition of $[\![\tau]\!]_n$ so that one may simulate complicated recursive definitions.

Dreyer et al. [DAB11] observed that it was not necessary to explicitly carry through the actual n; rather, it suffices to implicitly thread the number through all constructions and to extend the language with a connective allowing one to express "P_{n-1} if $n > 0$ or $\top$ if $n = 0$". Implicitly threading n through a logical expression is the domain of the Kripke-Joyal interpretation of logic into $\mathbf{PSh}(\omega)$ and the novel connective is $\blacktriangleright$. Finally, the ability to define $[\![\tau]\!]_n$ by induction on n follows immediately from loeb. In other words, the logical aspect of step-indexed logical relations is precisely the logic of $\mathbf{PSh}(\omega)$. ◇

Recasting step-indexed logical relations in this way has proven to be a highly successful idea and has led to a cottage industry of logical relations and program logics based upon guarded recursion, for instance [DNB12; Jun+16; Jun+18; Jun+15; TDB13; Tur+13]. However, it was apparent already in Birkedal et al. [Bir+12] that it would be useful to have not merely a logic for guarded recursion, but a full dependent type theory.

For an example of where such a dependent type theory might be useful consider the guarded program logic Iris. In Iris, like all such program logics, it is often necessary to define data through guarded recursion instead of merely propositions; the core notion of proposition must be defined simultaneously with a highly recursive notion of *resource* to support the impredicativity crucial to the system. When working only with guarded logic, these proof-relevant constructions must be carried out through an entirely separate mechanism and, more concerningly, this apparatus is the most complex part of the Coq formalization of Iris.

With a full dependent type theory, it becomes possible to give denotational semantics in a similar style to step-indexed logical relations, but without limiting semantic types to predicates [BBM14; Biz+16; MP16; MV19; MV21; PMB15; VV20]. This style of semantics has been referred to as *synthetic guarded domain theory* as it replaces traditional domains with a type X equipped with a map $\blacktriangleright X \to X$.

Finally, a guarded dependent type theory offers a chance for unification. It enables a programmer to take advantage of the productive recursive definitions that originally motivated Nakano [Nak00] in the same system that a mathematician may construct Iris.

Guarded dependent type theory With motivation coming from several different directions, it should be no surprise that several different guarded dependent type theories have been proposed. The vast majority of these type theories are extensions of extensional MLTT and as such were created without implementation considerations in mind—they were designed to serve as pen-and-paper metalanguages for informal guarded reasoning.

One such system was proposed already in Birkedal et al. [Bir+12, Section 4]. It includes a fibered $\blacktriangleright$ modality along with the operations of an applicative functor and Löb induction. This type theory also includes a somewhat technical excursion to allow for *types* to be defined via a principle akin to loeb induction. It was quickly recognized [BM13] that this addition could be replaced by plain loeb induction and a *dependent* version of $\blacktriangleright$ given the more refined type $\blacktriangleright\mathcal{U} \to \mathcal{U}$. Working directly with the applicative functor operations proved cumbersome in general, and they were replaced by a more sophisticated notion of *delayed substitution* and accompanying equations which was better suited for reasoning about complex guarded types [Biz+16; Clo+15]. The syntax and equational theory around delayed substitutions was quite complex and sparked a search for a suitable replacement while retaining the same flexibility and concision they offered.[1]

The development of delayed substitutions happened more-or-less simultaneously with the introduction of *clocks* into dependent type theory. Roughly, guarded type theory captures programs that compute a stream of information measured by a single time stream.[2] The $\blacktriangleright$ modality allows us to describe computation available one step in the future and Löb induction allows us to perform induction on the time remaining. Clocks generalize the situation by allowing for multiple independent streams of time and, therefore, multiple independent $\blacktriangleright$ modalities. Concretely, clocked type theories add a new sort of variable (clock variables) and each such variable indexes a separate time stream. Consequently, streams can be bound and manipulated. Originally introduced in Atkey and McBride [AM13], the primary draw of clocks was a crucial increase in expressivity; the ability to bind a clocked variable allowed one to express a computation which could be examined for arbitrarily many steps *internally to the type theory.* While this may appear like a somewhat arcane goal, it has immediate consequences. For instance, with the type of (guarded) streams described above, there is no way to describe a function $\mathsf{tail} : \mathsf{Str}(A) \to \mathsf{Str}(A)$. Instead, one can only define $\mathsf{tail} : \mathsf{Str}(A) \to \blacktriangleright\mathsf{Str}(A)$ and this version of tail is insufficient to e.g., define a map between streams dropping every other element. With the addition of clocks [Møg14], guarded type theory could therefore express not only guarded operations on streams, but define and reason about arbitrary coinductive types [Biz+16].

[1] See Section 5.3 for an extended discussion of the trade-offs of delayed substitutions.

[2] Returning to the early connection with step-indexing, all definitions are indexed by a single natural number.

After this rapid development from 2011 through 2017, two main issues remained. The first was the implementability of guarded type theory. Even if one removed extensional equality from these systems, the delayed substitutions and Löb induction which were so convenient for reasoning on paper obstructed the decidability of conversion and therefore type-checking. Moreover, while clocks gave a way to reason about global behavior in a syntactically convenient way, their semantics were far more complicated than $\mathbf{PSh}(\omega)$ [BM15; BM20]. Some exploration of using the global sections comonad on $\mathbf{PSh}(\omega)$ by Clouston et al. [Clo+15] showed that many of the applications of clocks could be recovered in this simpler semantics but the syntax of the global sections comonad (particularly in conjunction with $\blacktriangleright$) is highly non-trivial.

Clocked type theory The first complete guarded type theory CloTT (Clocked Type Theory) to include all the features necessary for synthetic guarded domain theory while remaining implementable was proposed by Bahr et al. [BGM17]. Summarizing, CloTT included the following:

- A full dependent type theory.
- A version of the $\blacktriangleright$ modality with a novel syntax of ticks for its manipulation and no delayed substitutions. [3]
- Full integration of clocks with clock quantification properly interacting with $\blacktriangleright$.
- A version of loeb which limits unfolding to enjoy normalization.

To elaborate on the last point, loeb induction in CloTT is restricted so that the expected computation rule holds only when used "at the top-level". The precise mechanics of this rule are slightly more complex and center around the interaction between clock quantification and $\blacktriangleright$ but the salient point is that one cannot rely only on definitional equality when dealing with loeb. While the first approach to CloTT did not fully address this gap, later iterations built upon a marriage of guarded type theory and cubical type theory [Bir+19] to add a *path* unfolding loeb induction in all cases [KMV22]. The earlier restriction on computation is still present: the path only degenerates to reflexivity in the "top-level" cases. The resulting system is undoubtedly complex, but it marries together cubical type theory and guarded type theory in a way that conjecturally still satisfies decidable type-checking. Moreover, CloTT has been implemented in Agda and used in several case studies [MV21; VV20].

Guarded MTT While CloTT has proven to be a highly versatile system, we choose to revisit the foundations of guarded type theory once more in search of simpler semantics. Indeed, the semantics of (cubical) CloTT are substantially more complex than the already difficult models of clocked type theory [KMV22; MMV20]. Given all that the system includes, this is to be expected. If, however, one is merely interested in providing an implementable metalanguage for synthetic guarded domain theory, the full power of clocks and cubical type theory seems unnecessary.

We note that both of these additions made a great deal of sense when extending type theory with a global sections comonad was just as costly, but the purpose of Part II

[3] This latter point was seen as so crucial that it found its way into the title of the paper.

was to demonstrate that this could be done without significant complications. Indeed, the work done in Chapters 6 to 8 shows that one can obtain a type theory with $\square$ and $\blacktriangleright$ correctly interacting without any further work.[4] In fact, in Section 9.3 we show that there is a whole family of mode theories one can instantiate MTT with to capture different aspects of guarded type theory in $\mathbf{PSh}(\omega)$ and generalizations thereof.

Unfortunately, $\square$ and $\blacktriangleright$ doth not a guarded type theory make. We shall see that introducing Löb induction is a complex task and seemingly cannot be done without some cost. In fact, there are two canonical ways to extended MTT with $\square$ and $\blacktriangleright$ to a full guarded type theory:

1. Lean into undecidable conversion and work with extensional MTT with $\square$, $\blacktriangleright$, and an axiomatized loeb.

2. Work in standard MTT and postulate certain crisp induction principles along with loeb and an explicit unfolding term.

Neither approach is fully satisfying: the first approach does not enjoy decidable type-checking while the second system does not satisfy canonicity. Both of type theories, however, are not without merit. We shall see in Section 9.4 that extensional guarded MTT can be seen as a completion of the program started by Clouston et al. [Clo+15]. It can be used to reason about synthetic guarded domain theory and coinductive types within the same language and with minimal syntactic burden. On the other hand, in Section 9.6 we will show that working with a version of loeb is not only theoretically possible but a practical reality.

Finally, we show how to merge these two systems into a *stratified* system for guarded recursion. In such a system, programs are type-checked using the intensional system and run a system where loeb unfolds definitionally. We prove that the latter satisfies a *guarded* version of canonicity. These two results form the basis for an alternative strategy for implementing guarded recursion: program in a “static” system and run programs in a “dynamic” system.

In this chapter, we explore the space of trade-offs available for guarded MTT. In Sections 9.2 and 9.3, we describe the intended models and typical mode theories for guarded MTT. These give an overview of the relatively uncontroversial semantic side of guarded recursion. In Section 9.4 we introduce guarded extensional MTT and carry out several case studies to show its utility—at least when working on paper. In Section 9.5 we state and prove a no-go theorem showing that in the presence of Löb induction, extensional guarded MTT is the best one can do. Finally, in Section 9.6 we show how one may modify and restrict Löb induction to obtain a system which enjoys normalization (but not canonicity) and can be *compiled* to one satisfying canonicity (but not normalization). This stratified fragment splits the difference between the two approaches described above to yield a reasonable candidate for an implementable guarded type theory.

9.2 Presheaf models of guarded recursion

Birkedal et al. [Bir+12] introduced the canonical model for guarded domain theory and guarded dependent type theory: the topos of trees $\mathbf{PSh}(\omega)$. The appeal of this model is

[4] Of course, this is not a coincidence; this situation was a motivating example for MTT.

its simplicity: it is a presheaf category whose base category is a poset. This makes it extremely amenable to calculation. The challenge of guarded type theory is to construct a well-behaved internal language that captures the salient features of $\mathbf{PSh}(\omega)$. In this section, we review those features.

To begin with, $\mathbf{PSh}(\omega)$ is a topos and therefore contains a rich internal type theory with all the features one could desire: dependent products, an impredicative universe of propositions, a hierarchy of universes. What distinguishes $\mathbf{PSh}(\omega)$ from other Grothendieck topoi is a collection of endofunctors on $\mathbf{PSh}(\omega)$ which we use to capture guarded recursion.

We begin with the realization of the $\blacktriangleright$ modality as a functor on $\mathbf{PSh}(\omega)$.

Lemma 9.2.1. *The* $\blacktriangleright$ *functor on* $\mathbf{PSh}(\omega)$ *is part of an infinite chain of adjoints:* $\ldots \dashv \blacktriangleleft \dashv \blacktriangleright$.

Proof. We define the $\blacktriangleright$ modality as follows:

$$(\blacktriangleright X)(n) = \begin{cases} \mathbf{1} & n = 0 \\ X(m) & n = m + 1 \end{cases}$$

For purely formal reasons, this modality is a right adjoint: it is accessible and commutes with all limits. However, we can also explicitly construct its left adjoint:

$$(\blacktriangleleft X)(n) = X(n + 1)$$

This operation is given by precomposition with $n \mapsto n + 1$, which is a right adjoint on ω with the left adjoint defined by predecessor. Accordingly, $\blacktriangleleft$ itself is a right adjoint:

$$(?_0\, X)(n) = X(\mathsf{pred}\, n)$$

In turn, $?_0\, X$ is a right adjoint as its given by precomposition. The left adjoint is given as follows:

$$(!_1\, X)(n) = \begin{cases} X(0) & n = 0 \\ X(n + 1) & \text{otherwise} \end{cases}$$

More generally, we define $?_i\, X$ and $!_i\, X$ as follows:

$$(?_i\, X)(n) = \begin{cases} X(n) & n \leq i \\ X(n - 1) & n > i \end{cases}$$

$$(!_i\, X)(n) = \begin{cases} X(n) & n < i \\ X(n + 1) & n \geq i \end{cases}$$

In particular, $!(0) = \blacktriangleleft$. Then $!_{i+1} \dashv ?_i \dashv !_i$. This chain continues indefinitely. □

Lemma 9.2.2. *The* $\blacktriangleright$ *modality commutes with connected limits and has a canonical point* $\mathsf{next} : \mathsf{id} \longrightarrow \blacktriangleright$.

While we will defer the proof until Section 9.4 (where we will prove a more general result), we note that $\blacktriangleright$ satisfies the expected principle of Löb induction.

Theorem 9.2.3. *For any $X : \mathbf{PSh}(\omega)$, there is a map* $\mathsf{loeb} : X^{\blacktriangleright X} \to X$ *such that* $\mathsf{loeb} \circ f = \epsilon \circ \langle f, \mathsf{next} \circ \mathsf{loeb} \circ f \rangle$.

The next class of functors we are concerned with is the chain of adjoints between $\mathbf{PSh}(\omega)$ and $\mathbf{Set}$. Some of these may be composed to obtain the interpretation of the $\square$ modality mentioned above.

Lemma 9.2.4. *The global sections functor $\Gamma : \mathbf{PSh}(\omega) \longrightarrow \mathbf{Set}$ is part of an adjoint chain: $\ldots \dashv \Pi_0 \dashv \Delta \dashv \Gamma$*

Proof. As a presheaf topos, the rightmost adjoints $\Delta \dashv \Gamma$ always exist. In particular, $(\Delta S)(n) = S$ for all n. The left adjoint to this operation is defined by evaluation at the initial object of ω, so $\Pi_0 X = X(0)$. As this functor is defined by precomposition, it has a left adjoint given as follows:

$$(L\,S)(n) = \begin{cases} \emptyset & n > 0 \\ S & n = 0 \end{cases}$$

Here the adjoint chain ends: $L\,\mathbf{1}$ is not terminal so L cannot be a right adjoint. □

Corollary 9.2.5. $\square = \Delta \circ \Gamma$ *is a right adjoint and preserves filtered colimits.*

Remark 9.2.6. Thus far, our discussion of guarded recursion has been limited to the functors $\blacktriangleright$ and $\square$ and those that arise in their study. A much richer framework of "time-altering" functors is available in $\mathbf{PSh}(\omega)$. This theory has been explored by Guatto and collaborators under the name *time-warps* [Goo+21; Gua18]. Roughly, a time-warp is given by a cocontinuous functor $w : \omega + 1 \longrightarrow \omega + 1$. Each such functor induces a right adjoint $w_* : \mathbf{PSh}(\omega) \longrightarrow \mathbf{PSh}(\omega)$ by taking right Kan extension and taking advantage of the equivalence between $\mathbf{PSh}(\omega)$ and $\mathbf{Sh}(\omega + 1)$ with the canonical topology. ◇

9.3 Mode theories for guarded recursion

While we have discussed several possible functors for working with guarded recursion, we will limit ourselves to just three functors: $\square$ (though often decomposed into its constituent adjoints), $\blacktriangleright$, and $\blacktriangleleft$. We will therefore consider MTT instantiated with some fragment of the following mode theory whose 0 and 1-cells are generated by the following diagram:

$$\begin{tikzcd} t \arrow[loop left, "{l,e}"] \arrow[rr, bend left, "d"] & & s \arrow[ll, bend left, "g"] \end{tikzcd} \tag{9.1}$$

Remark 9.3.1. It may surprise the reader to see two modes in this mode theory given that the goal was to study $\mathbf{PSh}(\omega)$. It proves easier to work with the $\square$ modality after decomposing it into a pair of adjoint modalities. We therefore introduce a mode to represent $\mathbf{Set}$ as well as $\mathbf{PSh}(\omega)$ to facilitate this decomposition. This approach to guarded recursion is unique to MTT, as it was the first system for guarded recursion to facilitate multiple modes. ◇

We will supplement the hom-sets of this category with a partial order. In particular, we require that (d, g) and (e, l) form Galois connections $d \dashv g$ and $e \dashv l$. We additionally require a 2-cell $\mathsf{id} \leq l$ as well as identifications $g \circ d = \mathsf{id}$, $e \circ l = \mathsf{id}$ and $g \circ l = g$.

Notation 9.3.2. We will write $b : t \longrightarrow t$ as shorthand for $d \circ g$.

Definition 9.3.3. Let $\mathcal{M}_{\mathsf{g}}$ denote the 2-category generated by the above data. Note that while we treat $\mathcal{M}_{\mathsf{g}}$ as a 2-category, its hom-categories are always posets.

Intuitively, $l = \blacktriangleright$, $e = \blacktriangleleft$, $g = \Gamma$, and $d = \Delta$. The identities posited by this mode theory are all trivially satisfied up to isomorphism by $\mathbf{PSh}(\omega)$ and they are *strictly* satisfied by the relevant left adjoints (which are all defined by precomposition).

The reason to work with this more restrictive mode theory is the greater degree of generality it affords. We can interpret all of these functors in $\mathbf{Sh}(\alpha)$ where α is an arbitrary inaccessible ordinal rather than merely over $\mathbf{PSh}(\omega)$.

Lemma 9.3.4. *Given any inaccessible ordinal α, there exists a pseudofunctor $F : \mathcal{M}_{\mathsf{g}} \longrightarrow \mathbf{Cat}$ sending t to $\mathbf{Sh}(\alpha)$ and s to $\mathbf{Set}$. The 1-cells are interpreted by corresponding generalizations of $\blacktriangleright$, $\blacktriangleleft$, Γ, and Δ.*

Proof. We shall construct a strict 2-functor $\hat{F} : {\mathcal{M}_{\mathsf{g}}}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ choosing left adjoints and obtain the required functor by transposing. The 0-cells have been fixed already, so it suffices to define $\hat{F}$ on 1- and 2-cells. We define each 1-cell by precomposition. In particular, $l = \mathsf{succ}^*$, $e = \mathsf{pred}^*$, $g = \mathsf{const}(\star)^*$ and $d = \mathsf{const}(0)^*$.

All the relevant 2-cells are obtained by applying precomposition to the (necessarily unique) inequalities of these operations. □

9.4 Extensional guarded MTT

In this section, we examine in some detail the model of MTT induced by Lemma 9.3.4. In particular, we show that this functor extends to a model of extensional MTT which validates Löb induction. We may then use MTT as an internal language for this model which can construct and reason about coinductive types. The results in this section form the basis and motivation of the constructions given in Chapter 10 where we use similar ideas to construct a synthetic model of Iris. Many of these results were present in Gratzer et al. [Gra+21], but we take advantage of improvements in the model theory of MTT and generalize all our results to apply to α for an arbitrary ordinal α.

9.4.1 A model of extensional guarded MTT

The first order of business is to improve Lemma 9.3.4 into a model of extensional MTT. Fortunately, both $\mathbf{Set}$ and $\mathbf{Sh}(\alpha)$ are Grothendieck topoi so this is a direct consequence of Theorem 7.2.12:

Theorem 9.4.1. *The functor defined in Lemma 9.3.4 induces a model of MTT internalizing these functors.*

Notation 9.4.2. The fact that $\mathcal{M}_{\mathsf{g}}$ is locally posetal ensures that a 2-cell is uniquely determined by its type. Accordingly, we will routinely elide 2-cells when writing terms in MTT instantiated with this mode theory.

Theorem 9.4.3. *This model of* MTT *validates Löb induction and its corresponding computation principle along with the evident substitution law:*

$$\frac{\Gamma.(l \mid A) \vdash M : A \mathbin{@} t}{\Gamma \vdash \mathsf{loeb}(M) : A \mathbin{@} t} \qquad \frac{\Gamma.(l \mid A) \vdash M : A \mathbin{@} t}{\Gamma \vdash \mathsf{loeb}(M) = M[\mathsf{id}.\mathsf{loeb}(M)] : A \mathbin{@} t}$$

Proof. Fix an interpretation X of Γ in $\mathbf{Sh}(\alpha)$. We first observe that the slice $\mathbf{Sh}(\alpha)/X$ has a particularly concrete presentation as $\mathbf{Sh}(\int X)$. We also have a relative version of $\blacktriangleright_X$ on this slice category using the point $\mathsf{id} \longrightarrow \blacktriangleright$. Unfolding definitions, we may interpret A and M as an object Y of $\mathbf{Sh}(\int X)$ and a morphism $f : \blacktriangleright_X Y \longrightarrow Y$.

It suffices to argue that Y has a suitable global element α satisfying $\alpha = f \circ \mathsf{next} \circ \alpha$. At this point, we take advantage of the presentation described above to construct α by describing its behavior at each $(\beta, x \in X_\beta)$. We proceed by induction on β.

The 0 case is trivial by the sheaf condition[5] so we begin with $\beta = 1$. In this case, $(\blacktriangleright_X Y)(1, x) = \mathbf{1}$, so we define $\alpha_{(1,x)} = f(\star)$. There is no naturality obligation in this case, and the required computation rule is immediate. The case where β is a limit ordinal is immediate; the sheaf condition on $\int X$ gives a unique α by amalgamating and it automatically satisfies the required equation by unicity of amalgamations.

The interesting case, therefore, is the case where $\beta = \gamma + 1$. In this case, we define α as follows:

$$\alpha_{\beta,x} = f_{\beta,x}(\alpha_{\gamma,x|_\gamma})$$

The computation rule for $\alpha_{\gamma,x|_\gamma}$ ensures that this definition is natural and $\alpha_{\beta,x}$ satisfies the computation rule by construction.

We omit the arguments regarding the substitution principle, as it follows essentially from the standard proof that guarded fixed points are unique. □

Notation 9.4.4. We avail ourselves of more standard notation for $\langle l \mid A\rangle$ and $\langle d \circ g \mid B\rangle$ by writing $\blacktriangleright A$ and $\square B$ respectively. Notice that these are the "dependent" forms of these modalities so e.g. A is essentially an element of $\blacktriangleright \mathcal{U}$.

9.4.2 Programming in guarded MTT

Having constructed a model of extensional MTT with $\mathcal{M}_\mathsf{g}$ and Löb induction, we now put it to use to reconstruct several familiar results from guarded recursion. In particular, we will the standard type of guarded streams and show that it can be used to *define* the final coalgebra for the functor $A \times -$. With this theorem to hand, we show that we can define and reason about stream combinators (constructed in the mode modeled by $\mathbf{Set}$) using the guarded apparatus.

Preliminaries Before developing these examples, we define and prove some preliminary results which internalize the structure of the mode theory. We give more memorable names to some operations which are frequently used:

$$\mathsf{next} : A \to \blacktriangleright A \tag{9.2}$$

[5] In this model, the case of 0 is trivial: any sheaf X must satisfy $X(0) = \{\star\}$.

$$
\begin{aligned}
\mathsf{now} : \Box\blacktriangleright A &\simeq \Box A && (9.3)\\
\mathsf{extract} : \Box A &\to A && (9.4)\\
\mathsf{dup} : \Box A &\simeq \Box\Box A && (9.5)
\end{aligned}
$$

Notation 9.4.5. Throughout this subsection we work internally to MTT and avail ourselves of the notation introduced in Section 6.1. We also write $a = b$ rather than $\mathsf{Id}(a, b)$.

Recall in Section 6.1.5, the equations between 2-cells in a mode theory manifest into equations between the combinators they induce. In this theory, all 2-cells are equal and so essentially every coherence equation one might expect to hold, does hold. We record two instances of this phenomenon for future use.

Lemma 9.4.6. $(\Box, \mathsf{extract}, \mathsf{dup})$ *is an idempotent comonad.*

Proof. We must show that these operations satisfy the comonad laws. That is, we wish to show that the following equations hold:

$$(a : \Box A) \to \mathsf{extract}(\mathsf{dup}(a)) = a \tag{9.6}$$

$$(a : \Box A) \to (\mathsf{mod}_b(\mathsf{extract}) \circledast \mathsf{dup}(a)) = a \tag{9.7}$$

$$(a : \Box A) \to (\mathsf{mod}_b(\mathsf{dup}) \circledast \mathsf{dup}(a)) = \mathsf{dup}(\mathsf{dup}(a)) \tag{9.8}$$

Each one of these identities follows from induction on a. The proofs are considerably simpler than those of Section 6.1.5 because there is no need to keep track of particular 2-cells; only their boundaries.

We describe the proof of the first identity to give a flavor for the proofs. Fix $a :_{\mathsf{id}} \Box A$. By induction on $\Box A$, we may assume that $a = \mathsf{mod}_b(a_0)$ for some $a_0 :_b A$. In this case, the necessary equation reduces to the following:

$$
\begin{aligned}
&\mathsf{extract}(\mathsf{dup}(\mathsf{mod}_b(a_0)))\\
&= \mathsf{extract}(\mathsf{mod}_b(\mathsf{mod}_b(a_0)))\\
&= \mathsf{mod}_b(a_0)\\
&= a
\end{aligned}
$$

The conclusion then follows immediately. □

Lemma 9.4.7. *The map* $\mathsf{mod}_b(\mathsf{next}) \circledast - : \Box A \to \Box\blacktriangleright A$ *is a section to* now*. In particular, it is an equivalence.*

Proof. We prove only the first claim, as the second is a routine calculation. We must show the following equation holds:

$$(a : \Box\blacktriangleright A) \to (\mathsf{mod}_b(\mathsf{next}) \circledast \mathsf{now}(a)) = a$$

To this end, fix $a :_{\mathsf{id}} \Box\blacktriangleright A$. Using the elimination principles for $\Box$ and $\blacktriangleright$, we may assume that $a = \mathsf{mod}_b(\mathsf{mod}_l(a_0))$ for some $a_0 :_b A$. In this case, $\mathsf{now}(a) = \mathsf{mod}_b(a_0)$. Computing further, we see the left-hand side of the desired equality is $\mathsf{mod}_b(\mathsf{mod}_l(a_0))$ which is equal to a by assumption. □

Constructing guarded streams We now set out to define guarded streams and the various combinators around them. To begin with, we prove an often-used result showing that guarded fixed points are unique. This proof also illustrates the general flavor of working in this guarded type theory.

Notation 9.4.8. When working in this informal style, it is often convenient to treat loeb as a closed element of $(\blacktriangleright A \to A) \to A$ subject to the equation $\mathsf{loeb}(f) = f(\mathsf{next}(\mathsf{loeb}(f)))$. We will pass back and forth between loeb which accepts a function and loeb which directly uses a binder.

Lemma 9.4.9. *Given $A : \mathcal{U}$, $f : \blacktriangleright A \to A$, and $a : A$, if $a = f(\mathsf{next}(a))$ then $a = \mathsf{loeb}(f)$. Phrased differently, the type of guarded fixed points is contractible:*

$$\textstyle\sum_{a:A} a = f(\mathsf{next}(a))$$

Proof. Suppose that $a = f(\mathsf{next}(a))$. We will show that $a = \mathsf{loeb}(f)$ by constructing a proof of equality using loeb itself. To this end, we recall that the following map is an equivalence (Lemma 6.3.4): $(a\, b :_l A) \to \mathsf{mod}_l(a) = \mathsf{mod}_l(b) \to \blacktriangleright(a = b)$. In particular, using the inequality $\mathsf{id} \leq l$ we conclude obtain the following map:

$$(a, b : A) \to \blacktriangleright(a = b) \simeq \mathsf{next}(a) = \mathsf{next}(b) \tag{9.9}$$

With this in mind, we use loeb to prove $a = \mathsf{loeb}(f)$. Accordingly, we may assume $\blacktriangleright(a = \mathsf{loeb}(f))$ and therefore $\mathsf{next}(a) = \mathsf{next}(\mathsf{loeb}(f))$. Applying f to both sides of the equivalence, we obtain:

$$a = f(\mathsf{next}(\mathsf{loeb}(f))) = f(\mathsf{next}(\mathsf{loeb}(f))) = \mathsf{loeb}(f)$$

□

The actual definition of guarded streams is straightforward. We have access to a universe and a dependent form of the $\blacktriangleright$ modality, so we can define streams through Löb induction.

Definition 9.4.10. We define $\mathsf{GStream}\, A = \mathsf{loeb}(s.\ A \times \blacktriangleright s)$.

As a consequence of loeb unfolding definitionally, we immediately conclude that $\mathsf{GStream}\, A = A \times \blacktriangleright \mathsf{GStream}\, A$. We use this equality to define the expected introduction and elimination forms for streams:

$$\begin{aligned}
&\mathsf{gcons} : A \to \blacktriangleright(\mathsf{GStream}\, A) \to \mathsf{GStream}\, A\\
&\mathsf{gcons}\, a\, s = \langle a, s \rangle\\[1em]
&\mathsf{ghd} : \mathsf{GStream}\, A \to A\\
&\mathsf{ghd}\, \langle a, s \rangle = a\\[1em]
&\mathsf{gtl} : \mathsf{GStream}\, A \to \blacktriangleright(\mathsf{GStream}\, A)\\
&\mathsf{gtl}\, \langle a, s \rangle = s
\end{aligned}$$

The β and η equalities for these operations hold by virtue of the corresponding equations for dependent sums:

Lemma 9.4.11. *The β and η principles for streams hold for $\mathsf{GStream}_A$. Explicitly:*

$$(s : \mathsf{GStream}\,A) \to s = \mathsf{gcons}\,(\mathsf{ghd}\,s)\,(\mathsf{gtl}\,s)$$
$$(a : A)(s : \blacktriangleright(\mathsf{GStream}\,A)) \to \mathsf{ghd}(\mathsf{gcons}\,a\,s) = a$$
$$(a : A)(s : \blacktriangleright(\mathsf{GStream}\,A)) \to \mathsf{gtl}(\mathsf{gcons}\,a\,s) = s$$

The presence of the $\blacktriangleright$ on the second argument to gcons enables us to use loeb induction to construct elements of $\mathsf{GStream}\,A$. For a pair of classic examples, we provide the following code which computes a constant stream of a given element along with the stream of Fibonacci numbers:

$$
\begin{aligned}
&\mathsf{const} : A \to \mathsf{GStream}\,A\\
&\mathsf{const}\,a = \mathsf{loeb}(s.\ \mathsf{gcons}\,a\,s)\\[1em]
&\mathsf{helper} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{GStream}\,\mathsf{Nat}\\
&\mathsf{helper} = \mathsf{loeb}(f.\ \lambda n\,m.\ \mathsf{gcons}\,n\,\mathsf{mod}_l(f\,m\,(n+m)))\\[1em]
&\mathsf{fib} : \mathsf{GStream}\,\mathsf{Nat}\\
&\mathsf{fib} = \mathsf{helper}\,0\,1
\end{aligned}
$$

We may just as easily define higher-order stream combinators and reason about them using guarded recursion. For instance, we may define the mapping operation on streams:

$$
\begin{aligned}
&\mathsf{map} : (A \to B) \to \mathsf{GStream}\,A \to \mathsf{GStream}\,B\\
&\mathsf{map}\,f = \mathsf{loeb}(\lambda r.\ \lambda s.\ \mathsf{gcons}\,(f(\mathsf{ghd}\,s))\,(r \circledast \mathsf{gtl}\,s))
\end{aligned}
$$

Reasoning about guarded-recursive combinators like map also factors through loeb (and judicious application of Lemma 6.3.4).

Lemma 9.4.12. *Given a pair of functions $f : A \to B$ and $g : B \to C$, we have an equality $\mathsf{map}\,g \circ \mathsf{map}\,f = \mathsf{map}\,(g \circ f)$.*

Proof. By function extensionality, we may fix $s : \mathsf{GStream}\,A$ and show that the following equation holds:

$$\mathsf{map}\,g\,(\mathsf{map}\,f\,s) = \mathsf{map}\,(g \circ f)\,s$$

We proceed by guarded recursion (after generalizing over s). Accordingly, we may assume the following:

$$\blacktriangleright((s :_{\mathsf{id}} \mathsf{GStream}\,A) \to \mathsf{map}\,g\,(\mathsf{map}\,f\,s) = \mathsf{map}\,(g \circ f)\,s)$$

Using the η principle for streams, we may assume that $s = \mathsf{gcons}\,h\,t$ for some $h : A$ and $t : \blacktriangleright\mathsf{GStream}\,A$. Using the elimination principle for $\blacktriangleright$, moreover, we assume $t = \mathsf{mod}_l(t)$ for some $t_0 :_l \mathsf{GStream}\,A$. We may now compute and observe that the above equation reduces to the following:

$$\mathsf{gcons}\,(g(f\,h))\,\mathsf{mod}_l(\mathsf{map}\,g\,(\mathsf{map}\,f\,t_0)) = \mathsf{gcons}\,(g(f\,h))\,\mathsf{mod}_l(\mathsf{map}\,(g \circ f)\,t_0)$$

Using congruence, it therefore suffices to argue that $\mathsf{mod}_l(\mathsf{map}\,g\,(\mathsf{map}\,f\,t_0))$ and $\mathsf{mod}_l(\mathsf{map}\,(g \circ f)\,t_0)$ are equal. Using Lemma 6.3.4, we may construct such an identification from a term of the following type:

$$\blacktriangleright(\mathsf{map}\,g\,(\mathsf{map}\,f\,t_0) = \mathsf{map}\,(g \circ f)\,t_0)$$

At this point, we invoke our induction hypothesis. By instantiating it with t_0, the desired conclusion is immediate. □

The combinators that cannot be described on $\mathsf{GStream}\, A$ are just as interesting. The presence of the $\blacktriangleright$ in the type of gcons allows us to use Löb induction to define streams but it forces the type of gtl to also contain a $\blacktriangleright$. This, in turn, prevents us from defining natural combinators that do not produce their output in perfect lock-step with their input. For instance, we might reasonably hope to define a combinator that selects the elements of a stream at odd positions (the first, third, fifth elements, etc.), but this cannot be done with $\mathsf{GStream}\, A$.

It is instructive to see why a naive attempt to define such a function fails. Let us attempt to use loeb to define this function and therefore fix $r : \blacktriangleright(\mathsf{GStream}\, A \to \mathsf{GStream}\, A)$ along with $s : \mathsf{GStream}\, A$. Selecting the first element of s is easy: $a_0 = \mathsf{ghd}\, s$ suffices. The complication comes from attempting to apply r. To ensure that we skip the second element, we want to apply r not to $\mathsf{gtl}\, s$ but to $\mathsf{gtl} \circledast \mathsf{gtl}\, s$. The type of this latter expression is $\blacktriangleright\blacktriangleright(\mathsf{GStream}\, A)$ and so if we apply r to it (using $\circledast$), we obtain $\blacktriangleright\blacktriangleright(\mathsf{GStream}\, A)$. This term, however, cannot be used as the tail of our stream; it has the type $\blacktriangleright\blacktriangleright(\mathsf{GStream}\, A)$ rather than the required $\blacktriangleright(\mathsf{GStream}\, A)$. At this point, we are stuck. If we attempted to adjust the type of this "drop" function, we would also change the type of r and we would end up with a nearly identical position.

In some ways, this is a feature of the $\mathsf{GStream}$ type; operations $\mathsf{GStream}\, A \to \mathsf{GStream}\, A$ are cut down to synchronous operations. Indeed, we can argue *internally* that it is impossible to define a function that violates this invariant:

Lemma 9.4.13. *If we are able to construct* $\mathsf{gtl}' : \mathsf{GStream}\, \mathsf{Bool} \to \mathsf{GStream}\, \mathsf{Bool}$ *such that* $s = \mathsf{gcons}\, (\mathsf{ghd}\, s)\, (\mathsf{next}(\mathsf{gtl}'\, s))$ *then* $\blacktriangleright\bot$.

Proof. Suppose that gtl' exists. We will first use it to obtain an (impossibly) well-behaved function $\blacktriangleright\mathsf{Bool} \to \mathsf{Bool}$:

$$\begin{aligned}&f : \blacktriangleright\mathsf{Bool} \to \mathsf{Bool}\\ &f\, b = \mathsf{ghd}(\mathsf{gtl}'(\mathsf{gcons}\, \mathsf{tt}\, (\mathsf{const} \circledast b)))\end{aligned}$$

Unfolding the definitions of const and using our presumed equations for gtl', we see that f satisfies the following equation:

$$\mathsf{next}(f\, b) = b$$

Let us now consider the following boolean:

$$b = \mathsf{loeb}(\mathsf{not} \circ f)$$

Unfolding, we conclude that $\mathsf{not}\, b = f(\mathsf{next}\, b)$. Applying next to both sides we have $\mathsf{next}(\mathsf{not}\, b) = \mathsf{next}(b)$. Using Lemma 6.3.4, we conclude $\blacktriangleright(\mathsf{not}\, b = b)$ and obtain $\blacktriangleright\bot$ from the standard argument. □

While it is interesting to be able to define such a strict type of streams, it is still often necessary to work with processors which violate such a strict productivity invariant. To accommodate them, we could consider enriching $\mathsf{GStream}$ to allow for a variable number of $\blacktriangleright$s (Section 9.6.2) but in this section, we opt to instead use the other modalities available in guarded MTT to produce a type of coinductive streams from the type of guarded streams.

Intuitively, we will define the type of coinductive streams as something like $\mathsf{Stream}\,A = \langle g \mid \mathsf{GStream}\,A\rangle$ but some care is needed: what mode does this definition live at and what, precisely, is the type of A? When working with guarded streams, we were able to work purely within mode t and take advantage of $1 \leq l$ to essentially ignore modalities. For coinductive streams, we will define $\mathsf{Stream}\,A$ @ s. We will draw A from mode s as well and so rather than taking the global sections of $\mathsf{GStream}\,A$ (which is ill-moded) we will instead take global sections of $\mathsf{GStream}\,\langle d \mid A\rangle$.[6] In total:

$$\mathsf{Stream}\,A \mathbin{@} s = \langle g \mid \mathsf{GStream}\,\langle d \mid A\rangle\rangle$$

Recall that $g \circ d = \mathsf{id}$, so no modification is needed to A to move it to a context restricted by g and then restricted again by d. Every modality in MTT satisfies a dependent version of axiom K so that, in particular, $\langle \mu \mid A \times B\rangle \simeq \langle \mu \mid A\rangle \times \langle \mu \mid B\rangle$. Instantiating $\mu = g$, we now obtain the following equivalences:

$$\begin{aligned}
&\mathsf{Stream}\,A \\
&= \langle g \mid \mathsf{GStream}\,\langle d \mid A\rangle\rangle \\
&\simeq \langle g \mid \langle d \mid A\rangle\rangle \times \langle g \mid \blacktriangleright \mathsf{GStream}\,\langle d \mid A\rangle\rangle && \text{Axiom K} \\
&\simeq A \times \langle g \mid \mathsf{GStream}\,\langle d \mid A\rangle\rangle && g \circ d = \mathsf{id} \text{ and } g \circ l = g \\
&\simeq A \times \mathsf{Stream}\,A
\end{aligned}$$

This equivalence induces the introduction and elimination rules for Stream:

$$\begin{aligned}
&\mathsf{cons} : A \to \mathsf{Stream}\,A \to \mathsf{Stream}\,A \\
&\mathsf{hd} : \mathsf{Stream}\,A \to A \\
&\mathsf{tl} : \mathsf{Stream}\,A \to \mathsf{Stream}\,A
\end{aligned}$$

The β and η principles for $\mathsf{GStream}$ follow directly from those for dependent sums.

Unlike with $\mathsf{GStream}$, the presence of this isomorphism does not fully constrain Stream; there is no analog of Lemma 9.4.9 which ensures that all fixed points to $A \times -$ are unique. In particular, we do not a priori have an induction or coinduction principle for Stream while $\mathsf{GStream}$ had both in the guise of Löb induction. We will now show, however, that Löb induction on $\mathsf{GStream}$ is sufficient to induce a coinduction principle on Stream.

Theorem 9.4.14. $\mathsf{Stream}\,A$ *together with* $\langle \mathsf{hd}, \mathsf{tl}\rangle$ *is the final coalgebra for the functor* $A \times -$.

Proof. Fix another coalgebra (Z, α) for $A \times -$ so $\alpha : Z \to A \times Z$. Let us write α_i for the composite $\pi_i \circ \alpha$. We must show that there is a unique function $f : Z \to \mathsf{Stream}\,A$

[6] Similar restrictions are manifested in prior work on guarded type theories with coinductive types [Biz+16; Clo+15; KMV22] where A must be suitably constant.

such that the following diagram commutes:

$$\begin{tikzcd}
Z \arrow[r] \arrow[d] & \mathsf{Stream}\,A \arrow[d] \\
A \times Z \arrow[r] & A \times \mathsf{Stream}\,A
\end{tikzcd}$$

We begin by defining f. Recall that $\mathsf{Stream}\,A = \langle g \mid \mathsf{GStream}\,\langle d \mid A\rangle\rangle$ and we have assumed that $\langle g \mid -\rangle$ is a right adjoint. Using Lemma 6.4.2, it suffices to construct a transposed version:

$$\widehat{f} : \langle g \mid \langle d \mid Z\rangle \to \mathsf{GStream}\,\langle d \mid A\rangle\rangle$$

This type is amenable to Löb induction:

$$\{g\} \vdash \mathsf{helper} : \langle d \mid Z\rangle \to \mathsf{GStream}\,\langle d \mid A\rangle \;@\; t$$
$$\{g\} \vdash \mathsf{helper} = \mathsf{loeb}(f.\ \lambda z.\ \mathsf{gcons}\,(\mathsf{mod}_d(\alpha_1) \circledast z)\,(\mathsf{mod}_l(f\,(\mathsf{mod}_d(\alpha_2) \circledast z))))$$

$$\widehat{f} = \mathsf{mod}_g(\mathsf{helper})$$

$$f\,z = \mathsf{mod}_g(\mathsf{helper}\,\mathsf{mod}_d(z))$$

In the above code, we have used $\{g\} \vdash -$ to indicate that the entire definition of helper takes place with the ambient context restricted by g. For convenience, we have unfolded the definition of f for later use; it is particularly simple in this case because the unit of the adjunction is an isomorphism.

It remains to show that f fits into the necessary commuting square. Note that the map $\langle \mathsf{hd}, \mathsf{tl}\rangle$ is an isomorphism, so it suffices to show that the map $\mathsf{gcons} \circ (A \times f) \circ \alpha$ is equal to f. This, in turn, follows more-or-less by computation; up to some details around guarded recursion, this is how f was defined:

$$\begin{aligned}
& f\,z \\
&= \mathsf{mod}_g(\mathsf{helper}\,\mathsf{mod}_d(z)) \\
&= \mathsf{mod}_g(\mathsf{gcons}\,(\mathsf{mod}_d(\alpha_1\,z))\,(\mathsf{next}(\mathsf{helper}\,\mathsf{mod}_d(\alpha_2\,z)))) \\
&= \mathsf{cons}\,(\alpha_1\,z)\,\mathsf{mod}_g(\mathsf{helper}\,\mathsf{mod}_d(\alpha_2\,z)) \\
&= \mathsf{cons}\,(\alpha_1\,z)\,(f(\alpha_2\,z))
\end{aligned}$$

It remains to show unicity.

Suppose we are given a second coalgebra morphism $h : Z \to \mathsf{Stream}\,A$. By Lemma 6.4.2, this is equivalent to an element $\widehat{h} : \langle g \mid \langle d \mid Z\rangle \to \mathsf{GStream}\,\langle d \mid A\rangle\rangle$. Our assumption that h is a coalgebra morphism can be rephrased in terms of $\widehat{h}$. To make the type easier to state, let us use the elimination principle for $\langle g \mid -\rangle$ and assume $\widehat{h} = \mathsf{mod}_g(h_0)$:

$$\langle g \mid (z :_{\mathsf{id}} \langle d \mid Z\rangle) \to h_0\,z = \mathsf{gcons}\,(\alpha_1 \circledast z)\,(\mathsf{next}(h_0\,\mathsf{mod}_d(\alpha_2 \circledast z)))\rangle$$

Notice, however, that this condition ensures that h_0 is a fixed point of a guarded equation. Using Lemma 9.4.9, we may therefore replace this condition with the following:

$$\langle g \mid h_0 = \mathsf{loeb}(r.\ \lambda z.\ \mathsf{gcons}\,(\alpha_1 \circledast z)\,(\mathsf{mod}_l(r\,\mathsf{mod}_d(\alpha_2 \circledast z))))\rangle$$

We immediately recognize the right-hand side of this equation as helper. Therefore, h_0 and $\widehat{f}$ are equal by Lemma 6.3.4 and so f and h are equal by Lemma 6.4.2. □

Notation 9.4.15. We write $\mathsf{corec}(Z, \alpha) : \mathsf{Stream}\, A \to Z$ for the unique coalgebra homomorphism induced by Theorem 9.4.14.

In a precise sense, Theorem 9.4.14 ensures that can construct all of the expected stream operations on $\mathsf{Stream}\, A$. For instance, the problematic example cited earlier (dropping every other element) is now straightforward to define. We must construct a non-standard coalgebra structure on $\mathsf{Stream}\, A$ and then simply apply Theorem 9.4.14 to obtain a unique coalgebra homomorphism witnessing the desired combinator:

$$\begin{aligned}
&\beta : \mathsf{Stream}\, A \to A \times \mathsf{Stream}\, A \\
&\beta\, s = \langle \mathsf{hd}\, s, \mathsf{tl}(\mathsf{tl}\, s)\rangle \\
&\\
&\mathsf{dropodds} = \mathsf{coiter}(\mathsf{Stream}\, A, \beta)
\end{aligned}$$

The presence of dependent types ensures that we are also able to effectively reason about programs computing with streams inside of guarded MTT and without the need for separate logic [Clo+15]. We conclude, therefore, with an example of such reasoning.

Theorem 9.4.16. *There exists a combinator* zipWith *with the following type:*

$$(A \to B \to C) \to \mathsf{Stream}\, A \to \mathsf{Stream}\, B \to \mathsf{Stream}\, C$$

Moreover, if $f : A \to A \to B$ is commutative then so is $\mathsf{zipWith}\, f$*.*

Proof. We begin by defining zipWith. In this case, it is easier to use guarded recursion as zipWith is synchronous and processes two streams at the same time.

$$\begin{aligned}
&\{g\} \vdash \mathsf{helper} : (\_ :_d A \to B \to C) \to \\
&\qquad\qquad \mathsf{GStream}\,\langle d \mid A\rangle \to \mathsf{GStream}\,\langle d \mid B\rangle \to \mathsf{GStream}\,\langle d \mid C\rangle \\
&\{g\} \vdash \mathsf{helper}\,(\oplus) = \mathsf{loeb}(\lambda r\, l\, r.\ \mathsf{gcons}\,(\mathsf{mod}_d(\mathsf{ghd}\, l \oplus \mathsf{ghd}\, r))\,(r \circledast \mathsf{gtl}\, l \circledast \mathsf{gtl}\, r)) \\
&\\
&\mathsf{zipWith} : (A \to B \to C) \to \mathsf{Stream}\, A \to \mathsf{Stream}\, B \to \mathsf{Stream}\, C \\
&\mathsf{zipWith}\, f\, l\, r = \mathsf{mod}_g(\mathsf{helper}\, f) \circledast l \circledast r
\end{aligned}$$

We now turn to commutativity. Examining definitions, it suffices to argue that if $\oplus$ is commutative then $\mathsf{helper}\,(\oplus)$ is commutative. In this case, we are able to take advantage of guarded recursion and the proof is straightforward.

Concretely, suppose fix the following induction hypothesis:

$$\blacktriangleright((l\, r : \mathsf{GStream}\, A) \to \mathsf{helper}\,(\oplus)\, l\, r = \mathsf{helper}\,(\oplus)\, r\, l)$$

Fix $l, r : \mathsf{GStream}\, A$. We wish to show now that $\mathsf{helper}\,(\oplus)\, l\, r = \mathsf{helper}\,(\oplus)\, r\, l$. Unfolding, it suffices to show the following pair of equalities:

$$\begin{gathered}
\mathsf{mod}_d(\mathsf{ghd}\, l \oplus \mathsf{ghd}\, r) = \mathsf{mod}_d(\mathsf{ghd}\, r \oplus \mathsf{ghd}\, l) \\
\mathsf{next}(\mathsf{helper}\,(\oplus)) \circledast \mathsf{gtl}\, l \circledast \mathsf{gtl}\, r = \mathsf{next}(\mathsf{helper}\,(\oplus)) \circledast \mathsf{gtl}\, r \circledast \mathsf{gtl}\, l
\end{gathered}$$

The first follows immediately from Lemma 6.3.4 and our assumption that $\oplus$ is commutative. The latter follows from Lemma 6.3.4 and the induction hypothesis. □

9.5 Decidable conversion and Löb induction

Extensional guarded MTT is a very usable approach to internalizing guarded recursion in type theory, but with a major caveat: as it is based on extensional type theory, type-checking is undecidable. Recall from Section 9.1 that guarded type theories based on delayed substitutions suffered from two deficiencies: undecidable type-checking and unwieldy syntax. We have essentially shown that extensional MTT addresses the second point, but it does not offer any solution to the first.

Naïvely, one might hope that restricting from extensional MTT to intensional MTT is sufficient to solve the problem. It is certainly true that the result enjoys decidable type-checking absent Löb induction (Chapter 8), but one must then reintegrate Löb induction. Doing so by directly adding the rules specified in Theorem 9.4.3 is unfortunately impossible. We show in this section that in the presence of a few minor requirements (implied, for instance, by the presence of $\square$) that the resulting system cannot have decidable type-checking.

This is an unfortunate state of affairs; a major motivation for MTT was the ability to integrate $\square$ and $\blacktriangleright$ into a single system without compromising either usability or decidability of type-checking. Having accomplished this, we find that the challenges around Löb induction and decidable type-checking are just as severe.

9.5.1 The no-go theorem

To make this theorem as broadly applicable as possible, we work in a stripped-down guarded type theory and isolate the minimal requirements for our theorem. For this section, we therefore fix Martin-Löf type theory extended with $\blacktriangleright$, $\circledast$, next and loeb. Suppose further that the following conditions hold:

1. loeb unfolds (as in Theorem 9.4.3)
2. There exists a type S equipped with an isomorphism $\iota : \mathsf{Nat} \times \blacktriangleright S \cong S$
3. next is *globally adequate*.[7]

Definition 9.5.1. We call next globally adequate if it is injective on closed terms.

Remark 9.5.2. Notice that next is automatically globally adequate in a system with $\square$ or $\blacktriangleleft$ as it has a retraction on closed terms. Similarly, the existence of S is automatic in the presence of a dependent $\blacktriangleright$ and a universe. ◇

We will show that if type-checking is decidable for this system, then we are able to solve the following (undecidable) problem:

Theorem 9.5.3 (Rosser [Ros36], Kleene [Kle50], Trahtenbrot [Tra53])**.** *There is no total computable function which separates the following sets:*

$$A = \{M \mid M \downarrow 0\} \qquad B = \{M \mid M \downarrow 1\}$$

In other words, no function terminates on all inputs while returning 0 *on machines that terminate with output* 0 *and* 1 *on those that return* 1*.*

[7] The terminology is borrowed from Palombi and Sterling [PS23], though their definition is incomparable; it applies only to natural numbers but requires next to be an isomorphism in this case.

Remark 9.5.4. The following reduction to Theorem 9.5.3 is directly inspired by the similar result due to Berger and Setzer [BS18]. It is from them that we also draw the above references to Theorem 9.5.3. $\diamond$

Essentially, we shall argue that the ability to decide definitional equality enables us to separate these two sets of Turing machines. To this end, we require several facts about encoding Turing machines and their execution within type theory. None of this is specific to our setting, indeed, it is well-known that the following operations can be defined in any logic supporting primitive recursion.

Lemma 9.5.5. *The following types and operations are definable within type theory:*

- *A type* TM *of effective encodings of Turing machines (*$\mathsf{TM} = \mathsf{Nat}$ *is sufficient).*
- *A type* State *representing the state of a tape and a Turing machine.*
- $\mathsf{init} : \mathsf{TM} \to \mathsf{State}$ *converting a Turing machine to a state with a fresh tape.*
- $\mathsf{step} : \mathsf{State} \to \mathsf{State}$ *which advances the state by one step of computation or does nothing if the machine has halted.*
- $\mathsf{done} : \mathsf{State} \to \mathsf{Bool}$ *which returns* tt *just when the machine is in a halted state.*
- $\mathsf{result} : \mathsf{State} \to \mathsf{Nat}$ *which returns the result of the computation if the machine has halted and* 0 *otherwise.*

With these operations to hand, we now give associate to each $M : \mathsf{TM}$ an (infinite) stream $\mathsf{exec}(M)$ which represents the progress of its computation. Intuitively, if M halts with result ϵ in n steps then $\mathsf{exec}(M)$ will be of the form $0^n\epsilon^\omega$ and if M diverges then $\mathsf{exec}(M) = 0^\omega$. Defining exec requires only a few combinators specific to guarded recursion:

$$\begin{aligned}
&\mathsf{exec} : \mathsf{TM} \to S\\
&\mathsf{exec}(M) = \mathsf{go}(\mathsf{init}\, M)\\
&\quad \mathsf{go} : \mathsf{State} \to S\\
&\quad \mathsf{go} = \mathsf{loeb}(r.\ \lambda s.\ \textbf{if}\ \mathsf{done}\, s\ \textbf{then}\ \mathsf{const}(\mathsf{result}\, s)\ \textbf{else}\ \iota(0, r \circledast \mathsf{next}(\mathsf{step}\, s)))
\end{aligned}$$

In particular, we require the following lemma characterizing the behavior of exec which follows directly from computation.

Lemma 9.5.6. *If* $M : \mathsf{TM}$ *represents a Turing machine that terminates in* n *steps with result* ϵ *then* $\mathsf{exec}\, M = \iota(0, \mathsf{next}(\iota(0, \mathsf{next}(\dots (\mathsf{const}\, \epsilon)))))$*.*

Corollary 9.5.7. *If* M_0 *computes to* 0 *and* M_1 *computes to* 1*, then* $\mathsf{exec}\, M_0$ *is not definitionally equal to* $\mathsf{exec}\, M_1$*.*

Proof. Suppose M_ϵ terminates after n_ϵ steps. Using Lemma 9.5.6, the elements at of these two streams at $k = \max(n_0, n_1)$ differ: one will be $\mathsf{next}^k(0)$ and one will be $\mathsf{next}^k(1)$. Invoking our assumption that next is globally adequate, if $\mathsf{next}^k(0) = \mathsf{next}^k(1)$ definitionally, then $0 = 1$ definitionally: a clear contradiction. $\square$

Corollary 9.5.8. *If* M *computes to* 0 *then* $\mathsf{exec}\, M_0 = \mathsf{const}(0)$*.*

Proof. This again follows from Lemma 9.5.6. In particular, we see that these two terms become identical after finitely many unfoldings of both. □

Theorem 9.5.9. *Guarded type theory with the above assumptions cannot have decidable type-checking.*

Proof. If the theory had decidable type-checking, then we could decide the equality of two terms (using, for instance, refl). In this case, we could separate the set of Turing machines computing to 0 and 1 by virtue of the following function:

$$F(M) = \mathsf{exec}\,\bar{M} \overset{?}{=} \mathsf{const}\,0$$

Using Corollary 9.5.8, this returns 1 for machines which terminate with 0 and by Corollary 9.5.7 it returns 0 for machines which compute to 1. □

9.6 Stratified guarded MTT

At this point the state of affairs for guarded MTT is as follows:

- if we are not concerned about implementability, extensional guarded MTT satisfies all the desired criteria;
- if we are concerned about implementability, there is no direct way to include Löb induction (irrespective of how we handle ▶ and □).

If we want to find an implementable version of guarded MTT, only one possible path remains then: find some alternative way to present Löb induction so as to escape Theorem 9.5.9.

A possible solution to the problem of Löb induction is suggested by CloTT: integrate Löb induction but severely curtail the situations where it can unfold. In particular, CloTT allows loeb to unfold just when the clock of the relevant ▶ is suitably fresh. In practice, this happens just when the clock has just been bound using the clock quantification operator so that one is reasoning about a “closed” program. While this approach is conjecturally sufficient to prove canonicity, most occurrences of Löb induction will not satisfy these requirements. In order to compensate for this fact, CloTT also includes an identification relating $\mathsf{loeb}(M)$ to its unfolding along with an equation forcing this identification to become reflexivity when loeb does compute.

Therefore, when one is using CloTT it is necessary to work with a version of Löb induction which does not unfold. Remarkably, the myriad of case studies carried out within CloTT show that this is not the show-stopper one might guess [MV21; VV20].[8]

We are therefore led to consider intensional guarded MTT extended with a constant for Löb induction and a further constant identifying $\mathsf{loeb}(M)$ with its unfolding. We refer to this system as *static guarded MTT* (sGuTT). In Section 9.6.1, we properly specify this system and use Chapter 8 to derive several important results regarding its metatheory. We also carry out an extended case study of a model of synchronous programming to demonstrate its usability even without a definitional unfolding for loeb.

[8] Indeed, until recently the Agda implementation of CloTT lacked the *tick constant* enabling any computation with loeb.

While sGuTT may suffice for constructing guarded programs and reasoning about them, there is no escaping the fact that it is ill-adapted to actually run programs. This situation parallels the fact that while the equations in CloTT may not be useful when reasoning, they are crucial for executing closed programs and obtaining canonical results. To calculate programs in sGuTT, we introduce a second system $\mathsf{DynGuTT}$ in which Löb induction always unfolds. This system certainly does not admit decidable type-checking, but we show in Sections 9.6.3 and 9.6.4 that it does admit a form of *guarded* canonicity.

These two systems combined give rise to a lightweight discipline for guarded recursive type theory: write programs in sGuTT and run them in $\mathsf{DynGuTT}$. The metatheory for both individually is unsatisfactory, but when combined they suffice.

9.6.1 Static guarded MTT

For technical reasons in the proof of guarded canonicity, it is simpler to limit sGuTT to a subset of $\mathcal{M}_{\mathsf{gtt}} \subseteq \mathcal{M}_{\mathsf{g}}$. In particular, we consider only a single mode t along with the two endomodalities representing earlier and later. In total, we have the following mode theory:

Definition 9.6.1. $\mathcal{M}_{\mathsf{gtt}}$ is the mode theory generated by one mode t, two modalities e and l, and four generating 2-cells: $\mathsf{id} \longrightarrow l$, $\mathsf{id} \longrightarrow l \circ e$, $e \circ l \cong \mathsf{id}$. We further quotient $\mathcal{M}_{\mathsf{gtt}}$ so that it is merely poset-enriched rather than a full 2-category.

We extend MTT instantiated with sGuTT with three additional principles: crisp identity induction for $\langle l \mid - \rangle$ (Section 6.3) along with the following two terms:

$$\frac{\Gamma.(l \mid A) \vdash M : A \mathrel{@} t}{\Gamma \vdash \mathsf{loeb}(M) : A \mathrel{@} t} \tag{9.10}$$

$$\frac{\Gamma.(l \mid A) \vdash M : A \mathrel{@} t}{\Gamma \vdash \mathsf{unfold}(M) : \mathsf{Id}(\mathsf{loeb}(M), M[\mathsf{id}.\mathsf{loeb}(M)]) \mathrel{@} t} \tag{9.11}$$

Crucially, we do not equip either loeb or unfold with any equations.

Definition 9.6.2. sGuTT is the type-theory obtained by extending MTT instantiated with $\mathcal{M}_{\mathsf{gtt}}$ with these three features.

As a first step, we note that sGuTT internalizes the same guarded situations as extensional guarded MTT.

Theorem 9.6.3. *The interpretation of the t mode in any model of extensional guarded MTT (such as $\mathbf{Sh}(\alpha)$) is a model of sGuTT.*

Proof. Extensional equality suffices to validate crisp identity induction (Lemma 6.3.4) so the result immediately follows. □

The second crucial feature of sGuTT comes from what it lacks compared to extensional guarded MTT. Having removed all additional definitional equalities available in the latter system, we may apply the results of Chapter 8 without additional effort.

Theorem 9.6.4. *Type-checking in sGuTT is decidable.*

Proof. Let us observe that a program in sGuTT is the same as a program in MTT with mode $\mathcal{M}_{\mathsf{gtt}}$ in a context extended with two additional variables (one for loeb and one for unfold). Accordingly, using Section 8.7, it suffices to argue that $\mathcal{M}_{\mathsf{gtt}}$ is decidable. The cases for 0-cells and 2-cells are immediate (both are propositional), so it remains only to show the (in)equality of modalities is decidable. Inspection shows that each modality in $\mathcal{M}_{\mathsf{gtt}}$ can be uniquely presented as $l^n e^m$ for some n, m. It remains to show that $l^{n_0}e^{m_0} \leq l^{n_1}e^{m_1}$ is decidable.

First, we observe that if this holds we must have $n_0 \leq n_1$: inspection on the model in $\mathbf{PSh}(\omega)$ shows that we can parlay the inequality $l^{n_0}e^{m_0} \leq l^{n_1}e^{m_1}$ into a morphism $\blacktriangleright^{n_0}\mathbf{0} \longrightarrow \blacktriangleright^{n_1}\mathbf{0}$ which only exists if $n_0 \leq n_1$. Let us therefore assume $n_0 \leq n_1$. In this case, we may transpose along $e \dashv l$ so that it suffices to consider $e^{n_1} \circ l^{n_0} \circ e^{m_0} = e^{m_0+(n_1-n_0)} \leq e^{m_1}$. This, in turn, holds if and only if $m_0 + (n_1 - n_0) \geq m_1$.

In total, $l^{n_0}e^{m_0} \leq l^{n_1}e^{m_1}$ if and only if $n_0 \leq n_1$ and $m_0 + (n_1 - n_0) \geq m_1$. This is clearly decidable. □

9.6.2 Programming in sGuTT

While Theorem 9.6.4 assures us that it is possible to implement sGuTT, this is only part of the picture. After all, sGuTT was deliberately chosen to contain only the portions of guarded type theory which were amenable to decidable type-checking. The real question was whether the resultant system would be expressive enough to actual carry out case studies. We demonstrate the utility of sGuTT through one such example: a reconstruction of the model of synchronous programming due to Boulmé and Hamon [BH01]. In so doing, we are able to simplify several of the constructions given in op. cit.: working in a guarded theory allows us to avoid some manual productivity checks which were required in order to encode the model in Coq.

Essentially our model of synchronous programming will interpret types in a simple first-order object language as a special stream of values. Unlike the streams described in Section 9.4, we will refine these streams by indexing them over a warp—an abstract representation of the rate of production of the stream.[9] Our reinterpretation of Boulmé and Hamon [BH01] will proceed in two steps. First, we will define *warped streams* as a type dependent on an ordinary guarded stream of booleans. Second, we will construct the suite of combinators on warp streams on warped streams to interpret a toy synchronous programming language.

We begin by recalling the definition of guarded streams.

$$\mathsf{GStream}\,A = \mathsf{loeb}(S.A \times \langle l \mid S\rangle)$$

Unlike before, however, we only have a propositional equality:

$$\mathsf{unfold}(S.A \times \langle l \mid S\rangle) : \mathsf{Id}(\mathsf{GStream}\,A, A \times \blacktriangleright\mathsf{GStream}\,A)$$

This is sufficient to define gcons, ghd and gtl with the expected types using transport, but we only have the β- and η-laws up to further propositional equalities:

$$\beta_0 : \mathsf{Id}(\mathsf{ghd}(\mathsf{gcons}\,a\,s), a) \qquad \beta_1 : \mathsf{Id}(\mathsf{gtl}(\mathsf{gcons}\,a\,s), t) \qquad \eta : \mathsf{Id}(s, \mathsf{gcons}\,(\mathsf{ghd}\,s)\,(\mathsf{gtl}\,s))$$

[9] In the literature, these objects are sometimes referred to as *clocks*. While it predates the terminology in clocked type theory, we opt to follow Guatto [Gua18] and refer to the objects from synchronous programming by this term.

As mentioned, for our purposes we are concerned about a particular class of streams:

$$\begin{array}{l} \mathbb{W} : \mathcal{U} \\ \mathbb{W} = \mathsf{GStream}\,\mathsf{Bool} \end{array}$$

With this definition of warps to hand, we can define the aforementioned type of warped streams. To facilitate doing so, however, we fix a few useful syntactic conveniences. First, when defining a function accepting an element of $\mathbb{W}$, we will use Agda-style pattern-matching syntax rather than directly using ghd and gtl. While translating between the two approaches is mechanical, they are equivalent only up to propositional equality. We will similarly allow ourselves to pattern-match on elements of $\langle \mu \mid - \rangle$ rather than explicitly using the eliminator for modal types. Second, rather than explicitly using loeb induction to define a guarded recursive function, we will simply use ordinary self-referential definitions. Once again, translating between the two styles is mechanical but computation rules are merely propositional.

Remark 9.6.5. In general, we will start to specify various notational conveniences for working with sGuTT in this section. This is a concession to the fact that writing terms in pure Martin-Löf type theory without the help of an elaborator is a chore. However, we strive to only include notations that could be supported by an elaborator such as implemented in e.g., Agda. Accordingly, these notations help maintain readability in these constructions without undermining the basic point that sGuTT would be usable if implemented. ◇

$$\begin{array}{l} \mathsf{WStream} : \mathcal{U} \to \mathbb{W} \to \mathcal{U} \\ \mathsf{WStream}\ A\ (\mathsf{gcons}(h, \mathsf{mod}_l(t))) = (\textbf{if}\ h\ \textbf{then}\ A\ \textbf{else}\ \mathsf{Unit}) \times \langle l \mid \mathsf{WStream}\,A\,t \rangle \end{array}$$

Notation 9.6.6. As it is defined by Löb induction, $\mathsf{WStream}$ unfolds only propositionally. We write $\mathsf{WStreamUnfold}$ for the proof that $\mathsf{WStream}$ is equal to its unfolding. More generally, given a guarded recursive definition foo, we will denote the associated unfolding proof $\mathsf{fooUnfold}$.

Intuitively, $\mathsf{WStream}\,w$ is a guarded stream that contains an element only when w is true. The degenerate case where $w = \mathsf{const}\,\mathsf{ff}$ is simply the unit type, while taking $w = \mathsf{const}\,\mathsf{tt}$ recovers the standard type of guarded streams.

The type of *head* and *tail* on warped streams are somewhat involved. In particular, the operation giving a first element of a stream is available only when the indexing warp indicates that the stream produces an element at this time. The result is that the types governing whd and wtl are fairly dependent:

$$\begin{array}{l} \mathsf{whd} : \{A : \mathcal{U}\}\{w : \mathbb{W}\} \to \mathsf{WStream}\,A\,(\mathsf{gcons}\,\mathsf{tt}\,w) \to A \\ \mathsf{whd}\,\{A\,w\}\ \textbf{rewrite}\ \mathsf{WStreamUnfold},\ \beta_1\,\mathsf{tt}\,w = \lambda(a,t).\ a \\ \\ \mathsf{wtl} : \{A : \mathcal{U}\}\{b : \mathsf{Bool}\}\{w : \mathbb{W}\} \to \mathsf{WStream}\,A\,(\mathsf{gcons}\,b\,w) \to \blacktriangleright\mathsf{WStream}\,A\,w \\ \mathsf{wtl}\,\{A\,b\,w\}\ \textbf{rewrite}\ \mathsf{WStreamUnfold},\ \beta_2\,b\,w\ = \lambda(a,t).\ t \end{array}$$

Remark 9.6.7. In order facilitate writing such dependent definitions, we have availed ourselves of a convenient **rewrite** syntax akin to what is present in Agda. Essentially, in order to ensure that the $\mathsf{WStream}$ type computes correctly we must transport it along the identification associated with loeb induction and then the proofs which have replaced the β- and η-laws for streams. ◇

We also have a version of cons, though it is parameterized by a boolean to allow us to choose whether or not we wish to append data:

$$\begin{aligned}
&\mathsf{wcons} : \{A : \mathcal{U}\}\{b : \mathsf{Bool}\}\{w : \mathbb{W}\} \\
&\quad \to (\textbf{if } b \textbf{ then } A \textbf{ else } \mathsf{Unit}) \to \blacktriangleright \mathsf{WStream}\, A\, w \to \mathsf{WStream}\, A\, (\mathsf{gcons}\, b\, w) \\
&\mathsf{wcons}\{A, b, w\}\ \ h\, t\ \textbf{rewrite}\, \mathsf{WStreamUnfold},\ \beta_1\, b\, w, \beta_2\ w = (h, t)
\end{aligned}$$

Finally, we will go one step further and allow ourselves to pattern-match on elements of $\mathsf{WStream}$. Once again, this can be justified only up to propositional equality but this is not an undue burden.

We are now in a position to construct the desired model of synchronous programming by providing implementations for a series of combinators on $\mathsf{WStream}$. Some are automatic in fact: loeb gives rise to exactly the correct fixed point combinator for $\mathsf{WStream}$ and, crucially, the presence of $\blacktriangleright$ allows us to avoid manually ensuring productivity in our system. In particular, this ensures that our definition of streams is substantially simplified from Boulmé and Hamon [BH01] where a "failure case" must be explicitly added. The following equivalence ensures that loeb can be used without any particular reference to $\blacktriangleright$ and with the standard formulation in synchronous programming:

$$\mathsf{WStream}\, A\, (\mathsf{gcons}\, \mathsf{ff}\, w) \simeq \blacktriangleright(\mathsf{WStream}\, A\, w)$$

The key remaining combinators are wconst, $\mathsf{wzipWith}$, and fby. Others are presented in op. cit., but these three suffice for examples and to illustrate the general pattern. Indeed, in all three cases, the machinery and notation presented thus far make their definitions straightforward:

$$\begin{aligned}
&\mathsf{wconst} : A \to \mathsf{WStream}\, A\, (\mathsf{const}\, \mathsf{tt}) \\
&\mathsf{wconst}\, a\ \textbf{rewrite}\ \mathsf{constUnfold} = \mathsf{wcons}\, a\, \mathsf{mod}_\ell(\mathsf{wconst}\, a)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{wzip} : (w_1\, w_2 : \mathbb{W})(A \to B \to C) \to \mathsf{WStream}\, A\, w_1 \to \mathsf{WStream}\, B\, w_2 \\
&\quad \to \mathsf{WStream}\, C\, (\mathsf{zipWith}\, \mathsf{and}\, w_1\, w_2) \\
&\mathsf{wzip}(\mathsf{gcons}\, \mathsf{tt}\, w_1)\, (\mathsf{gcons}\, \mathsf{tt}\, w_2)\, f\, (\mathsf{wcons}\, a\, s_1)\, (\mathsf{wcons}\, b\, s_2)\ \textbf{rewrite}\ \mathsf{zipWithUnfold} = \\
&\quad \mathsf{wcons}\, (f\, a\, b)\, (\mathsf{mod}_\ell(\mathsf{wzip}\, w_1\, w_2\, s_1\, s_2)) \\
&\mathsf{wzip}(\mathsf{gcons}\, \_\, w_1)\, (\mathsf{gcons}\, \_\, w_2)\, f\, (\mathsf{wcons}\, \_\, s_1)\, (\mathsf{wcons}\, \_\, s_2)\ \textbf{rewrite}\ \mathsf{zipWithUnfold} = \\
&\quad \mathsf{wcons} \star \mathsf{mod}_\ell(\mathsf{wzip}\, w_1\, w_2\, s_1\, s_2)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{fby} : \{w : \mathbb{W}\} \to A \to \mathsf{WStream}\, A\, (\mathsf{gcons}\, \mathsf{ff}\, w_1) \to \mathsf{WStream}\, A\, (\mathsf{gcons}\, \mathsf{tt}\, w_1) \\
&\mathsf{fby}\, a\, (\mathsf{wcons}\, \star\, s) = \mathsf{wcons}\, a\, s
\end{aligned}$$

The remaining connectives follow this pattern. Importantly, the fact that Löb induction does not unfold definitionally can be managed through careful use of common conveniences available in most proof assistants.

9.6.3 Dynamic guarded MTT

While the previous subsection shows that we are able to define a language for guarded recursion with decidable type-checking, sGuTT does not enjoy canonicity in any sense. This is hardly surprising: we have added several axioms to MTT without providing any computational justification. In this section, we introduce DynGuTT, a minimal extension of sGuTT with definitional equalities ensuring that unfold and loeb both compute. While this language does not have decidable type-checking (Theorem 9.5.9), we will show that it admits a form of *guarded canonicity* in Section 9.6.4.

Importantly, DynGuTT constitutes a small extension of sGuTT so that the majority of models of sGuTT are also models of DynGuTT. This allows us to write programs in sGuTT and run them in DynGuTT when attempting to reason about a model such as those introduced in Section 9.2.

DynGuTT extends sGuTT in two ways. The first we have already mentioned: a pair of definitional equalities governing loeb and unfold:

$$\frac{\Gamma.(l \mid A) \vdash M : A \mathbin{@} t}{\Gamma \vdash \mathsf{loeb}(M) = M[\mathsf{id}.\mathsf{loeb}(M)] : A \mathbin{@} t} \tag{9.12}$$

$$\frac{\Gamma.(l \mid A) \vdash M : A \mathbin{@} t}{\Gamma \vdash \mathsf{unfold}(M) = \mathsf{refl}(\mathsf{loeb}(M)) : \mathsf{Id}(\mathsf{loeb}(M), M[\mathsf{id}.\mathsf{loeb}(M)]) \mathbin{@} t} \tag{9.13}$$

Remark 9.6.8. We choose to keep the explicit term form unfold to make plain the embedding of sGuTT into DynGuTT. Of course, the added definitional equality ensures that it is merely shorthand for $\mathsf{refl}(\mathsf{loeb}(M))$ ◇

The other extension is more surprising. Briefly, we extend the contexts in DynGuTT with a new context former: $\mathbf{0}[\mu]$. This context is meant to represent the initial object under the modality μ. So, in particular, $\mathbf{0}[\mathsf{id}]$ is the initial object and trivializes all judgments: every type is uniquely inhabited and there exists a unique substitution out of $\mathbf{0} \triangleq \mathbf{0}[\mathsf{id}]$. The following rules govern this new form of context:

$$\frac{}{\vdash \mathbf{0}[\mu]\ \mathsf{cx}} \tag{9.14}$$

$$\frac{}{\mathbf{0}.\{\mu\} \vdash {!}\ \mathsf{type} \qquad \mathbf{0}.\{\mu\} \vdash {!} : A \qquad \mathbf{0}.\{\mu\} \vdash {!} : \Delta} \tag{9.15}$$

$$\frac{\Delta \vdash \rho : \mathbf{0}.\{\mu\} \qquad \Delta \vdash M : A \qquad \Delta \vdash \gamma : \Gamma}{\Delta \vdash A = {!}[\rho]\ \mathsf{type} \qquad \Delta \vdash M = {!}[\rho] : A \qquad \Delta \vdash \gamma = {!} \circ \rho : \Gamma} \tag{9.16}$$

$$\hom(\Gamma, \mathbf{0}[\mu]) \cong \hom(\Gamma.\{\mu\}, \mathbf{0}) \tag{9.17}$$

The first few rules ensure that $\mathbf{0}$ behaves like an initial object while simultaneously ensuring that it is preserved by $-.\{\mu\}$. The final rule can be seen as capturing several separate rules: two for transposing between substitutions $\Gamma \longrightarrow \mathbf{0}[\mu]$ and $\Gamma.\{\mu\} \longrightarrow \mathbf{0}$ and others for ensuring that these operations are inverses and natural. Stated even more concisely: $\mathbf{0}[\mu]$ represents the action of a right adjoint to $-.\{\mu\}$ on $\mathbf{0}$.

Remark 9.6.9. Semantically, $\mathbf{0}$ is interpreted by the initial object and so in the model within $\mathbf{PSh}(\omega)$, we interpret $\mathbf{0}[l^n]$ as $\blacktriangleright^n \mathbf{0}$. ◇

Remark 9.6.10. The rules governing $\mathbf{0}[\mathsf{id}]$ are similar to those of cubical type theory around the cofibration $0 = 1$. ◇

The reason for this addition—unlike the above definitional equalities—is *not* to facilitate additional computation. Rather, these contexts are added so that we may formulate guarded canonicity. Let us postpone this motivation for the moment to discuss the model theory of DynGuTT.

Using the same observations as with sGuTT, a model of DynGuTT is a model of sGuTT extended with some extra data, in particular:

Definition 9.6.11. A model of DynGuTT is a model $\mathcal{I}$ of sGuTT in which unfold is interpreted by reflexivity (and so loeb unfolds definitionally) together with the following:

- An initial object $\mathbf{0}$ in the category of contexts which is preserved by $\mathcal{I}(\mu)$ for all μ.
- The presheaves of terms and types are both orthogonal to $\mathbf{0} \longrightarrow \mathbf{y}(\mathbf{0})$.
- An object representing $\hom(\mathcal{I}(\mu)-, \mathbf{0})$ for each μ.

Corollary 9.6.12. *The syntax of DynGuTT is a model of sGuTT i.e. one may interpret sGuTT into DynGuTT.*

In practice, naturally-arising models of sGuTT will satisfy these requirements automatically. For instance, the first requirement is automatic in the presence of equality reflection. Furthermore, if the interpretation of modalities arises from adjunctions on the category of contexts (e.g., if the model is democratic Remark 7.2.3), both requirements reduce to the existence of an initial object in the category of contexts. We may package these requirements up as follows:

Theorem 9.6.13. *A model of sGuTT which validates equality reflection, possesses a strict initial object such that $\mathcal{T}(\mathbf{0}) = \mathcal{T}^{\bullet}(\mathbf{0}) = \mathbf{1}$, and where each modality is interpreted by a DRA arising from a weak CwF morphism is also a model of DynGuTT.*

Corollary 9.6.14. *The sheaf models of sGuTT given by Theorem 9.6.3 extend to models of DynGuTT.*

The force of this last theorem comes from the fact that these models of DynGuTT *extend* those of sGuTT. In a situation where we wish to use sGuTT as an internal language of these topoi, this result guarantees that we may freely compute using DynGuTT to simplify a troublesome equation away and return to working in sGuTT without compromising our ability to interpret the results in the intended model.

Guarded canonicity and $\mathbf{0}$ With these formal details in place, we return to $\mathbf{0}$ and guarded canonicity. We begin by taking a moment to discuss the broad idea of guarded canonicity. Typically, a canonicity statement for type theory says that each closed boolean is equivalent to tt or ff. In the case of DynGuTT, this is insufficient: we want to not only characterize the closed values of Bool, but also of ►Bool and ◄Bool. Contemplating the shape of closed forms for either of these, we see that we must consider not only a closed context but a context of the form $\mathbf{1}.\{l\}$ or similar. This, however, presents a secondary problem: what should the canonical form of an infinite stream be?

Given that we wish our canonicity theorem to apply to $\blacktriangleright^n\mathsf{Bool}$, one possible answer would be to allow for infinite canonical forms. This option is initially appealing but the technical challenges become apparent if we consider it in more depth. We must not only deal with canonical forms of types like streams but arbitrary guarded types. It is then unclear whether or not canonical forms can be characterized coinductively in such circumstances. Instead of attempting to produce a single canonical form characterizing all the information of a closed term, our result will produce an arbitrary finite approximation of a closed term. In spirit, this is similar to step-indexing: a program is run or verified with respect to some fuel.

Unlike with a programming language, however, $\mathsf{DynGuTT}$ is a type theory and therefore comes equipped only with a specified notion of equality. In particular, equality is not a directed reduction relation and so we cannot discuss what it means for a program to *run for n steps*. It is for this purpose that we have added $\mathbf{0}[\mu]$. Rather than specify a natural number to represent fuel, we will allow our canonicity theorem to apply only in contexts of the form $\mathbf{0}[\mu].\{\nu\}$ and this context will encode the fuel *intrinsically*. This "fuel" is not consumed by a program evaluating—this notion is no longer available—but rather as we descend further and further into a type more and more fuel is consumed.

A small example of the process is in order. Suppose we are given a term $\mathbf{0}[l] \vdash M : \langle l \mid A\rangle$, the guarded canonicity theorem will ensure that $M = \mathsf{mod}_l(M_0)$ where $\mathbf{0}[l].\{l\} \vdash M_0 : A$. At this point, however, guarded canonicity applies again but it is trivial: $\mathbf{0}[l].\{l\}$ has a map onto $\mathbf{0}$ and this trivializes the statement: all types are uniquely inhabited in this context. Intuitively, applying the canonicity theorem with $\mathbf{0}[l]$ as the starting context allows us to descend only beneath one $\blacktriangleright$. If we instead started with $\mathbf{0}[l^n]$, we could descend beneath n $\blacktriangleright$s before the terms trivialize. As a rough approximation, typically step-indexed fuel decreases as a program evaluates while this approach decreases fuel in conjunction with the type of the term. This approach is only possible with the addition of $\mathbf{0}[\mu]$; without it, there is no way to force the contexts to eventually trivialize.

In total, the formal statement for guarded canonicity is the following:

Theorem 9.6.15. *Given a term $\mathbf{0}[\mu].\{\nu\} \vdash M : A$, the following conditions hold:*

- *If $A = \mathsf{Nat}$ then $M = \bar{n}$ for some numeral n.*
- *If $A = \langle \xi \mid B\rangle$ then $M = \mathsf{mod}_\xi(N)$ where $\mathbf{0}[\mu].\{\nu \circ \xi\} \vdash N : B$.*
- *If $A = \mathsf{Id}(B, N_0, N_1)$ then $\mathbf{0}[\mu].\{\nu\} \vdash N_0 = N_1 : B$ and $M = \mathsf{refl}(N_0)$.*

9.6.4 Proving guarded canonicity

Remark 9.6.16. The material in this section is directly adapted from Section 4 and Appendices B and C of Gratzer and Birkedal [GB22]. The text has been modified to better fit this context as well as to integrate the proofs deferred to appendices in op. cit. back into the main text. ◇

After Chapters 4 and 8, it should be unsurprising that we will prove Theorem 9.6.15 using synthetic Tait computability. As in the prior proofs, we will construct a gluing cosmos modeling $\mathsf{DynGuTT}$ and use this to obtain the required result. Compared to the proof of canonicity of MLTT and the proof of normalization of MTT, two unique challenges present themselves: the interpretation of $\mathbf{0}$ and the interpretation of loeb.

Interpreting $\mathbf{0}$ poses some challenges: in Chapter 8, we introduced the notion of an MTT cosmos to simplify some aspects of the construction of the normalization model. While cosmoi axiomatize the salient features of presheaves over context categories, $\mathbf{0}$ does not have a simple universal property in this setting; the Yoneda embedding does not preserve initial objects. we rectify this by passing to certain sheaf subtopos in which $\mathbf{0}$ has the expected universal property. A similar localization is required for codomain of the functor we will glue along.

Interpreting loeb poses a challenge for a similar reason. The entire syntactic cosmos will not satisfy Löb induction—only terms—and so some care is required to set up the construction of the attendant computability structure.

Modulo these differences, the proof is similar to those we have encountered before. Accordingly, we will focus on the differences from Chapter 8 rather than belaboring points which have already been made there.

We begin with the revised definition of an MTT cosmos specialized to this setting an extended to include Löb induction:

Definition 9.6.17. A Löb cosmos is a strict 2-functor $G : \mathcal{M}_{\mathsf{gtt}} \longrightarrow \mathbf{Cat}$ such that $G(t)$ (which we abusively write G) is a locally Cartesian closed category with an initial object and each $G(\mu)$ is a right adjoint. We require the following additional structure:

1. A morphism $\tau_G : \mathcal{T}^{\bullet}_G \longrightarrow \mathcal{T}_G$ in G representing the universe of types and closed under all the connectives of MTT e.g. for each μ a map $G(\mu)(\mathcal{T}_G) \longrightarrow \mathcal{T}_G$ encoding the formation rule for modal types.
2. An element loeb with the appropriate type and necessary equation.

A morphism of Löb cosmoi is a 2-natural transformation of LCC functors satisfying Beck-Chevalley which preserves all structure.

In Chapter 8, could organize syntax into a cosmos enjoying a property akin to initiality by taking presheaves on the category of contexts and substitutions Cx, the same maneuver is not available here: $\mathbf{0}$ would no longer be initial when embedded into $\mathbf{PSh}(\mathsf{Cx})$. We therefore localize at $\mathbf{0}_{\mathbf{PSh}(\mathsf{Cx})} \longrightarrow \mathbf{y}(\mathbf{0}[])$ and consider *sheaves* over the category of contexts. Explicitly, we equip Cx with a Grothendieck topology J:

$$J(\Gamma) = \begin{cases} \{\hom(-,\Gamma), \emptyset\} & \text{if there exists a substitution } \Gamma \longrightarrow \mathbf{0} \\ \{\hom(-,\Gamma)\} & \text{otherwise} \end{cases}$$

The rules for $\mathbf{0}$ are organized to ensure that this Grothendieck topology is subcanonical and that the presheaves of types and terms are in fact sheaves for this topology. It is also easily seen that the precomposition functors induced by $-.\{\mu\}$ restrict to right adjoints on sheaves and that $\mathbf{Sh}(\mathsf{Cx})$ is closed under finite limits and dependent products in $\mathbf{PSh}(\mathsf{Cx})$.

Theorem 9.6.18. *There is a Löb cosmos $\mathcal{S}(t) = \mathbf{Sh}(\mathsf{Cx})$ where $\mathcal{T}$ (respectively $\mathcal{T}^{\bullet}$) are realized by the sheaves of types (respectively terms) and $\mathcal{S}(\mu)$ is given by precomposition with $-.\{\mu\}$.*

Passing to sheaves enables us to recover *quasi-projectivity* as in Theorem 8.3.5.

Theorem 9.6.19. *Given a Löb cosmos G and a map $\pi : G \longrightarrow \mathcal{S}$, the following holds:*

1. *For every context $\vdash \Gamma\,\mathsf{cx}$, there exists an object $[\![\Gamma]\!] : G$ and $\alpha_\Gamma : \pi([\![\Gamma]\!]) \cong \mathbf{y}(\Gamma)$.*
2. *For every type $\Gamma \vdash A\,\mathsf{type}$, there is a morphism $[\![A]\!] : [\![\Gamma]\!] \longrightarrow \mathcal{T}_G$ such that $\pi([\![A]\!]) \circ \alpha_\Gamma = \lfloor A \rfloor$.*
3. *For every $\Gamma \vdash M : A$, there exists $[\![M]\!] : [\![\Gamma]\!] \longrightarrow \mathcal{T}^\bullet_G$ over $[\![A]\!]$ such that $\pi([\![M]\!]) \circ \alpha_\Gamma = \lfloor M \rfloor$.*

Here $\lfloor - \rfloor$ is half of the isomorphism induced by the Yoneda lemma.

Just as with normalization we will now construct a particular Löb cosmos and use Theorem 9.6.19 to derive the theorem. We now turn to constructing this Löb cosmos by gluing the syntactic cosmos along a functor to a Grothendieck topos.

As in all gluing proofs, the choice of functor to glue along is crucial. For instance, when proving a standard canonicity result in Chapter 4 we used $\mathbf{PSh}(\mathsf{Cx}) \longrightarrow \mathbf{Set} = \mathbf{PSh}(\mathbf{1})$ given by precomposition with $\mathbf{1} \longrightarrow \mathsf{Cx}$. For normalization in Chapter 8 we worked with arbitrary contexts but normal forms are stable under a limited class of *renamings*. Accordingly, one glues along $\mathbf{PSh}(\mathsf{Cx}) \longrightarrow \mathbf{PSh}(\mathsf{Ren})$ given by precomposing with the inclusion $\mathsf{Ren} \longrightarrow \mathsf{Cx}$.

In our case, because we wish to prove a result about terms in context $\mathbf{0}[\mu].\{\nu\}$ we will take a category spanned by contexts of this form. Moreover, because guarded canonical forms are stable under the natural transformations $\mathbf{0}[\mu].\{\nu_0\} \longrightarrow \mathbf{0}[\mu].\{\nu_1\}$, we can recast this subcategory Cx spanned by contexts $\mathbf{0}[\mu].\{\nu\}$ as a partial order:

Definition 9.6.20. Define $(P, \leq)$ to be a partial order whose elements are pairs of modalities (μ, ν) such that $(\mu_0, \nu_0) \leq (\mu_1, \nu_1)$ if $\mu_0 = \mu_1$ and $\nu_1 \leq \nu_0$. There is a functor $i : P \longrightarrow \mathsf{Cx}$ sending (μ, ν) to $\mathbf{0}[\mu].\{\nu\}$

Unlike in prior gluing proofs, we represent syntax with $\mathbf{Sh}(\mathsf{Cx})$ rather than $\mathbf{PSh}(\mathsf{Cx})$. Accordingly, we must impose a Grothendieck topology on P so that the inclusion $P \longrightarrow \mathsf{Cx}$ induces a functor $\mathbf{Sh}(\mathsf{Cx}) \longrightarrow \mathbf{Sh}(P)$ and it is this functor that we will glue along.

Definition 9.6.21. Transporting the Grothendieck topology on Cx along the functor $i : P \longrightarrow \mathsf{Cx}$ yields a new topology on P covering (μ, ν) with the empty family if $\exists \xi.\ \mu \circ \xi \leq \nu$.

Lemma 9.6.22. *Precomposition by $(\mu, \nu) \mapsto (\mu, \nu \circ \xi)$ induces a right adjoint $R_\xi : \mathbf{Sh}(P) \longrightarrow \mathbf{Sh}(P)$.*

Lemma 9.6.23. *Recalling $\mathcal{S}(\xi) = (-.\{\xi\})^*$ from Theorem 9.6.18, $i^* \circ \mathcal{S}(\xi) = R_\xi \circ i^*$.*

Observe that for any pair (μ, ν), there exists n such that $\exists \xi.\ \mu \circ \xi \leq \nu \circ \ell^n$. Accordingly, given $X : \mathbf{Sh}(P)$ and $(\mu, \nu) : P$ there exists n such that $R^n_\ell(X)(\mu, \nu) = \{\star\}$. This eventual trivialization ensures that $\mathbf{Sh}(P)$ satisfies Löb induction:

Lemma 9.6.24. *For any $X : \mathbf{Sh}(P)$ there is a morphism $\mathsf{loeb}_X : X^{R_\ell(X)} \longrightarrow X$ satisfying the unfolding equation for Löb induction.*

Lemma 9.6.25. *Precomposition with i induces a right adjoint $i^* : \mathcal{S} = \mathbf{Sh}(\mathsf{Cx}) \longrightarrow \mathbf{Sh}(P)$.*

Gluing along i^*, we obtain a Grothendieck topos $\mathcal{G}(t) = \mathbf{Gl}(i^*)$ whose objects are triples $\big(X : \mathbf{Sh}(\mathsf{Cx}), Y : \mathbf{Sh}(P), f : Y \longrightarrow i^*X\big)$. Intuitively, these are proof-relevant predicates on syntax so that constructing a Löb cosmos in $\mathcal{G}$ is akin to a proof-relevant logical relation.

Lemma 9.6.26. *There is a strict 2-functor $\mathcal{G} : \mathcal{M} \longrightarrow \mathbf{Cat}$ sending $\mathcal{G}(t) = \mathbf{Gl}(i^*)$ and $\mathcal{G}(\mu)$ is determined component-wise by $\mathcal{S}(\mu)$ and R_μ. Furthermore, $\pi : \mathcal{G} \longrightarrow \mathcal{S}$ is a 2-natural transformation spanned by LCC functors and satisfying Beck-Chevalley.*

Proof. This follows from a slight variation on Theorem 8.4.15 and Lemma 9.6.22. □

It remains only to equip $\mathcal{G}$ with the structure of a Löb cosmos i.e. a universe $\tau_{\mathcal{G}} : \mathcal{T}^\bullet_{\mathcal{G}} \longrightarrow \mathcal{T}_{\mathcal{G}}$ closed under various connectives. In fact, this process is remarkably routine. $\mathcal{G}$ is a Grothendieck topos and therefore models extensional Martin-Löf type theory with a hierarchy of cumulative universes (Theorem 3.3.12) satisfying the realignment principle formulated by externally by Shulman [Shu15a] and internally by Orton and Pitts [OP18]. Moreover, $\mathcal{G}$ is a model of extensional MTT with a pair of complementary idempotent monads $\bigcirc$ and $\bullet$ defined by a proposition $\mathbf{syn}$ presenting $\mathbf{Sh}(\mathsf{Cx})$ (respectively $\mathbf{Sh}(P)$) as an open (resp. closed) subtopos. The structure of a Löb cosmos in $\mathbf{Sh}(\mathsf{Cx})$ yields a family of constants in this internal language. This combination of modalities shapes $\mathcal{G}$ into a model of *multimodal synthetic Tait computability* as described in Section 8.4. We note that in a context with $z : \mathbf{syn}$ we have the following constants manifesting some of the Löb cosmos structure on $\mathcal{S}$:

$$
\begin{aligned}
&\mathsf{Ty}(z) : \mathcal{U}\\
&\mathsf{Tm}(z) : (A : \mathsf{Ty}) \to \mathcal{U}\\
&\mathsf{loeb}(z) : (A : \mathsf{Ty}) \to (\langle l \mid \mathsf{Tm}\, A\rangle \to \mathsf{Tm}\, A) \to \mathsf{Tm}\, A
\end{aligned}
$$

While there are minor differences in the precise properties of multimodal synthetic Tait computability, this interpretation ensures that we can virtually replay the construction of a universe closed under the connectives of MTT given in Section 8.5. In particular, we can construct a universe of types and terms laying strictly over Ty and Tm:

$$
\begin{aligned}
&\mathbf{record}\ \mathsf{Ty}^* : \{\mathcal{U}_2 \mid z : \mathbf{syn} \mapsto \mathsf{Ty}(z)\}\ \mathbf{where}\\
&\quad \mathsf{code} : \bigcirc_z \mathsf{Ty}(z)\\
&\quad \mathsf{pred} : \{\mathcal{U}_1 \mid z : \mathbf{syn} \mapsto \mathsf{Tm}(z, \mathsf{code})\}
\end{aligned}
$$

$$\mathsf{Tm}^*(A) = A.\mathsf{pred}$$

Closing $(\mathsf{Ty}^*, \mathsf{Tm}^*)$ under the standard connectives of type theory is routine, so we focus on the last remaining novelty of the construction: Löb induction. Interpreting loeb hinges on the fact that $\mathbf{Sh}(P)$ and R_ℓ satisfy Löb induction. Lifting Lemma 9.6.24 this into the internal language of $\mathcal{G}$ gives us a variant of Löb induction:

$$\mathsf{loeb}_\bullet : \textstyle\prod_{A:\mathcal{U}}(\langle \ell \mid \bullet A\rangle \to \bullet A) \to \bullet A$$

This limited constant is sufficient to construct the proper interpretation of Löb induction:

Lemma 9.6.27. *There is a constant* $\mathsf{loeb}^* : (\langle \ell \mid \mathsf{Tm}^*(A)\rangle \to \mathsf{Tm}^*(A)) \to \mathsf{Tm}^*(A)$ *lying strictly over* loeb *and satisfying the unrolling rule.*

Proof. Let us fix $f : \langle \ell \mid \mathsf{Tm}^*(A)\rangle \to \mathsf{Tm}^*(A)$. We must construct $\mathsf{loeb}^*(f)$. We now use the fracture theorem: $A \cong \bigcirc A \times_{\bullet\bigcirc A} \bullet A$.

We have the left component of this pullback already: $\lambda z.\ \mathsf{loeb}(z, f)$. It remains to construct an element of $\bullet\mathsf{Tm}^*(A)$ which coheres appropriately with loeb. Here we use $\mathsf{loeb}_\bullet$ induction with the target $\bullet\mathsf{Tm}^*(A)$. We must produce a function $\langle \ell \mid \bullet\mathsf{Tm}^*(A)\rangle \to \bullet\mathsf{Tm}^*(A)$. We have $g = \bullet f : \langle \ell \mid \bullet A\rangle \longrightarrow \bullet A$. Therefore $\mathsf{loeb}_\bullet(g) : \bullet\mathsf{Tm}^*(A)$. We must show the following:

$$(\bullet\eta_{\bigcirc})(\mathsf{loeb}_\bullet(g)) = \eta_\bullet(\lambda z.\ \mathsf{loeb}(z, f)) \tag{9.18}$$

Let us prove this through Löb induction, available because equality of terms of $\bullet$-modal type is a $\bullet$-modal type. We then assume Eq. (9.18) under $\langle \ell \mid -\rangle$. Taking advantage of $\langle \ell \mid -\rangle$ as a fully-fledged dependent right adjoint, we rephrase this assumption as the equality

$$\mathsf{next}(\bullet\eta_{\bigcirc}(\mathsf{loeb}_\bullet(g))) = \mathsf{next}(\eta_\bullet(\lambda z.\ \mathsf{loeb}(z, f)))$$

We may simplify this by taking advantage of our ability to commute MTT modalities past those induced by gluing:

$$\bullet(\eta_{\bigcirc} \circ \mathsf{next})(\mathsf{loeb}_\bullet(g)) = \eta_\bullet(\lambda z.\ \mathsf{next}(\mathsf{loeb}(z, f))) : \bullet\bigcirc\langle \ell \mid \mathsf{Tm}^*(A)\rangle$$

Returning now to our goal, after applying both computation rules for the different forms of Löb induction, we are left with the following:

$$\bullet\eta_{\bigcirc}(g((\bullet\mathsf{next})(\mathsf{loeb}_\bullet(g)))) = \eta_\bullet(\lambda z.\ f(\mathsf{nextloeb}(z, f)))$$

Rewriting, we obtain

$$\bullet(\bigcirc f \circ \eta_{\bigcirc} \circ \mathsf{next})(\mathsf{loeb}_\bullet(g)) = (\bullet\bigcirc f)(\eta_\bullet\lambda z.\ \mathsf{next}(\mathsf{loeb}(z, f)))$$

The result now follows from our induction hypothesis. □

We are now able to conclude the fundamental lemma:

Theorem 9.6.28. *There is a Löb cosmos in* $\mathcal{G}$ *such that* $\pi : \mathcal{G} \longrightarrow \mathcal{S}$ *is a map of Löb cosmoi.*

Finally, from Theorems 9.6.19 and 9.6.28 combined with the definition of terms in $\mathcal{G}$ we can prove guarded canonicity.

Proof of Theorem 9.6.15. Fix a term $\mathbf{0}[\mu].\{\nu\} \vdash M : A$. By Theorem 9.6.19 we obtain an element of $M^* : A^*$ which lies over M up to isomorphism of contexts. We have the following square in $\mathbf{Sh}(P)$

$$\begin{tikzcd}
(F_\mu)_!(F_\nu(\mathbf{0}_{\mathbf{Sh}(P)})) \arrow[r] \arrow[d] & (\tau_{\mathcal{G}}[A^*])_0 \arrow[d, "\tau_{\mathcal{G}}[A^*]"] \\
i^*\mathbf{y}(\mathbf{0}[\mu].\{\nu\}) \arrow[r, "i^*\mathbf{y}(M)"'] & i^*(\tau_{\mathcal{S}}[A])
\end{tikzcd} \tag{9.19}$$

We instantiate this diagram of sheaves at (μ, ν). Let us construct an (μ, ν)-point of $(F_\nu)_!(F_\mu(\mathbf{0}_{\mathbf{Sh}(P)}))$. By functoriality, it suffices to construct a (μ, id) point of $F_\mu(\mathbf{0})$. Transposing, it is sufficient to construct a (μ, μ)-point of $\mathbf{0}$, which exists uniquely (it is a map between initial objects). Calculating, this lies over $\mathsf{id} : i^*\mathbf{y}(\mathbf{0}[\mu].\{\nu\})(\mu, \nu)$. This, together with the definition of the computability data for Nat, $\langle - \mid - \rangle$, and Id, yields the desired result. □

9.7 Conclusions

This chapter has surveyed several different approaches to guarded recursion within MTT. The major distinction between each approach is the balance it chooses to strike between faithfully capturing models like $\mathbf{Sh}(\alpha)$ and the extent to which the resulting calculus satisfies various metatheorems like canonicity and normalization. An unfortunate reality of guarded recursion is that there can be no calculus which does both simultaneously; Theorem 9.5.9 proves that faithfully including Löb induction will necessarily preclude decidable type-checking.

We have therefore presented two rather distinct approaches to guarded recursion within MTT: one using extensional MTT and therefore best suited to working on paper and one with ordinary MTT, but split into two systems. The examples carried out in the extensional system were first presented by Gratzer et al. [Gra+21], but the proofs here have been simplified and improved. Moreover, advances in the semantics of MTT allow us to interpret the calculus within sheaves over arbitrary ordinals rather than just ω. The examples and results governing the stratified system were first presented by Gratzer and Birkedal [GB22] but we have incorporated additional details in this thesis that were elided by op. cit.

While not discussed in this thesis, the question of internalizing guarded canonicity in some form remains a tantalizing prospect. In ongoing work with Jonathan Sterling and Lars Birkedal, we have shown that one may use multiple modes rather than multiple type theories to give a suitably internal version of the guarded canonicity theorem. While the result is still restricted by Theorem 9.5.9, the resulting system is conjectured to enjoy canonicity in a manner similar to clocked type theory.

10 A synthetic version of Iris

> It is now time to discuss the non-logical notions and to be led by this discussion even to further logical notions.
>
> Dana Scott
> *Advice on modal logic*

In Chapter 9, we explored how MTT could be instantiated in various ways to reason about guarded recursion. In this chapter, we continue the theme but with a particular focus on using (extensional) guarded MTT to encode a synthetic variant of Iris [Jun+18]. This development is explicitly separated into its own chapter so that it can better serve as a Rosetta stone for those knowledgeable about Iris who are curious about MTT and, conversely, modal type theorists who are curious about Iris. We begin by discussing what Iris is at a high level and what a synthetic account of the theory might entail.

Iris from the top down As mentioned briefly in Chapter 9, Iris is a *higher-order impredicative separation logic* designed with an eye towards concurrent languages with general references (mutable state that may contain arbitrarily complex data—including other references or functions).

A first and crucial point to make is that, in some sense, the idea that Iris is a logic is a pleasant fiction. Rather, Iris is a particular axiomatization of an intended model. The Coq formalization of Iris maintains strict control over which facts are added to the internal logic, so this illusion is not completely unfounded. However, there is no proof theory to speak of, merely a set of axioms and a Hilbert calculus to facilitate working with the intended model. In practice, therefore, Iris is used as a simply-typed lambda calculus with a specific type of propositions iProp modeling higher-order logic along with a handful of axioms and constants importing specific features of iProp from the model.

This sleight of hand is not unique to Iris, but what distinguishes Iris from many separation logics is the relatively small set of axioms that must be imported. If one were to inspect a typical statement in an Iris proof, the majority of the connectives being used are derived operations. Indeed, even the central connective for reasoning about programs $\mathsf{wp}\, e\, \{v.Q(v)\}$ and the rules for manipulating it are entirely derived.

In addition to those of higher-order logic, the primitive connectives and axioms in Iris fall into three classes:

1. A second symmetric closed monoidal structure $(*, -\!\!*)$ expressing *separating conjunction.*

2. A family of propositions $\mathsf{Own}(a)$ expressing ownership of some particular resource.
3. A handful of modal operators mediating and interpolating between Own, $*$, $-\!\!*$ and the other connectives of the logic.

One of the modalities referenced in point (3) is a propositional version of the $\blacktriangleright$ modality from Chapter 9. Indeed, the ambient simply-typed lambda calculus used to manipulate iProp is also extended with guarded primitives to permit guarded recursive definitions of propositions. Types are not quite presheaves on ω as one might expect from the preceding discussion of guarded recursion. Rather, they are drawn from a reflective subcategory of $\mathbf{PSh}(\omega)$ spanned by *flabby presheaves* or *complete ordered families of equivalences (COFEs)* in the parlance of Iris. The advantage of this subcategory is pragmatic: the data of such a presheaf is determined by a set together with an ordered collection of equivalences relations. This is easier to manipulate in Coq than a presheaf as e.g., a function between such objects is a single Coq function equipped with an additional property.

However, as Iris is manipulated in practice, the distinction drawn above is far from perfectly clear. The actual use of Iris in Coq mixes together standard Coq types, COFEs, standard Coq propositions, and elements of iProp. None of these levels can be entirely discarded and standard Coq types and propositions are used pervasively. For instance, a specification of an ordered queue in Iris will necessarily use the Coq type of lists, a Coq proposition to specify that such a list is ordered and at least one COFE—iProp—to parlay this information into the program logic.

Thus, while pen-and-paper presentations of Iris often specify it as a guarded higher-order logic of some sort, this is not truly faithful to how it is used in practice. The goal of synthetic Iris is then to more faithfully match how the system is used in practice by taking seriously the ambient language around iProp.

Remark 10.0.1. For those unfamiliar with separation logic, the following toy model may be helpful. Begin with the category of heaps H specified as a poset under the extension ordering and adjoin an initial object $\bot$ to obtain $H_\bot$. There is a symmetric monoidal structure $\otimes$ on this category given by taking the disjoint union of heaps and sending $h_1 \otimes h_2$ to $\bot$ if h_1 and h_1 are not disjoint or either is $\bot$. Passing to presheaves over H, Day convolution induces a closed symmetric monoidal structure and it descends to a similar structure on the sheaf topos $\mathcal{E}$ given by inverting the map $\mathbf{0} \longrightarrow \mathbf{y}(\bot)$.

It is within $\mathcal{E}$ that we have a model of separation logic. The closed symmetric monoidal structure is used to interpret $*$ and $-\!\!*$ and the primitive ownership propositions $\mathsf{Own}(a)$ are precisely the representables.

In particular, propositions are monotone predicates on heaps and $P * Q$ signifies that one may decompose a heap into two disjoint pieces such that one half satisfies P and the other satisfies Q. For instance, the following implication holds:

$$\mathsf{Own}(\{l_0 \mapsto v_0\}) * \mathsf{Own}(\{l_1 \mapsto v_1\}) \vdash l_0 \neq l_1$$

This simple picture is complicated by the realities of program verification. Realistic programs have invariants that cannot be fully described merely through the heap; acquiring a lock in a concurrent program grants ownership to some notion of resource but this resource cannot necessarily be specified merely in terms of the heap. To account for this, rather than taking presheaves on heaps we must take sheaves on a more general *partial commutative monoid*.

The situation is somewhat more complex than this may suggest: the aforementioned impredicativity of Iris ensures that the partial commutative monoid and resultant type of propositions must be defined simultaneously through a kind of domain equation. For this reason, Iris also includes guarded recursion (Chapter 9). In particular, the partial commutative monoid and iProp must be regarded as presheaves in $\mathbf{PSh}(\omega)$.

As a result, the real model of Iris cannot be constructed through such a direct procedure, but it is useful to keep the above model in mind in what follows. In some sense, the goal of a synthetic account of Iris is to parallel this story as closely as possible. ◇

Synthetic Iris—a prospectus Generally, the goal of synthetic mathematics is to replace the base objects of discourse—typically sets of some sort—with objects which carry the additional structure we wish to study and to ensure that every function preserves it. If we wish to study differential geometry, make every object a manifold and every map smooth. Likewise with domain theory, homotopy theory, or algebraic geometry.[1]

In the case of Iris, it is natural to arrange that all our types admit a guarded structure; this is what is typically claimed in pen-and-paper presentations of Iris after all. However, we will arrange for a full dependent type theory and, in particular, a universe of types and a universe of propositions. Correspondingly, all our functions will respect this guarded structure and so we will have Löb induction available by default at both a type and proposition level. For those already familiar with Iris, we will arrange that all types are (a generalization of) COFEs and all propositions belong to SProp. Unlike the Coq development, during this process, we have no need to work externally. In effect, this means that we never need to prove non-expansiveness propositions nor will we ever explicitly define a COFE structure on some type. Everything will be carried forward intrinsically. For those familiar with Chapter 9, we will work in guarded extensional MTT as discussed in Section 9.4.

On top of this layer, we will define and build up iProp and the familiar program logic. All of the relevant definitions—the generalization of partial commutative monoids to CMRAs, the solution to the domain equation, the definition of the programming language, etc.—will all be carried out within this guarded type theory. We shall see that this yields far more concise and conceptual descriptions than the unfolded versions familiar to Iris developers.

Adequacy There is one crucial place where some amount of external reasoning is required: adequacy. This is a fundamental result in Iris which relates a proof of specification of a program $\mathsf{wp}\,e\,\{v.Q(v)\}$ to the actual execution behavior of e. This result is fundamentally external: it is simply invalid given an internal proof $\mathsf{wp}\,e\,\{v.Q(v)\}$ and holds only for a global element of such a proof. In the parlance of Iris: adequacy is only applicable if we know that $\mathsf{wp}\,e\,\{v.Q(v)\}$ holds at all steps.

[1] Of course, this replacement cannot come without cost: if we are only able to define smooth functions we must ensure that e.g., we cannot write down an indicator function separating 0 from the rest of the interval. Typically, one isolates a class of maps with sufficiently good closure properties so that the requirement is not overly burdensome.

Fortunately, the machinery for handling such conditions without leaving type theory is precisely the subject of modal type theory. Accordingly, rather than developing Iris in just a version of MTT with the ▶ modality, we will also include an additional mode representing **Set** along with the adjunction giving rise to the global sections comonad. The adequacy result can then be formulated within this language as a property about $\square\mathsf{wp}\, e\, \{v.Q(v)\}$ rather than $\mathsf{wp}\, e\, \{v.Q(v)\}$.

Remark 10.0.2. While this is the only place within our definition that we require this additional modal structure, it could potentially be used for an additional purpose. In the formalization of Iris in Coq, it is typical to regard normal Coq datatypes as discrete COFEs of some form. We can crystallize this process synthetically with the help of these additional modalities. Normal Coq types are those types in the mode representing **Set** and the process of promoting them to discrete COFEs is half of the adjunction giving rise to □. In most cases, however, it is more convenient to directly define the type of e.g., lists observe a posteriori that it is discrete if this fact is necessary. ◇

Contents The remainder of this chapter is broken into several distinct sections. In Section 10.1 we give a more precise accounting of the ambient type theory we will use to construct our version of Iris. In Sections 10.3 to 10.5, we define the various components of the program logic within this language. Finally, in Section 10.6 we prove the promised adequacy result.

Remark 10.0.3. Throughout this chapter, we will make extensive use of the informal discipline for MTT introduced in Section 6.1 but we otherwise strive to make this chapter as self-contained as possible. In particular, while there is overlap with Chapter 9, the material there freely passed between informal and formal MTT. Here we shall consistently use only the informal discipline. ◇

10.1 A type theory for synthetic Iris

The ambient language for synthetic Iris will be an instantiation of MTT with a few caveats compared to the theory as it was introduced in Chapter 6. First and foremost, as we are committed to working on paper, it is convenient to work with extensional MTT.

The mode theory in use here is the mode theory for guarded recursion from Section 9.3. In particular, we will have two modes: one for guarded recursive types t and one for "normal" types s. These two modes are related to each other by a pair of modalities $d : s \longrightarrow t$ (vocalized *discrete*) and $g : t \longrightarrow s$ (*global*). Finally, the guarded mode t has an additional generating modality $l : t \longrightarrow t$ (*later*). Diagrammatically, the situation is represented as follows:

$$\begin{tikzcd}
t \arrow[loop left, "l"] \arrow[r, bend left, "g"] & s \arrow[l, bend left, "d"]
\end{tikzcd}$$

In addition to these three generating modalities, we require several 2-cells relating them to each other. However, in this instance we can identify all 2-cells with a shared boundary; if $\alpha, \beta : l \longrightarrow d \circ g \circ l$, for instance, then $\alpha = \beta$. As an immediate consequence, any 2-cell from μ to itself must be the identity. Accordingly, instead of naming 2-cells explicitly in this description we will just note the existence of a 2-cells from μ to ν by writing $\mu \leq \nu$ and this behaves like a partial order; transitivity and reflexivity follow from vertical composition and identity and antisymmetry follows from the above point. The generating inequalities are then as follows:

$$\mathsf{id} \leq l \qquad\qquad g \circ d = \mathsf{id} \qquad\qquad d \circ g \leq \mathsf{id} \qquad\qquad g \circ l = g$$

We note that the ability to horizontally compose 2-cells ensures that $\circ$ is monotone in both arguments: if $\mu_0 \leq \mu_1$ and $\nu_0 \leq \nu_1$ then $\nu_0 \circ \mu_0 \leq \nu_1 \circ \mu_1$.

Notation 10.1.1. As there is at most one choice of 2-cell for any given solution, we will not bother to write it out in general. Therefore, instead of writing x^α when accessing some variable, we will simply write x and omit the already determined 2-cell.

Notation 10.1.2. We will write the more familiar $\blacktriangleright A$ rather than $\langle l \mid A \rangle$. Note, however, that $\blacktriangleright A$ still only requires A to be a type after l-restricting the context; it is a dependent $\blacktriangleright$ modality.

We also explicitly extend MTT with Löb induction at mode t. That is, we add the following rules to the system explicitly:[2]

$$\frac{x :_l A \vdash M : A \mathrel{@} t}{\mathsf{loeb}(x.\, M) : A \mathrel{@} t}$$

$$\frac{x :_l A \vdash M : A \mathrel{@} t}{\mathsf{loeb}(x.\, M) = M[\mathsf{loeb}(M)/x] : A \mathrel{@} t}$$

Finally, we extend both modes of our language with an explicit universe of propositions $\Omega : \mathcal{U}$ and a decoding map $[-] : \Omega \to \mathcal{U}$. The defining property of Ω is that it classifies the elements of $\mathcal{U}$ which have only one element. We further require that Ω is *reflective* within $\mathcal{U}$. That is, there exists a map $L : \mathcal{U} \to \Omega$ and a map $\eta : A \to [L(A)]$ such that the following holds:

1. For each $\phi : \Omega$, the type $[\phi]$ satisfies the following:

$$(x\, y : [\phi]) \to x = y$$

2. If $A, C : \mathcal{U}$ such that $(c_0\, c_1 : C) \to c_0 = c_1$ then η induces an isomorphism between the following types:

$$(A \to C) \cong ([L(A)] \to C)$$

3. We further require that Ω satisfies propositional univalence. That is, if $[\phi] \cong [\psi]$ then $\phi = \psi$.

Remark 10.1.3. The second point in particular ensures that every type $A : \mathcal{U}$ with at most one element may be replaced by $[\phi]$ for some $\phi : \Omega$. In particular, we may take $\phi = L(A)$. ◇

[2] Equivalent rules were presented in Section 9.4, though we have chosen to present these in a more informal style to keep the presentation consistent throughout this section.

This universe is comparable to `SProp` in Coq or `Prop` in Agda—all elements are definitionally subsingleton—but unlike these, our strong defining property for Ω makes it e.g., impredicative and reflective in $\mathfrak{U}$. These additional properties suffice to *automatically* close it under all the connectives of higher-order logic e.g., $\forall$, $\wedge$, $\vee$, $\exists$, and similar.

We can construct inductively-defined propositions using either impredicative quantification or the reflector L. Propositional univalence ensures that there can be no ambiguity between the two choices, so we will not belabor the point.

We also note that Lemma 6.3.4 ensures that $\langle \mu \mid - \rangle$ sends propositions to propositions. So, for instance, if ϕ is a proposition at mode t in a g-restricted context, $\langle g \mid \phi \rangle$ is a proposition at mode s. As one might expect, we also require $[\langle \mu \mid \phi \rangle] = \langle \mu \mid [\phi] \rangle$ i.e. modal operators send propositions to propositions. In particular, there is a propositional version of $\blacktriangleright$ and the above equation ensures that it satisfies the appropriate propositional version of Löb induction.

Finally, we note that if μ is an internal left adjoint then it commutes with L (Lemma 6.4.16).

Definition 10.1.4. We refer to the total combination of these features as *the language of synthetic Iris.*

Remark 10.1.5. For those familiar with it, we will essentially regard Ω as the subobject classifier of each mode. For instance, the familiar comprehension operator $\{A \mid \phi\}$ is definable using ordinary dependent sums and $[-]$. ◇

Notation 10.1.6. We will generally omit $[-]$ and simply write $\phi : \mathfrak{U}$ when $\phi : \Omega$.

Theorem 10.1.7. *The language of synthetic Iris is sound and admits a model which interprets mode s as* **Set***.*

While it is perfectly possible to work with the above language completely axiomatically and rely on Theorem 10.1.7, it is helpful to have an intended interpretation of these constructs in mind. We present two explanations below, one for the working Iris practitioner and one for the more categorically minded.

Remark 10.1.8 *(The intended interpretation for Iris researchers).* The above language is meant to capture several salient aspects of the Coq formalization of Iris, with small adjustments to make the resulting type theory richer. In particular, one should imagine mode s as the mode of *standard* Coq types. These are just ordinary types and Ω at this mode is a version of Coq's `Prop` supplemented with a few additional axioms to make it more pleasant to work with on paper.

The t mode, on the other hand, is an enlargement of the universe of COFEs. Each type at mode t should be thought of as a (generalized) COFE and every term is a non-expansive map from the context to the type. The universe of propositions here is denoted `SProp` in the Coq formalization of Iris.[3] This ensures that we never need to directly address non-expansiveness or even contractiveness of maps: all maps are non-expansive and to model a contractive map $A \to B$ it suffices to construct a term $\blacktriangleright A \to B$.

The two generating modalities between s and t capture the two most common operations for passing between COFEs and ordinary types. The modality $\langle g \mid - \rangle$ forgets

[3] These are *step-indexed* propositions and should not be confused with *strict* propositions.

the guarded structure and gives back the underlying type of global points. In essence, it just erases the family of equivalences associated to a COFE. The modality in the reverse direction $\langle d \mid - \rangle$ gives an ordinary type a trivial COFE structure wherein each equivalence relation is simply equality.

With this interpretation to hand, one can easily check that the various identities of the modalities hold as expected. For instance, our mode theory requires $\langle g \mid \langle d \mid A \rangle \rangle \simeq A$ which states that taking a type A, equipping it with the trivial COFE structure and then forgetting this structure results in A which is apparent. We invite the reader to convince themselves that this interpretation validates $d \circ g \leq \mathsf{id}$ and $g \circ l = g$ as well.

As mentioned above types in mode t are properly understood to be generalized COFEs. In particular, recall that an ordinary COFE is a type A together with a family of equivalence relations R_n such that $R_n \supseteq R_{n+1}$. A type in mode t, by contrast, is a family of types A_n together with *restriction maps* $A_n \to A_{n-1}$. The gap is not quite as large as it may appear at first: one can obtain such a family from a COFE by setting $A_n = A/R_n$ and taking the maps $A_n \to A_{n-1}$ to be the induced maps on quotients. In fact, a COFE is precisely a generalized COFE whose restriction maps are surjective.

By allowing for non-surjective restriction maps for instance, we obtain a better-behaved version of the $\blacktriangleright$ modality which enables the use of Löb induction at types without first requiring them to be inhabited. Indeed, rather than merely shifting equivalence relations $\blacktriangleright A$ replaces A_0 with $\mathbf{1}$ and A_{n+1} with A_n. As a result, $\blacktriangleright\mathbf{0}$ is not a COFE even though $\mathbf{0}$ is. Aside from this somewhat technical point, it is acceptable to continue to regard types in t as COFEs of some sort. ◇

Remark 10.1.9 *(The intended interpretation for modal type theorists).* For those more familiar with the semantics of MTT (Chapter 7), we can give a more concise account of the intended model. We intend for mode t and s to be interpreted by $\mathbf{PSh}(\omega)$ and $\mathbf{Set}$, respectively. The modalities linking t and s are then given by adjunction induced by the discrete functor and the global elements functor. The modality $l : t \longrightarrow t$ is given by the $\blacktriangleright$ modality detailed extensively in Chapter 9. Tersely, it is the right adjoint to precomposition by $n \mapsto n + 1$. Finally, we note that as a presheaf topos, the discrete functor is both a left and a right adjoint and therefore all three of these functors are right adjoints. Theorem 7.2.12 then applies and induces the model.

The strict universe of propositions is interpreted by the subobject classifier in $\mathbf{Set}$ and $\mathbf{PSh}(\omega)$. That modalities send propositions to propositions is an immediate consequence of (1) the universal property of subobject classifiers and (2) the fact that all modalities are interpreted by right adjoints and therefore preserve monomorphisms. Finally, the validity of loeb induction is a well-known fact [Bir+12]. A direct proof is given in Theorem 9.4.3. ◇

10.2 A language for concurrency and references

Before developing a program logic, we need to specify the set of programs and their behavior. Iris is not tied to any specific programming language and it can accommodate languages with a wide range of features. We will copy the most widely used programming language with contains mutable references and `fork`-based concurrency.

It is at this point that we will work with the second mode s available in our language. This mode is intended to capture ordinary sets and therefore has no guarded content.

This quite directly reflects what is done already in the Coq formalization of Iris where the definition of the language and its operational semantics are done with ordinary Coq types—not COFEs. We will use the modalities relating s and t to lift our language and its operational semantics into the mode with guarded features in order to define the program logic (Section 10.5).

This separation gives two key advantages: first, it allows us to ensure that the programs and operational semantics interact correctly with the $\blacktriangleright$ modality, e.g. that they are timeless in the parlance of Iris. More importantly, it ensures that the adequacy theorem when unfolds in the intended model coincides with the standard definition.

We would otherwise have to convince ourselves that each object corresponded to what someone not using guarded type theory would expect.

Convention 10.2.1. Within this section, we work internally to s.

The language under question is an untyped version of PCF with mutable references and $\mathsf{fork}(-)$. The language itself is fairly standard and does not differ substantively from its presentation in Iris, so we will be relatively terse in our exposition. We present the grammar for the language below:

$$\begin{array}{lccl} \textit{(Expressions)} & e & ::= & x_i \mid \ell \mid e\,e \mid \mathsf{fixlam}(e) \mid e \leftarrow e \mid {!}\,e \mid \mathsf{ref}(e) \mid \mathsf{fork}(e) \\ & & & \mid e = e \mid \bar{n} \mid \mathsf{ifz}(e; e; e) \mid \mathsf{cas}(e; e; e) \mid \dots \\ \textit{(Values)} & v & ::= & \ell \mid \lambda\, e \mid \bar{n} \end{array}$$

We formally can organize this grammar into a pair of inductive types Exp and Val closed under the expected operators e.g., $- \leftarrow - : \mathsf{Exp} \to \mathsf{Exp} \to \mathsf{Exp}$. Unlike Iris, we have chosen to use De Bruijn indices to represent binding but we will not belabor the details of binding or substitution and simply use these operations without comment. We likewise note the evident inclusion $\mathsf{Val} \to \mathsf{Exp}$ which we will treat silently.

Remark 10.2.2. We note both Exp and Val are *countable*: we can form equivalences $\mathsf{Exp} \simeq \mathsf{Nat}$ and $\mathsf{Val} \simeq \mathsf{Nat}$ using e.g., some form of Gödel encoding. In particular, this ensures that $\langle d \mid \mathsf{Exp}\rangle$ and $\langle d \mid \mathsf{Val}\rangle$ are both inductive types closed under the expected operations in mode t. $\diamond$

The semantics of this language are specified through small-step operational semantics. This is complicated by two factors: the presence of concurrency and state. In order to accommodate the former, we will introduce two distinct “steps” relation. One of these will specify the process of stepping a single expression by a single step relating an input expression to the reduced expression and the collection of expressions *forked off* during this step. This relation is then used by a secondary stepping relation which steps a thread pool to another thread pool. To account for state, both of these relations will manipulate a heap σ mapping a finite set of locations to values.

Definition 10.2.3. Define $\mathsf{Heap} = \mathsf{Nat} \to_{\mathsf{fin}} \mathsf{Val}$ to be the type of partial maps $\mathsf{Nat} \to \mathsf{Val}$ with (specified) finite support. We write $\mathsf{alloc} : \mathsf{Heap} \to \mathsf{Nat}$ for the function which sends a heap to the smallest number not in the support of the input. We write $\mathsf{extend} : \mathsf{Heap} \to \mathsf{Nat} \to \mathsf{Val} \to \mathsf{Heap}$ for the function which extends a heap to map some supplied natural number to a supplied value.

The operational semantics are then specified by the following binary relations:

$$(\mapsto) : \mathsf{Exp} \times \mathsf{Heap} \to \mathsf{Exp} \times \mathsf{Heap} \times \mathsf{List}\,\mathsf{Exp} \to \Omega$$

$$(\rightsquigarrow) : \mathsf{List\ Exp} \times \mathsf{Heap} \to \mathsf{List\ Exp} \times \mathsf{Heap} \to \Omega$$

These relations are both specified inductively—recall that Ω models higher-order logic and so constructions such as the Knaster-Tarski theorem are available [BL13]. Rather than specifying the full litany of inferences for these relations, we present a representative few and refer the reader to e.g., Jung et al. [Jun+18] or Bizjak and Birkedal [BB22] for more details.

$$\frac{\ell = \mathsf{alloc}\,\sigma}{(\mathsf{ref}(v), \sigma) \mapsto (\ell, \mathsf{extend}\,\sigma\,\ell\,v, \epsilon)} \qquad \frac{\ell \in \mathsf{dom}\,\sigma}{(\ell \leftarrow v, \sigma) \mapsto (v, \mathsf{extend}\,\sigma\,\ell\,v, \epsilon)}$$

$$\frac{\ell \in \mathsf{dom}\,\sigma}{(!\,\ell, \sigma) \mapsto (\sigma(\ell), \sigma, \epsilon)} \qquad \frac{}{(\mathsf{fork}(e), \sigma) \mapsto (\bar{0}, \sigma, [e])}$$

$$\frac{}{(\mathsf{fixlam}(e)\,v, \sigma) \mapsto (e[\mathsf{id}.\mathsf{fixlam}(e).v], \sigma, \epsilon)}$$

$$\frac{\vec{e}_0, e, \vec{e}_1 = \vec{e} \qquad (e, \sigma) \mapsto e', \sigma', \vec{e}_2}{(\vec{e}, \sigma) \rightsquigarrow (\vec{e}_0, e', \vec{e}_2, \vec{e}_1, \sigma')}$$

Remark 10.2.4. For those unfamiliar with `HeapLang`, we note that both recursion and functions are bundled up into a single expression $\mathsf{fixlam}(-)$ which defines a recursive function. ◇

Remark 10.2.5. We note that there is no static type-system associated with `HeapLang`. We shall rely fully on the program logic to establish the runtime properties of a program. ◇

As is standard, we write $\mapsto^*$ and $\rightsquigarrow^*$ reflexive transitive closures of $\mapsto$ and $\rightsquigarrow$, respectively.

Notation 10.2.6. In what follows, we will frequently have occasion to take a closed term foo at mode s with the type $A \to B$ (both closed) and regard it as a function $\langle d \mid A \rangle \to \langle d \mid B \rangle$. Explicitly, we could write $\mathsf{mod}_d(\mathsf{foo}) \circledast -$ but this is somewhat unwieldy. We will therefore write $\mathsf{foo}^\dagger$ as shorthand for this expression, though we stress that this is only valid when foo is closed; otherwise $\mathsf{mod}_d(\mathsf{foo})$ may be ill-formed.

Notation 10.2.7. Given an infix operator such as $\mapsto$, we write $x \mapsto^\dagger y$ as infix notation for $(\mapsto)^\dagger\, x\, y$.

10.3 Cored resource algebras

The first ingredient in our account of Iris is a recasting of CMRAs (*cameras*) [Jun+18] into this synthetic setting. Briefly, a CMRA in classical Iris extends the notion of a commutative monoid in the following ways:

- The carrier set is replaced by a COFE and multiplication is non-expansive.
- Multiplication is allowed to be partial

- Rather than asking for a single unit, a partial idempotent operation termed *core* gives a partial assignment of unit for each object.

The last change is perhaps the most surprising. In more detail, a CMRA M does not necessarily have a single unit for multiplication. Rather, there is an operation $|-| : M \longrightarrow M$ which, when defined, satisfies the following:

$$|x| \cdot x = x$$

Some other aspects of this definition are motivated by the practicalities of formalization. For instance, multiplication and core are partial operations but the encoding of partiality is not identical: multiplication is represented as a total operation but equips M with a predicate (*valid*) delineating between defined and undefined elements of M while $|-|$ sends M to $1 + M$ and which sends valid elements to either the left disjunct or a valid element. As we shall see, in some circumstances having the support of $|-|$ be decidable is crucial but some structure does not require it. Moreover, partiality for multiplication interferes with $|-|$ in a subtle way: as a function on M, the latter is defined even for those invalid elements representing undefined multiplications.

When considering a synthetic reformulation of CMRAs, we therefore have two classes of decisions to make: how will we encode the salient mathematical aspects of CMRAs in this new language and to what extent should the synthetic definition preserve aspects of CMRAs motivated by Coq. We choose not to preserve these aspects of the definition in our setting—indeed, what is practical in Coq is not usually what is practical on paper—and therefore focus on capturing the important details on CMRAs as concisely as possible.

Convention 10.3.1. In this section, we will work exclusively within the language of synthetic Iris at mode t.

We will decompose our version of CMRAs into two halves: a structure with a partial multiplication operation and a core operation. Both core and multiplication are partial and we therefore take a moment to recall the *partial map classifier* and its role in encoding partial maps.

Definition 10.3.2. We define the partial map classifier $-^? : \mathcal{U} \longrightarrow \mathcal{U}$ as follows:

$$A^? = \textstyle\sum_{\phi:\Omega}[\phi] \to A$$

An element of $A^?$ consists of two pieces of data: a proposition ϕ and a *partial* element of A available only when ϕ is true. It is instructive to consider the extreme hypothetical where $\Omega = \mathsf{Bool}$. In this case, $A^?$ degenerates to $\mathbf{1} + A$ so an element of $A^?$ consists either of a genuine element of A or a formally adjoined “undefined” value. In reality, Ω contains many values between $\top$ and $\bot$ so $A^?$ is slightly richer, but the same intuition applies: a map into $A^?$ is exactly a partial map into A. The first component dictates where the map is defined and the second component is the (suitably partially defined) result.

Remark 10.3.3. It is standard to define a partial map from A to B in a category to consist of a subobject of A_0 of A together with a map from A_0 to B. It is clear that this data is precisely captured by $A \to B^?$. Indeed, the first projection yields a subobject via

$\phi : A \to \Omega$ and the assumption of ϕ in the second component ensures that the second component consists of a map out of this subobject to B. ◇

Notation 10.3.4. Given $x : A^?$ we write $x\!\downarrow$ for $\pi_1\, x$

Theorem 10.3.5. *The partial map classifier $-^?$ is a monad on $\mathcal{U}$.*

Proof. As this is a classical fact, we content ourselves with merely recalling the unit and join operations and refer to the literature for a verification of the laws:

$$\begin{aligned}&\eta : A \to A^? \\ &\eta\, a = (\top, a)\end{aligned}$$

$$\begin{aligned}&\mu : (A^?)^? \to A^? \\ &\mu\,(\phi, a^?) = ((\exists z : \phi.\, \pi_1(a^?\, z)), \lambda(z_1, z_2).\, \pi_2\,(a^? z_1)\, z_2)\end{aligned}$$

In the definition of join, we have used informal pattern-matching notation for existential propositions rather than elaborating to a definition using the impredicative encoding given earlier directly. □

In particular, we can use "do notation" to work with partially defined elements by writing e.g. $y \leftarrow f\,x; g\,y$ for the composition of two partial maps f and g.

Lemma 10.3.6. *The universe of propositions Ω supports a $(-)^?$-algebra structure which we denote $\mathsf{Part} : \Omega^? \longrightarrow \Omega$.*

Proof. We define $\mathsf{Part}(\phi, \psi) = \forall z : \phi.\, \psi(z)$. It is routine to show that this satisfies the algebra laws using propositional univalence. □

The partial map classifier gives us a uniform way to encode partial operations. To pass from a totally defined operation multiplication $A \times A \to A$ to a partial operation, we change the type to $A \times A \to A^?$.

Definition 10.3.7. A partial commutative semigroup is a type $A : \mathcal{U}$ equipped with a commutative operation $(\cdot) : A \times A \to A^?$ satisfying the following version of associativity:

$$(a_0\, a_1\, a_2 : A) \to (a_{01} \leftarrow a_0 \cdot a_1; a_{01} \cdot a_2) = (a_{12} \leftarrow a_1 \cdot a_2; a_0 \cdot a_{12})$$

Notation 10.3.8. For convenience, we will occasionally write $a_0 \cdot a_1 \cdot a_2$ as shorthand for $a_{01} \leftarrow a_0 \cdot a_1; a_{01} \cdot a_2$.

Lemma 10.3.9. *A partial commutative semigroup structure on A is equivalent to a commutative semigroup structure on $A^?$.*

Corollary 10.3.10. *Given a partial commutative semigroup $(A, \cdot)$, the extension order $\sqsubseteq$ on the commutative semigroup $A^?$ restricts to one on A. Explicitly, $a \sqsubset b$ if there exists $c : A$ such that $a \cdot c = \eta\, b$.*

Notation 10.3.11. Note that as $A^?$ is merely a commutative semigroup, $\sqsubset$ is not generally reflexive; without a unit there need not be some b such that $a \cdot b \sqsubseteq a$. Jung et al. [Jun+18] note that the main use of $\sqsubset$ is reflexive and therefore accept the definition as-is. We will take a slightly different approach and write A_+ for $A + \mathbf{1}$. We then extend $\cdot$ to a function $A_+ \to A \to A^?$ by setting $\mathsf{in}_2(\star) \cdot - = \mathsf{id}$ and write $a \sqsubseteq b$ for the formula $\exists s : A_+.\, s \cdot a = \eta\, b$. This is the reflexive completion of the extension order defined above.

Definition 10.3.12. A core structure on a partial semigroup $(A, \cdot)$ is a partial function $|-| : A \to A^?$ satisfying the following properties:

$$(a : A) \to \mathsf{Part}(c \leftarrow |a|; \eta(c \cdot a = \eta\, a)) \qquad (a : A) \to \mathsf{Part}(c \leftarrow |a|; \eta(\eta\, c = |c|))$$
$$(a_0\, a_1 : A) \to a_0 \sqsubseteq a_1 \to \mathsf{Part}(c_0 \leftarrow |a_0|; \eta(\eta\, c_0 \sqsubseteq |a_1|))$$

Roughly, these properties ensure that $|-|$ is a local unit, idempotent, and monotone with respect to the extension order. We refer to such a partial semigroup as a *resource algebra (RA)*.

We will take cored resource algebras as our synthetic version of CMRAs. We note that by virtue of our approach, RAs are intrinsically step-indexed. There is no need to ensure that multiplication and coring are non-expansive operations, for instance, because all operations are non-expansive. We further note that we have unified the two disparate approaches for partiality into the uniform abstraction provided by the subobject classifier. As a result, we have dropped the validity predicate and the requirement that $|-|$ have a decidable image. We shall find that the latter must be reintroduced in certain specific places but we will explicitly mark where it is required.

Remark 10.3.13. Aside from the differences in partiality, we note that the synthetic approach forces us to take (generalized) COFEs as the carriers of a RA. In Iris, it is a helpful generalization to consider CMRAs on OFEs—lacking a completeness requirement. In our setting, an OFE consists essentially of (1) a type A at mode t and (2) a type A_g at mode s equipped with a map $A_g \to \langle g \mid A \rangle$. In particular, it is additional data on top of a type at mode t and therefore somewhat awkward to use in a formulation of resource algebras. Fortunately, the major motivations for the switch in Iris are obviated by the switch to more general COFEs. ◇

Remark 10.3.14. For those not familiar with Iris, some foreshadowing as to why RAs are a reasonable choice of abstraction is in order. We will eventually fix one RA R and interpreted propositions as monotone predicates $R \to \Omega$. Within this setting, the "representable" predicates have a distinguished role and are used to codify ownership of a resource. Within this setting, owning two representables will—roughly—be equivalent to owning their product. The partiality allows us to express when such combinations should be disallowed. In particular, if $x, y : R$ should not be possible to "own" simultaneously, their product should be undefined. ◇

10.3.1 Examples of RAs

We now introduce a variety of RAs both to build intuition and for future use in the development of synthetic Iris. Unlike standard Iris, we begin by defining the (internal) category of RAs.

Definition 10.3.15. A homomorphism of RAs $f : A \longrightarrow B$ consists of a function between the carrier types satisfying the following additional properties:

$$(a_0\, a_1 : A) \to \mathsf{Part}(a_{01} \leftarrow a_0 \cdot a_1; \eta(\eta(f\, a_{01}) = f\, a_0 \cdot f\, a_1))$$
$$(a : A) \to \mathsf{Part}(c \leftarrow |a|; \eta(\eta(f\, c) = |f\, a|))$$

Intuitively, a morphism of RAs preserves both multiplication and core when they are defined.

It is a routine calculation to show that the identity function is a RA homomorphism and that RA homomorphisms compose. In particular, we may define the category of RAs as follows:

Definition 10.3.16. The category of RAs **RA** has RAs for objects and RA homomorphisms for morphisms.

Lemma 10.3.17. RA *is closed under finite products and coproducts.*

Proof. The case for finite products is essentially pointwise, so we show only the case for coproducts. First of all, the initial object of **RA** is given by the empty type **0** endowed with the (necessarily unique) RA structure.

We therefore must only show that **RA** is closed under binary coproducts. To this end, fix RAs A and B. We will endow $A + B$ with the structure of a resource algebra as follows:

$$\mathsf{in}_i(x) \cdot \mathsf{in}_j(y) = \begin{cases} z \leftarrow x \cdot y; \eta\, \mathsf{in}_i(x) & i = j \\ \perp & i \neq j \end{cases}$$

We must show that this operation is associative and commutative and both of these properties follow immediately from the relevant properties of A and B.

We define the core operation pointwise:

$$|\mathsf{in}_i(x)| = c \leftarrow |x|; \eta\, \mathsf{in}_i(c)$$

Once more the relevant properties for $|-|$ follow from those of A and B. It is similarly routine to check that the inclusions $A \to A + B$ and $B \to A + B$ are RA homomorphisms.

It remains to show that $A + B$ is indeed the coproduct in **RA**. Fix a RA C along with morphisms $f : B \to C$ and $g : B \to C$, we must show that there is a unique map $A + B \to C$ extending f and g. First note that since f and g induce morphisms of carriers, there is at most one RA homomorphism satisfying this property. It remains to check that this map $[f, g]$ is a RA morphism. We must show that it preserves multiplication and core when they are defined, but this follows precisely from the fact that f and g are RA morphisms. □

We now turn to two classic examples of RA: the exclusive RA and the agreement RA. Each captures an important future use of RAs and two ways of viewing the partiality of the operations. The exclusive RA enforces ownership and the partiality of multiplication reflects this: multiplication is always undefined because there is no consistent situation where two "tokens" representing exclusive access can coexist. In particular, the exclusive ownership is the left adjoint to the forgetful functor $\mathbf{RA} \longrightarrow \mathbf{Type}$ where the latter is the standard category of small types and functions.

Lemma 10.3.18. *The forgetful functor* $\mathbf{RA} \longrightarrow \mathbf{Type}$ *has a left adjoint* Excl.

Proof. Define $\mathsf{Excl}\, A$ to have the carrier A and totally undefined multiplication and $|-|$. In particular, there are no additional properties to check on these operations because all relevant properties apply only when these operations are defined. Moreover, a homomorphism $\mathsf{Excl}\, A \to B$ is merely a map between the carriers for the same reason. □

Remark 10.3.19. In fact, $\mathbf{RA} \longrightarrow \mathbf{Type}$ factors through the category of partial commutative semigroups. The functor from RAs to partial commutative semigroups also admits a left adjoint equipping each RA with a core that is always undefined. We will identify a partial commutative semigroup with its image under this functor and regard it as a RA with a trivial core in some situations. $\diamond$

Definition 10.3.20. An idempotent element r of a RA R is one where $\eta\, r = |r|$ i.e. the core of r is totally defined and equal to r itself. We write R^{idem} for the subset of idempotent elements of R.

Lemma 10.3.21. *Given a type X, the agreement RA $\mathsf{Ag}\, X$ is the unique RA such that* $\hom(\mathsf{Ag}\, X, R) \cong (X \to R^{\mathsf{idem}})$*. Equivalently,* Ag *is the unique colimit-preserving functor* $\mathbf{Type} \longrightarrow \mathbf{RA}$ *such that* $\mathsf{Ag}\, \mathbf{1}$ *corepresents* $-^{\mathsf{idem}} : \mathbf{RA} \longrightarrow \mathbf{Type}$.

Proof. We define the carrier of $\mathsf{Ag}\, X$ to be X. The core operation is simply η (so core is in particular total). Multiplication is slightly more involved:

$$x \cdot y = (x = y, \lambda \_ .\, x)$$

Informally, $x \cdot y$ is partially defined with support $x = y$. When it is defined—that is, when $x = y$—it is simply x. It is routine to show that $\cdot$ is commutative and associative. It remains to argue that $|-|$ has the expected properties. As $|-|$ is η, computation shows that it is idempotent and a local unit. It remains to show that $|-|$ respects the extension order, but in this RA the extension order collapses to equality. Therefore, $|-|$ is immediately seen to respect extension ordering.

Finally, we turn to the universal property of $\mathsf{Ag}\, X$. A morphism $f : \mathsf{Ag}\, X \to A$ contains a map of $f : X \to A$. Moreover, each $f\, x$ must satisfy $|f\, x| = \eta(f\, x)$ ($|-|$ is totally defined on $\mathsf{Ag}\, X$ and therefore f must respect it). In fact, this condition is also sufficient: multiplication in $\mathsf{Ag}\, X$ is defined just when is of the form $x \cdot x = |x| \cdot x$. $\square$

Definition 10.3.22. A *unital* RA R is a RA together with a (necessarily unique) element $\epsilon : R$ such that $\epsilon \cdot - = \eta : R \to R^?$ and $|\epsilon| = \eta\, \epsilon$.

Unital RAs play an important role in Iris for two reasons. Firstly, the associated theory of propositions (Section 10.4) is simpler for unital RAs. More importantly, however, the most important construction on RAs—the “direct sum” operation—produces a unital RA.

This construction arises when we attempt to define the special universe of propositions used for the program logic (in Section 10.4). There, we will have to fix a particular RA to instantiate things with but frequently we will want to use more than one RA in actual proofs. To thread this needle, we will require a construction mixing two (or more) RAs into a single RA. We have already encountered two procedures for doing this: sums and products. However, neither of these is really satisfactory for our purposes. We wish to have an operation that allows us to combine two RAs in a more “independent” way.

Given a collection of RAs $R : I \to \mathbf{RA}$, we want a single RA $\bigoplus R$ such that:

1. for each $i : I$, there is an inclusion of RAs $\mathsf{in}_i : R(i) \to \bigoplus R$,
2. such that $\mathsf{in}_i\, x \cdot \mathsf{in}_j\, y$ is defined for all $x : R\, i$ and $y : R\, j$ such that $i \neq j$.

In practice, we also have an additional requirement that is harder to articulate: we would like for it to be the case that $R\,i$ and the RA generated by the image of in_i are closely related. Unfortunately, it is impossible to arrange for them to be isomorphic; it is easy to arrange for a situation where the core operation on the image of in_i is total even when the same is not true on $R\,i$. The best we can hope for is that this is the only difference between $R\,i$ and its image. We can capture this fact in several ways, but the most reasonable is to ask that $\bigoplus R$ be unital.

To construct the direct sum, it will be necessary to assume that each $R\,i$ has a decidable core. In practice, this is not an issue and it is actually required of all RAs in Iris. The decidability of core is crucial for the following construction:

Lemma 10.3.23. *If X is a RA whose core operation has decidable support, there exists a unital RA X_+ along with a map $X \longrightarrow X_+$.*

Proof. We take $X_+ = \mathbf{1} + X$ and define multiplication to treat the left-hand disjunct as a global unit. The definition of $|-|$ is slightly more subtle, we set $|\mathsf{in}_1(\star)| = \mathsf{in}_1(\star)$ while $|\mathsf{in}_2(x)| = |x|$ if the latter is defined and $\mathsf{in}_1(\star)$ if it is not. This slightly convoluted definition is necessary to ensure that $|-|$ is monotone with respect to the extension order. □

Lemma 10.3.24. *Fix a type I with decidable equality and a family of RAs with decidable cores $R : I \to \mathbf{RA}$. Under these assumptions, one can construct $\bigoplus R$.*

Proof. We content ourselves with only defining $\bigoplus R$ and leave the verification of its universal property to the reader. We take the carrier of $\bigoplus R$ to be the subtype of maps $(i : I) \to R(i)_+$ with finite support:

$$\bigoplus R = \{f : (i : I) \to R(i)_+ \mid \exists m > 0, l : [m] \to I.\, \forall i : I.\, f\,i = \epsilon \iff i \notin \mathsf{im}(l)\}$$

We emphasize that because this is a subtype of dependent functions, we are not able to depend on the particular labeling function $l : [m] \to I$ but—since m exists uniquely—we are able to use m. This distinction will prove crucial for defining the *gap map* witnessing the universal property of $\bigoplus R$.

We must endow this map with suitable core and multiplication operations. We begin with multiplication. First, notice that because I has decidable equality, given two maps $f : [m_1] \to I$ and $g : [m_2] \to I$ there exists a map $h = f \uplus g : [k] \to I$ which labels the union of the images of f and g. This map is not unique. Accordingly, fix $x = (f_1, l_1 : [m_1] \to I)$ and $y = (f_2, l_2 : [m_2] \to I)$ and write $l : [m] \to I$ for $l_1 \uplus l_2$. We define the xy as follows:

```
$xy =$
    $x_0 \leftarrow f_1(l\,0) f_2(l\,0);$
    $\ldots$
    $x_{m-1} \leftarrow f_1(l(m-1)) f_2(l(m-1));$
    $\eta\,(\{l\,k \mapsto x_k\}_{0 \le k < m}, l)$
```

In the above, $\{l\,k \mapsto x_k\}$ represents the function which returns x_k on $l\,k : I$ and is ϵ everywhere else. We note that this operation is well-defined because the monad structure associated with $-^?$ is commutative. This last fact follows in turn from propositional univalence. The verification that this map is commutative and associative is tedious but unsurprising. Each property hinges on careful case analysis as well as arithmetic and the relevant property holding for each $R(i)$.

The core operation is defined in a similar manner:

$$\begin{aligned}
&|(f, l : [m] \to I)| = \\
&\quad x_0 \leftarrow |f(l\,0)|; \\
&\quad \ldots \\
&\quad x_{m-1} \leftarrow |f(l\,m)|; \\
&\quad \eta\,(\{l\,k \mapsto x_k\}_{0 \le k < m}, l)
\end{aligned}$$

Its properties again stem from the relevant properties of $R(i)_+$ and case analysis. Note that $|{-}|$ is always defined. □

Remark 10.3.25. Without propositional univalence, we could only give the above construction after fixing some total order on I. This is sufficient for all uses; generally, I will be either finite or Nat—but avoiding such hypotheses is a pleasant side-effect of working such a rich internal language. ◇

Remark 10.3.26. A major deficiency of $\bigoplus$ is that it does not enjoy an easily stated universal property. Indeed, it need not even be functorial in R! This is closely related to the fact that X_+ is almost—but not quite!—the unital completion of X.

This suggests that our present definition of the category of RAs is not entirely satisfactory. The existing definition given in Iris, however, suffers even more deficiencies: the resulting category lacks sums and products, among other useful constructions. The search for the correct definition of **RA** is left to future work, though we do note that the crux of the issue centers around the core operation. It seems likely that a better category could be found if one imposes additional requirements on $|{-}|$ giving it more structures in order to ensure that e.g., $\bigoplus R$ is a certain unital completion. ◇

We conclude Section 10.3 with the *authoritative* RA Auth. This RA is a mix of the exclusive and agreement RA which will be used to encode a number of features in the program logic. Fix a unital RA X and define the carrier of $\mathsf{Auth}\,X$ as follows:

$$\mathsf{Auth}\,X = X + X + \left(\textstyle\sum_{x,y:X} y \sqsubseteq x\right)$$

Borrowing notation from Iris, we will write $\bullet x$ and $\circ y$ for $\mathsf{in}_1\,x$ and $\mathsf{in}_2\,y$ respectively. We will not have much occasion to explicitly write $\mathsf{in}_3(x, y)$ as it will be used to represent multiplications of $\bullet\,x$ and $\circ\,y$. Intuitively, $\bullet x$ is *leader* while $\circ y$ is the *follower*. That is if we own $\bullet x$ then (1) no one else can $\bullet x'$ for any x' and every valid $\circ y$ must satisfy $y \sqsubseteq x$. We encode this through RA structure on $\mathsf{Auth}\,X$ as follows (omitting cases forced by the various laws of a RA):

$$\begin{aligned}
&\bullet x \cdot \_ = (\bot, !) \\
&\circ y \cdot \circ y' = z \leftarrow y \cdot y'; \eta \circ z \\
&\circ y \cdot \bullet x = (y \sqsubseteq x, \mathsf{in}_3(x, y, -))
\end{aligned}$$

$$\begin{aligned}
&|\bullet x| = \bot \\
&|\circ y| = z \leftarrow |y|; \eta \circ z
\end{aligned}$$

Lemma 10.3.27. $\mathsf{Auth}\,X$ *with the above definitions of multiplication and core forms a RA.*

10.4 The type of propositions

With the machinery of RAs available, we are now in a position to define Ω^*. We will proceed in two steps. First, we will define a type Ω^*_R for an arbitrary RA R and show that this type can be equipped with the structure of a model of higher-order logic along with several connectives reflecting the structure of R (such as $*$ and $-\!\!*$). After this machinery is in place, we will use loeb-induction to construct a suitably recursive version of Ω^*_R which allows R to depend on Ω^*_R itself. This type will be the basic type of propositions within the program logic of synthetic Iris.

10.4.1 Monotone predicates on a RA

For the remainder of this subsection, fix a RA R. Our aim is to define Ω^*_R and close it under various connectives. Intuitively, Ω^*_R is the type of *monotone predicates on R with respect to the extension order* or, symbolically, $\Omega^*_R = R \to_{\mathsf{mon}} \Omega$.

Remark 10.4.1. For those more categorically-minded, this is the type of Ω-valued presheaves on R^{op}. This immediately—together with the symmetric monoidal structure on R^{op}—yields several of the results of this subsection, but we will spell them out for clarity. For those more familiar with Iris, the constructions will appear to be strict simplifications of the definitions given by e.g., Jung et al. [Jun+18]. Indeed, by working systematically within mode t we are able to avoid any mention of steps or step-indexing.[4] ◇

We begin by observing that there is a natural ordering on Ω^*_R inherited point-wise from Ω:

$$\begin{aligned}&(\vdash) : \Omega^*_R \times \Omega^*_R \to \Omega\\&\Phi \vdash \Psi \triangleq \forall r : R.\, \Phi\, r \to \Psi\, r\end{aligned}$$

We will now show that this relation $\vdash$ behaves more-or-less like an entailment relation from higher-order logic. In particular, we will now define operations like $(\wedge) : \Omega^*_R \times \Omega^*_R \to \Omega^*_R$ satisfying the expected properties:

$$(\Psi \vdash \Phi_0 \wedge \Phi_1) \iff (\Psi \vdash \Phi_0) \wedge (\Psi \vdash \Phi_1)$$

Note by propositional univalence that if $\Phi \vdash \Psi$ and $\Psi \vdash \Phi$ then $\Phi = \Psi$.

Fortunately, there is a way to streamline this process. Almost all of the relevant connectives can be defined provided we can show that $(\Omega^*_R, \vdash)$ is a complete lattice. That is, it has arbitrary upper- and lower-bounds. For instance, the above specification of conjunction shows that it boils down to the existence of binary meets. Similar thinking shows that true, false, disjunction, universal quantification, and existential quantification

[4] These two facts do give me some concern that the results of this subsection will be obvious to all prospective readers. However, they will be obvious for such a disjoint set of reasons that I could not omit them entirely without some guilt.

can be modeled by a top element, a bottom element, binary joins, infinitary meets, and infinitary joins respectively.

Lemma 10.4.2. $(\Omega_R^*, \vdash)$ *is an complete lattice.*

Proof. Let us begin by defining infinitary joins as follows:

$$\bigvee_i \Phi_i = \lambda r.\, \exists i.\, \Phi_i\, r$$

Inspection shows that this is monotone: if $(\bigvee_i \Phi_i)\, r_0$ holds and $r_0 \sqsubseteq r_1$, we must have $\Phi_i\, r_0$ and therefore $\Phi_i\, r_1$ for some i. We will now argue that it is the least upper-bound. To this end, suppose that we are given Ψ such that $\Phi_i \vdash \Psi$ for all i, we must show that $\bigvee_i \Phi_i \vdash \Psi$. To this end, fix $r : R$ so that it suffices now to show that $\exists_i \Phi_i\, r \to \Psi\, r$ holds. By assumption, $\Phi_i\, r \to \Psi\, r$ for all i and so the conclusion follows.

The definition and argument for arbitrary meets are symmetric:

$$\bigwedge_i \Phi_i = \lambda r.\, \forall i.\, \Phi_i\, r$$

Again, monotonicity follows from the monotonicity of each Φ_i. □

Not all of the structure from higher-order logic follows from this result. In particular, implication is not definable as either a join or a meet. In order to close Ω_R^* under implications, two potential approaches present themselves. First, we could show that Ω_R^* satisfies a certain distributivity law: $\Psi \wedge \bigvee_i \Phi_i = \bigvee_i \Psi \wedge \Phi_i$. With this property to hand, one can *define* implication as follows:

$$\Phi \to \Psi = \bigvee \{\Xi \mid \Xi \wedge \Phi \vdash \Psi\}$$

A standard argument shows that this definition satisfies the intended property. One can also explicitly define implication through a formula reminiscent of a standard Kripke semantics for logic. A standard argument shows that if implication exists, then the aforementioned distributive property holds, so these two approaches are completely equivalent. We will opt for the latter.

$$\begin{array}{l}(\to) : \Omega_R^* \times \Omega_R^* \to \Omega_R^* \\ \Phi \to \Psi = \lambda r.\, \forall r' \sqsupseteq r.\, \Phi\, r' \to \Psi\, r'\end{array}$$

Notice that by construction this definition is monotone.

Lemma 10.4.3. *This definition of implication satisfies the expected property:* $\Xi \vdash \Phi \to \Psi$ *if and only if* $\Xi \wedge \Phi \vdash \Psi$.

Proof. This follows more-or-less by unfolding. $\forall r.\, \Xi\, r \to (\forall r'.\, \Phi\, r \to \Psi\, r)$ is equivalent to $\forall r.\, \Xi\, r \wedge \Phi\, r \to \Psi\, r$. It is clear that the first implies the second by choosing $r' = r$ and currying. The latter implies the former, however, by instantiating at r' and using the monotonicity of Ξ. □

Finally, we note that any standard proposition $\phi : \Omega$ gives rise to an element of Ω_R^* and this procedure preserves and reflects provability.

Lemma 10.4.4. *There is a function* $\ulcorner - \urcorner : \Omega \to \Omega^*_R$ *such that if* R *is non-empty* $\ulcorner \phi \urcorner \vdash \ulcorner \psi \urcorner$ *if and only if* $\phi \to \psi$.

Proof. There are two ways to construct $\ulcorner \phi \urcorner$. We can define it explicitly by setting $\ulcorner \phi \urcorner\, r = \phi$ and, calculating using the assumed point $r : R$, the desired bi-implication follows immediately. It is worth noting that we can actually define $\ulcorner \phi \urcorner$ with the already available machinery as $\exists z : \phi.\, \top$. □

This completes the construction of the normal connectives of intuitionistic logic so three classes of connectives remain: guarded connectives ($\blacktriangleright$), resource connectives (Own, $*$, and $-\!\!*$), novel Iris modalities[5] ($\Box$, $\Rrightarrow$). We will handle these in order.

Lemma 10.4.5. Ω^*_R *is close under a* $\triangleright$ *modality satisfying* $\Phi \vdash \triangleright \Phi$, $\triangleright \Phi \wedge \triangleright \Psi = \triangleright(\Phi \wedge \Psi)$ *and a version of Löb induction:*

$$(\Phi \wedge \triangleright \Psi \vdash \Psi) \to (\Phi \vdash \Psi)$$

Furthermore,

Proof. Recall by construction that there is a map $\blacktriangleright : \Omega \to \Omega$ using both the restriction of $\langle l \mid - \rangle$ to Ω along with the inequality $\mathsf{id} \leq l$. We will define $\triangleright$ by post-composition:

$$\triangleright \Phi = \blacktriangleright \circ \Phi$$

Monotonicity comes from the fact that all MTT modalities satisfy axiom K along with the inequality $\mathsf{id} \leq l$ again. Indeed, these same two facts immediately yield $\Phi \vdash \triangleright \Phi$ and $\triangleright \Phi \wedge \triangleright \Psi = \triangleright(\Phi \wedge \Psi)$. Accordingly, we choose to focus on the Löb induction scheme.

Let us suppose that $\Phi \wedge \triangleright \Psi \vdash \Psi$ and attempt to prove $\Phi \vdash \Psi$. Unfolding definitions, let us fix $r : R$ and suppose $\Phi\, r$ holds. We must show $\Psi\, r$ holds and here we use Löb induction and thereby suppose $\blacktriangleright(\Psi\, r)$ holds. We now instantiate $\Phi \wedge \triangleright \Psi \vdash \Psi$ at r to obtain a proof of $\Phi\, r \wedge \blacktriangleright(\Psi\, r) \to \Psi\, r$. The conclusion now follows from our hypotheses of $\Phi\, r$ and $\blacktriangleright(\Psi\, r)$. □

Remark 10.4.6. There is a *dependent* version of $\hat{\triangleright}$ with the type $\blacktriangleright \Omega^*_R \to \Omega^*_R$. One can then define the earlier version of $\triangleright = \hat{\triangleright} \circ \mathsf{next}$. ◇

Remark 10.4.7. The above result is closely related to the construction of Palombi and Sterling [PS23] showing that presheaves valued in a model of guarded recursion are themselves a model of guarded recursion. ◇

Thus far all of the structure of Ω^*_R has been inherited more-or-less directly from Ω. We now finally use some of the specifics of R to formulate several connectives which deal with ownership. Intuitively, we will construct an alternative version of conjunction (another symmetric monoidal product) that captures the idea of *separating conjunction* $\Phi * \Psi$. Roughly, at $r : R$ the separating conjunction $\Phi * \Psi$ should hold if (1) there exists some decomposition $\eta\, r = r_1 r_2$ such that $\Phi\, r_1$ and $\Psi\, r_2$. We have discussed the motivations for such a connective already in Remark 10.0.1.

We can turn the above intuition into a concrete definition as follows:

[5] This is a somewhat unfortunate terminological clash: these modalities are distinct from anything we have worked with thus far and are completely disjoint from the mode theory of synthetic Iris.

$$(*) : \Omega^*_R \times \Omega^*_R \to \Omega^*_R$$
$$(\Phi * \Psi) = \lambda r.\, \exists r_1, r_2.\, (r_1 \cdot r_2 = \eta\, r) \wedge \Phi\, r_1 \wedge \Psi\, r_2$$

We must show that $*$ satisfies a number of properties, but we will begin with the basic structural properties of $*$:

$$\Phi * \Psi = \Psi * \Phi \qquad \Phi * (\Psi * \Xi) = (\Phi * \Psi) * \Xi \qquad \Phi * \top = \Phi$$
$$(\Phi_0 \vdash \Phi_1) \wedge (\Psi_0 \vdash \Psi_1) \to (\Phi_0 * \Psi_0 \vdash \Phi_1 * \Psi_1)$$

Remark 10.4.8. These properties all correspond to recognizable categorical structures: they essentially organize $*$ into a symmetric monoidal product with $\top$ as the unit. Indeed, if one scrutinizes the above definition of $*$ it can be immediately recognized as an instance of *Day convolution* [Day70]—a highly general instance of it at least relying on an enriched pro-monoidal structure. The benefit of recasting $*$ as an instance of Day convolution is that all of the above properties then follow automatically from the corpus of known results about Day convolution. ◇

Lemma 10.4.9. *The above structural properties of $*$ hold.*

Proof. Each of the properties listed above holds by unfolding definitions and computing (some are nearly immediate e.g., commutativity follows immediately from the commutativity of $\wedge$). We will focus on associativity as a representative case.

Fix Φ, Ψ, Ξ together with $r : R$. Suppose that we are given $(\Phi * (\Psi * \Xi))\, r$. Unfolding the definition of $*$, this amounts to a decomposition of r as $\eta\, r = (r_{12} \leftarrow r_1 r_2; r_0 r_{12})$ for some $r_0, r_1, r_2 : R$ together with $\Phi\, r_0$, $\Psi\, r_1$ and $\Xi\, r_2$. However, by associativity of we know that $\eta\, r = (r_{01} \leftarrow r_0 r_1; r_{01} r_2)$ and this decomposition of r suffices to show $((\Phi * \Psi) * \Xi)\, r$. □

Lemma 10.4.10. *For any $\Phi, \Psi : \Omega$ we have $\triangleright\, \Phi * \triangleright\, \Psi \vdash \triangleright(\Phi * \Psi)$.*

Remark 10.4.11. As we have shown that $\top$ is the unit for $*$ and that $*$ is functorial in both arguments, we have the following:

$$\Phi * \Psi \vdash \Phi * \top \vdash \Phi$$

Notably, however, we do not have $\Phi \vdash \Phi * \Phi$ in general. This would amount to a proof that $\eta\, r = r \cdot r$ for each $r : R$. This is fortunate: if $\Phi \vdash \Phi * \Phi$ held then $\Phi * \Psi$ would degenerate to $\Psi \wedge \Phi$. ◇

Just as $\wedge$ comes equipped with $\to$, there is a type of implication associated to $*$ denoted $-\!\!*$ i.e. $*$ is a *closed* monoidal product. Once again we could choose to construct $\Phi -\!\!* \Psi$ either directly or indirectly and we will again opt for giving a concrete formula:

$$(-\!\!*) : \Omega^*_R \times \Omega^*_R \to \Omega^*_R$$
$$\Phi -\!\!* \Psi = \lambda r_0.\, \forall r_1 r.\, (\eta\, r = r_0 \cdot r_1 \wedge \Phi\, r_1) \to \Psi\, r$$

Calculation with the above definition yields the expected result:

Lemma 10.4.12. $\Phi * \Psi \vdash \Xi$ *if and only if* $\Phi \vdash \Psi -\!\!* \Xi$.

Remark 10.4.13. Just as $*$ can be seen as an instance of Day convolution, $-\!\!*$ is a specialization of Day's formula for the right-adjoint to the tensor product [Day70]. $\diamond$

The utility of separating conjunction only becomes apparent when used together with another connective: $\mathsf{Own} : R \to \Omega^*_R$. In plain English, $\mathsf{Own}(r)$ is true at r' just when $r \sqsubseteq r'$. Intuitively then, $\mathsf{Own}(r)$ is a sort of "threshold" proposition: true when the current world contains r and false otherwise.

$$\begin{aligned}&\mathsf{Own} : R \to \Omega^*_R\\ &\mathsf{Own}\, r = \lambda r'.\ r \sqsubseteq r'\end{aligned}$$

Remark 10.4.14. Categorically, Own is given by the Yoneda embedding of R^{op} into Ω^*_R. The crucial properties of Own—its functoriality and interaction with $*$—are consequences of this observation. $\diamond$

This connective is what gives Ω^*_R its character as a separation logic i.e., if separation logic is a logic of resources, this is the primitive form resource. We shall eventually see it as the (highly generalized) form of the classical $\ell \mapsto v$ connective, but for now, we shall continue to work without assumption on R and explore its properties at this level.

Lemma 10.4.15. *If* $r_0 \sqsubseteq r_1$ *then* $\mathsf{Own}\, r_1 \vdash \mathsf{Own}\, r_0$

Proof. Unfolding definitions, this follows from the transitivity of $\sqsubseteq$. □

What makes $*$ and Own so powerful in conjunction is their ability to internalize the multiplication operator for R into the logic. More crucially, the partiality of the multiplication can be used to show that owning two incompatible elements of R yields a contradiction:

Lemma 10.4.16. *Fix* $r_0, r_1 : R$ *and write* $(\phi, r_\phi) = r_0 r_1$*. Under these assumptions,* $\mathsf{Own}\, r_0 * \mathsf{Own}\, r_1 = \exists z : \phi.\, \mathsf{Own}(r_\phi\, z)$

Proof. Fix $r : R$ and let us compute both sides of the purported equality at r:

$$\begin{aligned}&(\mathsf{Own}\, r_0 * \mathsf{Own}\, r_1)\, r = \exists r'_0, r'_1.\, r'_0 r'_1 = \eta\, r \wedge r_0 \sqsubseteq r'_0 \wedge r_1 \sqsubseteq r'_1\\ &(\exists z : \phi.\, \mathsf{Own}(r_\phi\, z))\, r = \exists z : \phi.\, r_\phi\, z \sqsubseteq r\end{aligned}$$

Let us first prove that the first implies the second. Assume that we are given r'_0 and r'_1 satisfying the above properties. In this case, there exist s_0 and s_1 such that $s_i r_i = \eta\, r'_i$ and so $(s_0 r_0)(s_1 r_1) = \eta\, r$. Applying associativity and commutativity, we therefore conclude that $r_0 r_1 = \eta\, r_{01}$ for some $r_{01} : R$ and that $s_0 s_1 r_{01} = \eta\, r$. Inspecting our goal, we conclude that $\phi = \top$ and $r_\phi = r_{01}$ and so the conclusion follows immediately.

For the reverse, suppose that $\phi = \top$ and that $r_\phi \sqsubseteq r$ so that $s \cdot r_\phi = \eta\, r$. By associativity, we can rewrite the left-hand side of this equation as $r_0(s \cdot r_1) = \eta\, r$ and, in particular, there exists r' such that $\eta\, r' = s \cdot r_1$. In this case, we choose $r'_0 = r_0$ and $r'_1 = r'$ and the conclusion follows immediately. □

Finally, we discuss two modalities $\square$ and $\Rrightarrow$. Both of these manipulate the "ambient element of R" being threaded through Ω^*_R, but in different ways. We begin with $\square$, which internalizes the coring operation on R. The definition of $\square\Phi$ intuitively precomposes Φ with $|-|$, but of course this is not directly available ($|-|$ maps to $R^?$ not R). The slightly refined definition is given below:

$$\begin{array}{l}\Box : \Omega^*_R \to \Omega^*_R \\ \Box\Phi = \lambda r.\, \textbf{let}\ (\phi, r_\phi) = |r|\ \textbf{in}\ \exists z : \phi.\, \Phi\,(r_\phi\, z)\end{array}$$

This definition is monotone precisely because we have required $|{-}|$ to be monotone with respect to the extension order.[6]

Remark 10.4.17. The classical definition of Iris given in e.g., Jung et al. [Jun+18] presupposes that R is unital so this definition is a strict generalization to the context of a general RA. However, certain properties of $\Box$ do not hold generally. Most notably, $\Box\top = \top$ is not true in general; it is equivalent to the requirement that $|{-}|$ is total. ⋄

Lemma 10.4.18. *$\Box$ is an S4 comonad: $\Box\Phi \vdash \Phi$, $\Box\Box\Phi = \Box\Phi$, $\Box\Phi \wedge \Box\Psi = \Box(\Phi \wedge \Psi)$, and $(\Phi \vdash \Psi) \to (\Box\Phi \vdash \Box\Psi)$.*

The essence of $\Box$ is captured by the following two lemmas; they both show how $\Box$ collapses the substructural connectives of Ω^*_R into structural counterparts.

Lemma 10.4.19. *Given $\Phi, \Psi : \Omega^*_R$, $(\Box\Phi) \wedge \Psi = (\Box\Phi) * \Psi$.*

Proof. Unfolding both propositions in question, this statement is equivalent to the following for each $r : R$: $s = |r|$ is defined with $\Phi\, s$ and $\Psi\, r$ if and only if there exists a decomposition $r_0 r_1 = \eta\, r$ such that $s_0 = |r_0|$ is defined, $\Phi\, s_0$, and $\Psi\, r_1$. The only if direction is straightforward: we choose $r_0 = |r|$ and $r_1 = r$ and both properties follow by assumption. For the reverse direction, we note that since $r_0 \sqsubseteq r$ and r_0 has a defined core s_0 by the properties of a RA r must also have a core s and $s_0 \sqsubseteq s$. Consequently, the conclusion follows by assumption and the monotonicity of Φ and Ψ. □

Lemma 10.4.20. *Given $r : R$, if $(\phi, r_\phi) = |r|$ then $\mathsf{Own}\, r \vdash \forall z : \phi.\, \Box\, \mathsf{Own}(r_\phi\, z)$*

Proof. Unfolding, this entailment is equivalent to the following: for all $r_0 : R$, if $r \sqsubseteq r_0$ and $\phi = \top$ then $|r_0|$ is defined and $r_\phi \sqsubseteq |r_0|$. This is a rephrasing of the monotonicity of $|{-}|$ with respect to the extension order. □

Recall that $\triangleright$ was defined by post-composition with $\blacktriangleright$. As $\Box$ is defined essentially by "precomposition" one might hope that these two modalities commute. This is not the case in general but does happen in our major case of interest: when R is unital.

Lemma 10.4.21. *If R is unital then $\Box \triangleright \Phi = \triangleright \Box\Phi$.*

Proof. Note that if R is unital then $|{-}|$ is a total function and therefore can be factored into $\eta \circ \mathsf{core}$. In this situation, $\Box\Phi = \Phi \circ \mathsf{core}$ and the desired conclusion follows by calculation. □

Similar considerations yield the following:

Lemma 10.4.22. *If R is unital then $\Box\forall a : A.\, \Phi\, a = \forall a : A.\, \Box(\Phi\, a)$ and $\Box\exists a : A.\, \Phi\, a = \exists a : A.\, \Box(\Phi\, a)$*

[6] Indeed, this is precisely the reason why such a requirement is necessary. A fact worth noting given the difficulties that this requirement caused in Section 10.3.

The final connective of our logic is the *frame-preserving update monad* $|\!\!\Rrightarrow$. Intuitively, $|\!\!\Rrightarrow \Phi$ holds at r when there is some other element r' for which (1) $\Phi\, r$ holds and (2) if $s \cdot r$ is defined, $s \cdot r'$ is also defined. The last point is the frame-preservation condition: we are allowed to consider a different element of r but only if the new choice is compatible with all possible *frames* of the original r. More formally:

$$\begin{aligned}&|\!\!\Rrightarrow : \Omega^*_R \to \Omega^*_R\\ &|\!\!\Rrightarrow \Phi = \lambda r.\, \forall s.\, (s \cdot r)\downarrow \to \exists r'.\, (s \cdot r')\downarrow \wedge \Phi\, r'\end{aligned}$$

This definition is indeed monotone: if $s \cdot r_1$ is defined and $r_0 \sqsubseteq r_1$, then $s \cdot r_0$ is also defined. By choosing $r' = r$ in the body of the existential, we also see that $\Phi \vdash |\!\!\Rrightarrow \Phi$. In fact, more is true:

Lemma 10.4.23. *$|\!\!\Rrightarrow$ is a strong monad with respect to separating conjunction.*

Proof. The proof that $|\!\!\Rrightarrow |\!\!\Rrightarrow \Phi \vdash |\!\!\Rrightarrow \Phi$ follows from unfolding definitions, so we choose to focus on the strength: $\Phi * |\!\!\Rrightarrow \Psi \vdash |\!\!\Rrightarrow(\Phi * \Psi)$. Fix r and suppose we are given $r_0 \cdot r_1 = \eta\, r$ such that $\Phi\, r_0$ and $(|\!\!\Rrightarrow \Psi) r_1$. We wish to show $(|\!\!\Rrightarrow(\Phi * \Psi))r$.

Accordingly, fix s such that $s \cdot r\downarrow$. We note then that $s' = s \cdot r_0$ must also be defined and that $s' \cdot r_1$ is defined (and equal to $s \cdot r$). From $(|\!\!\Rrightarrow \Psi)\, r_1$, we then conclude that there exists some r_1' such that both $s' \cdot r_1'\downarrow$ and $\Psi\, r_1'$ hold.

Returning to our goal, we must show that there exists some r' such that $\Phi * \Psi$ holds at r' and $s \cdot r'\downarrow$. We choose r' such that $\eta\, r' = r_0 \cdot r_1'$—noting that the right-hand side of this equality is indeed defined as $r_0 \sqsubseteq s'$. Our assumption of $\Phi\, r_0$ and $\Psi\, r_1'$ then yield the conclusion. □

It remains to characterize how Own behaves with respect to $|\!\!\Rrightarrow$. Intuitively, $\mathsf{Own}\, r$ should imply $|\!\!\Rrightarrow(\mathsf{Own}\, r')$ just when $r \cdot s\downarrow$ implies $r' \cdot s\downarrow$ for all s. In order to codify this situation properly, we introduce the following notion:

Definition 10.4.24. Fix an element $r : R$ along with a subset $S : R \to \Omega$, we say that $r : R$ updates to S (written $r \rightsquigarrow S$) when the following holds:

$$r \rightsquigarrow S = \forall s.\, (s \cdot r)\downarrow \to \exists r' \in S.\, (s \cdot r')\downarrow$$

We write $r \rightsquigarrow s$ as shorthand for $r \rightsquigarrow (s = -)$.

Remark 10.4.25. We note here that we have taken advantage of a standard trick in type theory to encode a subset via an *indicator function*. Specifically, to describe a subset of R, we require a function $\phi : R \to \Omega$ and view ϕ as encoding the subset of elements of R where ϕ holds. For instance, $\emptyset = \lambda\_.\bot$ and the maximum subset is encoded by $\lambda\_.\top$. ◇

Remark 10.4.26. This definition of frame-preserving update differs slightly from the one given in Iris. We require the frame to be drawn from R rather than R_+ [Jun+18]. In particular, in certain RAs with our definition, one has $a \rightsquigarrow \emptyset$ while this can never occur with the standard Iris definition. We have opted for our slightly simpler definition and simply explicitly note when we must require that B is non-empty. ◇

Lemma 10.4.27. *Given $r : R$ and $S : R \to \Omega$ such that $r \rightsquigarrow S$ the following holds:*

$$\mathsf{Own}\, r \vdash |\!\!\Rrightarrow(\exists s \in S.\, \mathsf{Own}\, s)$$

In particular, if $r \rightsquigarrow s$ then $\mathsf{Own}\, r \vdash |\!\!\Rrightarrow \mathsf{Own}\, s$

10.4.2 A general RA

We now set about choosing a suitable RA R so that we can define $\Omega^* = \Omega^*_R$. This is one of the more subtle points in the formalization of Iris in Coq: except at the leaves of development (the closed proofs that some program satisfies some particular specification) it is impossible to actually fix R. In practice, Iris developments work relative to *some* R and place constraints on what sort of operations R supports. When one wishes to obtain a closed proof of some statement, a particular R is chosen to satisfy all the necessary constraints. This discipline is essential for modular development: a formalization of a queue data structure cannot realistically be expected to know all of the requirements on R that will be necessary to verify any code which uses the queue itself.

In fact, this process is less complex than it might appear. We will choose $R = R_F = \bigoplus_{I \times \mathsf{Nat}} F \circ \pi_1$ where $F : I \to \mathbf{RA}$ is a collection of RAs. Rather than attempting to specify F all at once, Iris developments in Coq contain constraints like "there is some i such that $F(i) = S$" for some specific RA S. At the end of the day, these constraints can be collected into an actual function F and R_F is used to instantiate Ω^*. One small complication remains: certain RAs, especially those used to model *invariants*, will actually depend upon Ω^* itself. This seeming circularity can be solved using a small amount of guarded recursion.

Remark 10.4.28. At this point, the reader unfamiliar with Iris may be confused why have tangled up the natural numbers into the definition of R instead of taking the more natural $R = \bigoplus_I F$. The addition of Nat will later be used to ensure that any i and any $a : F\,i$ we are always able to obtain a piece of ghost state $\mathsf{Own}(\mathsf{in}_{i,n}\,a)$ for some n. ⋄

In total then, a user (gradually) specifies a function $F : \mathcal{U} \to I \to \mathbf{RA}$ and Ω^* is taken to be the following:

$$\Omega^* = \mathsf{loeb}(X.\,\mathbf{let}\ R = \bigoplus_{I \times \mathsf{Nat}} (F(\blacktriangleright X) \circ \pi_1)\ \mathbf{in}\ \Omega^*_R)$$

Theorem 10.4.29. *Ω^* is a sound model of guarded higher-order logic with Own, $*$, $-\!\!*$, $\Box$, and $\Rrightarrow$ together with their expected rules. There is an embedding $\ulcorner - \urcorner : \Omega \to \Omega^*$ which preserves and reflects provability.*

Proof. By the unfolding equation for Löb induction, we know that $\Omega^* = \Omega^*_R$ for some RA R and so this follows from the results of the prior subsection. □

Notation 10.4.30. Hereafter we write R for $\bigoplus_{(i,\_):I \times \mathsf{Nat}}(F(\blacktriangleright\Omega^*, i))$ so that, in particular, $\Omega^* = \Omega^*_R$.

Remark 10.4.31. We emphasize that this definition only uses standard Löb induction. Unlike the Coq formalization of Iris, there is no need to construct a special *domain equation* solver; Löb induction on the universe plays the same role. This enables us to bypass one of the most technical portions of the Iris formalization. ⋄

For this subsection, we will fix $F : \mathcal{U} \to I \to \mathbf{RA}$ for some type I with decidable equality and discuss the special features of Ω^*. Any direct sum is a unital RA; in particular, there is an element $\epsilon : R$.

Lemma 10.4.32. *In Ω^*, $\mathsf{Own}\,\epsilon = \top$*

Proof. This is easiest to see by direct calculation. Fix $r : R$, we must show that $\epsilon \sqsubseteq r$ but this follows immediately from the equation $\eta\, r = r \cdot \epsilon$. □

Remark 10.4.33. For those more categorically minded, this last lemma follows from the observation that ϵ is initial in any uRA and therefore terminal in R^{op} in particular. The above lemma then is a specialization of the well-known fact that the Yoneda embedding preserves terminal objects. ◇

There is a good reason to use the direct sum construction over the sum or product of RAs: only the direct sum enables us to embed the Own connective for each $F(i)$ in such a way that the rules governing the interactions of $*$ and $\mid\!\Rrightarrow$ remain unchanged.

Lemma 10.4.34. *Given $a, b : F\,i$ such that $(\phi, c) = a \cdot b$ the following holds:*

$$\mathsf{Own}(\mathsf{in}_{i,n}\, a) * \mathsf{Own}(\mathsf{in}_{i,n}\, b) \vdash \exists z : \phi.\, \mathsf{Own}(\mathsf{in}_{i,n}(c\, z))$$

Lemma 10.4.35. *Fix $a : F\,i$ and non-empty subset $B : F\,i \to \Omega$ such that $a \rightsquigarrow B$ in $F\,i$. Writing $\mathsf{in}_{i,n}$ for the inclusion $F\,i \to R$, in this situation $\mathsf{in}_{i,n}\, a \rightsquigarrow \{\mathsf{in}_{i,n}\, b \mid b \in B\}$.*

Proof. We begin by unfolding the definition of $\rightsquigarrow$. Fixing $r : R$ such that $\mathsf{in}_{i,n}\, a \cdot r$ is defined, we must show that there exists $b \in B$ such that $\mathsf{in}_{i,n}\, b \cdot r$ is also defined.

We observe that it suffices to consider two distinct cases: either (i, n) is in the support of r or it is not. This proposition is seen to be decidable by inspecting the construction of $\bigoplus$; r contains a function with finite domain whose image contains (i, n) just when this holds. Let us suppose first that (i, n) is *not* in the support of r. In this case, any element of B suffices (and, by assumption, B is non-empty).

Suppose instead that (i, n) is in the support of r. In this case, we may write $r = r_0 \cdot \mathsf{in}_{i,n}\, c$ for some $c : F\,i$ and $r_0 : R$ such that (i, n) is not in the support of r_0. Inspecting the definition of multiplication on $\bigoplus$ we conclude that $a \cdot c$ is defined because $\mathsf{in}_{i,n}\, a \cdot \mathsf{in}_{i,n}\, c$ is defined. Therefore, there exists some $b \in B$ such that $b \cdot c$ is defined whereby $\mathsf{in}_{i,n}\, b \cdot \mathsf{in}_{i,n}\, c$ is also defined. It suffices to show that $\mathsf{in}_{i,n}\, b \cdot \mathsf{in}_{i,n}\, c \cdot r_0$ is also defined, but this follows from our assumption that (i, n) is not in the support of r_0. □

Notation 10.4.36. We will write $\boxed{a : F\,i}^{n}$ as shorthand for $\mathsf{Own}(\mathsf{in}_{i,n}\, a)$. Often $F\,i$ will be apparent from context and we will simply write $\boxed{a}^{n}$.

Corollary 10.4.37. *If $a \rightsquigarrow B$ and B is non-empty, then $\boxed{a}^{n} \vdash \mid\!\Rrightarrow \exists b \in B.\, \boxed{b}^{n}$.*

The usual pattern is something like the following: we wish to start using a particular RA to specify some program and we therefore wish to obtain $\boxed{a_0}^{n}$ for where a_0 represents the "initial configuration" that the ghost state is meant to model. Ideally, we would use the above rule to obtain $\mid\!\Rrightarrow \boxed{a_0}^{n}$ but this is not valid in general: such an update is only available if we knew every possible $r : R$ would be compatible with $\mathsf{in}_{i,n}\, a$ and this is unlikely to be the case.

However, because there is an infinite number of "copies" of $F\,i$ (one for each natural number) and each frame $r : R$ has finite support, there always exists some number n such that (n, i) is not in the support of r. Accordingly, for each frame r there exists some n such that $r \cdot \mathsf{in}_{i,n}\, a$ is defined. Written more formally, for any $a : F\,i$ the following holds:

$$\epsilon \rightsquigarrow \{\mathsf{in}_{i,n}\, a \mid n : \mathsf{Nat}\}$$

Combining this with the frame-preserving update rule and the equality $\top = \mathsf{Own}\,\epsilon$, we obtain the following:

Lemma 10.4.38. $\top \vdash \mathrel{|\!\!\Rrightarrow} \exists n. \boxed{a}^{n}$.

Having worked at this level of generality, we now acknowledge that we can actually be concrete about which $F : \mathcal{U} \to I \to \mathbf{RA}$ is required for our development. Unlike Iris where one wants to be fully agnostic in the available RAs to enable a user to customize the logic, this synthetic account is entirely closed. We will require four distinct RAs, so we will take $I = \mathbf{4}$ and define F as follows:

$$\begin{aligned}
F\,X\,0 &= \mathsf{Auth}(\mathsf{Nat} \rightharpoonup_{\mathsf{fin}} \mathsf{Ag}\,X) \\
F\,X\,1 &= \mathcal{P}_{\mathsf{(co)fin}}(\mathsf{Nat}) \\
F\,X\,2 &= \mathcal{P}_{\mathsf{fin}}(\mathsf{Nat}) \\
F\,X\,3 &= \mathsf{Auth}\,\langle d \mid \mathsf{Heap}\rangle
\end{aligned}$$

Here $\mathcal{P}_{\mathsf{fin}}(\mathsf{Nat})$ ($\mathcal{P}_{\mathsf{(co)fin}}(\mathsf{Nat})$) are the decidable finite (finite or cofinite) subsets of the natural number endowed with the structure of a RA through disjoint union with a trivial core.

10.4.3 Commuting $\triangleright$ with other connectives

Experts in Iris will have noticed that we have bypassed several widely used rules in Iris governing the interaction between $\blacktriangleright$ and $\exists$ or $*$:

$$\triangleright(\Phi * \Psi) \vdash \triangleright\,\Phi * \triangleright\,\Psi \qquad\qquad \triangleright\,\exists a : A.\,\Phi\,a \vdash \triangleright\,\bot \vee \exists a : A.\,\triangleright\,\Phi\,a$$

$$\forall a : A.\,\triangleright\,\Phi\,a \vdash \triangleright\,\forall a : A.\,\Phi\,a$$

Their absence is not coincidental: neither rule is valid for Ω^* in synthetic Iris. This issue stems from an issue well-known Iris practitioners: both of these rules rely on specific properties of (1) the intended model of Iris and its connection to step-indexing over ω as opposed to a more general ordinal and (2) additional requirements on CMRAs and types as COFEs to ensure that the map $A \to \blacktriangleright A$ is always surjective. Neither of these two properties are available in synthetic Iris and there are worthwhile models which falsify them [Spi+21].

Remark 10.4.39. Forgoing the rule $\triangleright(\Phi * \Psi) \vdash \triangleright\,\Phi * \triangleright\,\Psi$ allows us to omit a certain condition for RAs:

$$\blacktriangleright(\exists a, b.\,\eta\,c = a \cdot b) \to \exists a, b.\,\blacktriangleright(\eta\,c = a \cdot b)$$

Adding this rule alone is insufficient to recover the aforementioned principle on Ω^* but when combined with certain specific facts about $\mathbf{PSh}(\omega)$ it suffices. ◇

Accordingly, we have two choices: we can either add principles to Ω manually to ensure that the corresponding rules for Ω^* hold or we can proceed with these principles. We opt for the latter. This affords more generality—transfinite instances Iris remains a model of synthetic Iris. All the results, including adequacy of the program logic, go through without these additional principles. What remains unclear, however, is whether the verification of particular programs would require them. For instance, the rule governing the interaction of $\triangleright$ and $\exists$ is frequently used when working with invariants. It is beyond the scope of this chapter to argue whether or not these rules can be omitted in practice, though such a question is of growing importance as Iris practitioners attempt to tackle liveness properties that seem to require transfinite Iris or other variants that fail to validate these rules.

10.5 The program logic

At this point, we have developed the theory of guarded higher-order separation logic through Ω^*, but it remains to actually put this logic to work to reason about programs. A remarkable fact about Iris is that this facet of the logic governing actual verification—the *program logic*—is entirely definable in terms of the primitives developed in Section 10.4. We show that this remains true in our synthetic account of Iris and develop a program logic for a small programming language with general references and concurrency based on `HeapLang` in Iris.

Convention 10.5.1. Within this section, we work internally to t.

The constructions of the program logic rely on various interlocking pieces of ghost state to coordinate invariants and manage state. In order to ensure that different components link together properly each definition is *parameterized* by a set of ghost names $\Sigma : \mathsf{GhostNames}$. In the Coq formalization of Iris, type-class magic is used to allow the user to simulate something akin to dynamic scoping. In this construction, however, we may simply define $\mathsf{GhostNames}$ as follows:

$$\begin{array}{l} \textbf{record}\ \mathsf{GhostNames} : \mathcal{U}\ \textbf{where} \\ \quad \gamma_{\mathsf{heap}} : \mathsf{Nat} \\ \quad \gamma_{\mathsf{inv}} : \mathsf{Nat} \\ \quad \gamma_{\mathsf{en}} : \mathsf{Nat} \\ \quad \gamma_{\mathsf{dis}} : \mathsf{Nat} \end{array}$$

Every construction of this section will be parameterized by such a Σ.

10.5.1 Encoding invariants

We now construct a crucial component of the program logic: *invariants.* This is particularly notable as it is a case where the dependence of the RA on Ω^* is necessary: invariants mention propositions which mention invariants, etc. While this is technically independent of the language defined above, we have chosen to present it in this place to localize the scope of certain assumptions we will make in this construction.

In order to build up invariants within Ω^*, we will require several distinct pieces of ghost state. To force these into existence, we must constrain the F used to construct Ω^* in Section 10.4.

Intuitively, the construction of invariants will factor into two parts: users will be provided with pieces of ghost state which—through the authoritative RA—will constrain the shape of some global registry assigning invariant names (natural numbers) to propositions. Behind the scenes, we then define a *world satisfaction predicate* which states that each proposition in the registry is true. A user can then work with their ghost state forcing some proposition to exist to “check it out” of the world satisfaction predicate and thereby temporarily gain access to the invariant. The entire interface is managed through a fancier version of $|\!\!\!\Rightarrow$ which is annotated with the list of invariants currently active.

This encoding is complex and contains multiple interacting components: the world satisfaction predicate, the fancy $|\!\!\!\Rightarrow$ modality, the propositions a user will hold to record an invariant, etc. All of these components must be connected through a collection of elements of Nat which annotates the underlying ghost state used to implement various constructions.

We start, somewhat arbitrarily, with the encoding of the proposition stating a user will own as a record that some invariant exists.

$$\begin{aligned}&\boxed{-}^{-} : \{\mathsf{GhostNames}\} \to \mathsf{Nat} \to \blacktriangleright\Omega^* \to \Omega^*\\ &\boxed{\Phi}^{\{\Sigma\};\iota} = \boxed{\circ\{\iota \mapsto \Phi\}}^{\Sigma.\gamma_{\mathsf{inv}}}\end{aligned}$$

Remark 10.5.2. In what follows, we will consistently surpress Σ when writing $\boxed{-}^{-}$ and simply write $\boxed{\Phi}^{\iota}$. This will be the general pattern for all definitions in this section. ◇

Lemma 10.5.3. $\boxed{\Phi}^{\iota}$ *is persistent;* $\boxed{\Phi}^{\iota} \vdash \Box\boxed{\Phi}^{\iota}$.

Proof. This follows immediately from the following observation:

$$|\circ\{\iota \mapsto \mathsf{next}\,\Phi\}| = \eta\,(\circ\{\iota \mapsto \mathsf{next}\,\Phi\})$$

□

The next crucial definition is the *world satisfaction predicate.* This is a large proposition which—though largely invisible to the user of Iris—is threaded through proofs to keep track of various invariants.

$$\begin{aligned}&\mathsf{WSat} : \{\mathsf{GhostNames}\} \to \Omega^*\\ &\mathsf{WSat}\,\{\Sigma\} = \exists I : \mathsf{Nat} \to_{\mathsf{fin}} \Omega^*.\,\boxed{\bullet I}^{\Sigma.\gamma_{\mathsf{inv}}} * \mathop{\Huge *}_{i \in \mathsf{dom}(I)} \boxed{\{i\}}^{\Sigma.\gamma_{\mathsf{en}}} \vee (\hat{\triangleright} I\,i * \boxed{\{i\}}^{\Sigma.\gamma_{\mathsf{dis}}})\end{aligned}$$

A few words of intuition are in order. WSat contains two separate components. First, it uses the authoritative RA to record the definitive mapping of invariant names to the actual propositions they are associated with. This does not record whether the invariants are currently in use etc., only what names and propositions are currently paired. Second, for each invariant that exists WSat contains either (1) a token stating that it is currently in use or (2) a token stating that it is currently *not* in use along with an actual proof of the relevant proposition. The process of opening and closing an invariant will boil down to flipping between these two states.

The final ingredient for encoding invariants is a version of $|\!\!\!\Rightarrow$ which passes through WSat and records which invariants are currently enabled. This *fancy update modality* is written $|\!\!\!\Rightarrow_{\mathcal{E}_1;\mathcal{E}_2}$ and defined as follows:

$$\begin{aligned}&|\!\!\!\Rightarrow_{-;-} : \{\mathsf{GhostNames}\} \to \mathcal{P}_{(\mathsf{co})\mathsf{fin}}(\mathsf{Nat}) \to \mathcal{P}_{(\mathsf{co})\mathsf{fin}}(\mathsf{Nat}) \to \Omega^*\\ &|\!\!\!\Rightarrow_{\{\Sigma\};\mathcal{E}_1;\mathcal{E}_2} \Phi = \boxed{\mathcal{E}_1}^{\Sigma.\gamma_{\mathsf{en}}} * \mathsf{WSat} \mathrel{-\!\!*} |\!\!\!\Rightarrow(\boxed{\mathcal{E}_2}^{\Sigma.\gamma_{\mathsf{en}}} * \mathsf{WSat} * \Phi)\end{aligned}$$

Remark 10.5.4. This definition is a slight simplification of the actual definition used in Iris [Jun+18]. The definition there includes the *timeless* operator $- \vee \triangleright \bot$ in front of $\mathrel{|\!\!\Rrightarrow}$. This allows for some conveniences around the eventual rules for opening invariants. However, we will not be discussing these rules in depth and the presence of the timeless operator does not substantively impact the points we do discuss. ◇

We emphasize that the *masks* $\mathcal{E}_i$ govern which invariants are potentially possible to access. Essentially, one should read $\mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_1;\mathcal{E}_2} \Phi$ as saying "assuming that any invariants in $\mathcal{E}_1$ which exist are active, Φ holds and any invariants in $\mathcal{E}_2$ which exist are then active". By varying the mask, certain invariants can be broken or reestablished through the fancy update modality. The verification of properties of the fancy update modality is largely routine and closely follows the proofs given in Jung et al. [Jun+18]. Accordingly, we simply state them here without proof and refer the reader to op. cit. for both further motivation and proof:

$$(\Phi \vdash \Psi) \to (\mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_1;\mathcal{E}_2} \Phi \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_1;\mathcal{E}_2} \Psi) \qquad \mathrel{|\!\!\Rrightarrow} \Phi \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\mathcal{E}} \Phi \qquad \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_0;\mathcal{E}_1} \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_1;\mathcal{E}_2} \Phi \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_0;\mathcal{E}_2} \Phi$$

$$\Phi * \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_1;\mathcal{E}_2} \Psi \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_1 \cup \mathcal{E};\mathcal{E}_2 \cup \mathcal{E}} \Phi * \Psi \quad (\mathcal{E} \cap \mathcal{E}_i = \emptyset)$$

These rules ensure that the fancy update modality behaves more-or-less like a graded version of $\mathrel{|\!\!\Rrightarrow}$. The following two crucial additional principles form the core interface for invariants:

$$\triangleright \Phi \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\mathcal{E}} \exists \iota.\, \boxed{\Phi}^{\iota} \qquad \boxed{\Phi}^{\iota} \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\mathcal{E}\setminus\iota} \triangleright \Phi * (\triangleright \Phi \mathrel{-\!\!*} \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}\setminus\iota;\mathcal{E}} \top) \quad (\iota \in \mathcal{E})$$

The first rule allows one to create and register an invariant. The second enables accessing an invariant which is presently active giving both (1) the actual content of the invariant $\triangleright \Phi$ and (2) a continuation $\triangleright \Phi \mathrel{-\!\!*} \mathrel{|\!\!\Rrightarrow}_{\mathcal{E}\setminus\iota;\mathcal{E}} \top$ enabling one to re-establish ι by providing $\triangleright \Phi$. Notice that in the latter rule in particular, the masks associated to the fancy update modality tracks which invariants are active.

10.5.2 Connecting physical state and ghost state

At this point, we have two very distinct concepts which we have used the word *state* to describe. Firstly, there is the logical ghost state present in Ω^* and manifested through Own and $\boxed{-}^{-}$. There is also the type of heaps Heap used to instrument the operational semantics for `HeapLang` and describe the behavior of references. A priori, nothing connects these two notions but relating the two is a crucial component of the eventual definition of the weakest precondition predicate.

As is done in standard Iris, we accomplish this via the *state-interpretation predicate*. In reality, this is merely some function $I : \langle d \mid \mathsf{Heap} \rangle \to \Omega^*$ which sends an element of the Heap type to some Ω^* meant to model it. By defining I with ghost state, we thereby link the two.

The state interpretation function for `HeapLang` is then quite straightforward to define:

$$\begin{aligned}&I : \{\mathsf{GhostNames}\} \to \langle d \mid \mathsf{Heap} \rangle \to \Omega^* \\ &I\,\{\Sigma\}\,\sigma = \boxed{\bullet\sigma}^{\Sigma.\gamma_{\mathsf{heap}}}\end{aligned}$$

In other words, the interpretation $I\,\sigma$ uses the authoritative RA to ensure that all views on the state are compatible with the actual heap, σ. The other half of the authoritative RA allows one to therefore constrain what σ must be. In particular, note that by axiom K there is a map $\mathsf{singleton} : \langle d \mid \mathsf{Nat}\rangle \to \langle d \mid \mathsf{Val}\rangle \to \langle d \mid \mathsf{Heap}\rangle$ constructing the (discrete) finite map associating one location to one value. Using this map, we define the following:

$$\begin{aligned}
&(\mapsto) : \{\mathsf{GhostNames}\} \to \langle d \mid \mathsf{Nat}\rangle \to \langle d \mid \mathsf{Val}\rangle \to \Omega^* \\
&\ell \mapsto^{\{\Sigma\}} v = \boxed{\circ(\mathsf{singleton}\,\ell\,v)}^{\Sigma.\gamma_{\mathsf{heap}}}
\end{aligned}$$

The crucial properties of these definitions are recorded in the following lemma:

Lemma 10.5.5.

- $\ell_0 \mapsto v_0 * \ell_1 \mapsto v_1 \vdash \ulcorner \ell_0 \neq \ell_1 \urcorner$
- $\ell \mapsto v_0 * I\,\sigma \vdash \ulcorner \sigma\,\ell = v \urcorner$
- $I\,\sigma \mathrel{-\!\!*} \mathrel{|\!\!\Rrightarrow} \exists l.\, \ell \mapsto v * I\,(\mathsf{extend}^{\dagger}\,\sigma\,\ell\,v)$.

Proof. We note that $\circ(\mathsf{singleton}\,\ell_0\,v_0) \cdot \circ(\mathsf{singleton}\,\ell_1\,v_1)$ is defined if and only if $\ell_0 \neq \ell_1$. The first point then follows from Lemma 10.4.16: if $\ell_0 = \ell_1$ then the antecedent implies $\bot$ and the conclusion follows immediately and if $\ell_0 \neq \ell_1$ the conclusion follows from the defining property of $\ulcorner - \urcorner$. The second point follows a similar argument, though in this case scrutinizing the behavior of multiplication between $\bullet\sigma$ and $\circ\mathsf{singleton}\,\ell\,v$.

The final point is slightly different. Using Lemma 10.4.27, it suffices to show the following where $\ell = \mathsf{alloc}^{\dagger}\,\sigma$:

$$\bullet\sigma \rightsquigarrow \bullet(\mathsf{extend}^{\dagger}\,\sigma\,\ell\,v) \cdot \circ(\mathsf{singleton}\,\ell\,v)$$

This follows from routine a routine calculation with the authoritative RA. □

10.5.3 Weakest preconditions

We are now (finally) in a position to define the central connective in Iris: weakest precondition. The weakest precondition proposition $\mathsf{wp}_{\mathcal{E}}\,e\,\{v.Q\,v\} : \Omega^*$ informally states that if the invariants named in $\mathcal{E}$ are active then e does not become stuck, and e happens to run to a value v then $Q\,v$ holds. This is more expressive than this may appear: Q is valued in Ω^* and so (by virtue of the state interpretation function) we may control the state of the heap after the execution of the program using $\ell \mapsto v$, $*$, and related connectives.

The definition of $\mathsf{wp}_{\mathcal{E}}\,e\,\{v.Q\,v\}$ is slightly complex, as it must tie together several disparate constructions: the operational semantics of `HeapLang`, guarded recursion, the invariants via $\mathrel{|\!\!\Rrightarrow}_{\mathcal{E}_0;\mathcal{E}_1}$, and the ghost state representing the heap via I. In the literature this definition is typically given in stages, we choose to opt for presenting the definition in its entirety first and highlight individual components afterward.

Definition 10.5.6. We write $\mathsf{val} : \langle d \mid \mathsf{Exp}\rangle \to \Omega$ for $\exists v : \langle d \mid \mathsf{Val}\rangle.\, v = -$; informally, is "$e$ a value". We additional write $\mathsf{red} : \langle d \mid \mathsf{Exp}\rangle \to \langle d \mid \mathsf{Heap}\rangle \to \Omega$ for $\exists \rho.\, - \mapsto^{\dagger} \rho$.

$$
\begin{aligned}
&\mathsf{wp}_{\_}\ -\ \{-\} : \{\mathsf{GhostNames}\} \to \mathcal{P}_{\mathsf{(co)fin}}(\mathsf{Nat}) \to \langle d \mid \mathsf{Exp}\rangle \to (\langle d \mid \mathsf{Val}\rangle \to \Omega^*) \to \Omega^*\\
&\mathsf{wp}_{\{\Sigma\};\mathcal{E}}\, e\,\{Q\} =\\
&\quad (\exists v.\, v = e \wedge Q\, v)\\
&\quad \vee\\
&\quad [\ulcorner \neg \mathsf{val}\, e \urcorner \wedge\\
&\quad\ \forall \sigma.\, I\,\sigma \mathrel{-\!\!*}\\
&\qquad {\mid\!\Rrightarrow}_{\mathcal{E};\emptyset}(\ulcorner \mathsf{red}\, e\,\sigma \urcorner \wedge \triangleright \forall e_2, \sigma_2, \vec{e}_f.\, \ulcorner (e,\sigma) \mapsto^{\dagger} (e_2, \sigma_2, \vec{e}_f)\urcorner \mathrel{-\!\!*}\\
&\qquad\quad {\mid\!\Rrightarrow}_{\emptyset;\mathcal{E}}(I\,\sigma_2 * \mathsf{wp}_{\mathcal{E}}\, e_2\,\{Q\} * \mathop{\Large\ast}_{e' \in \vec{e}_f} \mathsf{wp}_{\mathcal{E}}\, e'\,\{\top\}))]
\end{aligned}
$$

At the coarsest level, $\mathsf{wp}_{\mathcal{E}}\, e\,\{Q\}$ is a disjunction stating that if e is a value v then $Q\,v$ and if not, then e is not stuck and whenever it steps to e' then $\triangleright\, \mathsf{wp}_{\mathcal{E}}\, e'\,\{Q\}$. The first case of this disjunction is fairly self-explanatory, so we focus on the second. The most immediate complication is the heap: we cannot just ask whether e is stuck without specifying the heap it is being executed with. For this reason, the definition quantifies over all possible heaps *which satisfy the state interpretation I*. Moreover, the fancy update modalities are arranged so that all invariants may be used to establish that e reduces provided they are re-established before proceeding. Finally, concurrency means that executing e may produce a collection of threads. To ensure that these threads do not crash either, we require that each satisfies $\mathsf{wp}_{\mathcal{E}}\ -\ \{\top\}$.

One rarely actually unfolds the definition of $\mathsf{wp}_{\mathcal{E}}\, e\,\{Q\}$ in practice. Instead, one uses a series of lemmas which give sufficient conditions to establish the predicate depending on the form e takes. For instance, if e is about to perform a step that does not interact with the heap or threads at all e.g. a pure β reduction, it suffices to show that the reduct satisfies the same predicate:

Lemma 10.5.7. *If $e = \mathsf{fixlam}(e_0)\, v$ then the following holds:*

$$\mathsf{wp}_{\mathcal{E}}\, e_0[\mathsf{id}.e.v]\,\{Q\} \mathrel{-\!\!*} \mathsf{wp}_{\mathcal{E}}\, e\,\{Q\}$$

In the above, recall that $\mathsf{fixlam}(e_0)\, v$ steps to $e_0[\mathsf{id}.e.v]$ without altering the heap or producing additional threads (Section 10.2).

Proof. Let us write Ψ for $\mathsf{wp}_{\mathcal{E}}\, e_0[\mathsf{id}.e.v]\,\{Q\}$. Unfolding definitions, we immediately note that it suffices to show that e satisfies the second disjunct. Accordingly, we must show the following:

$$
\begin{aligned}
&\Psi \vdash \forall \sigma.\, I\,\sigma \mathrel{-\!\!*} {\mid\!\Rrightarrow}_{\mathcal{E};\emptyset}(\ulcorner \mathsf{red}\, e\,\sigma \urcorner \wedge \triangleright \forall e_2, \sigma_2, \vec{e}_f.\, \ulcorner (e,\sigma) \mapsto^{\dagger} (e_2, \sigma_2, \vec{e}_f)\urcorner \mathrel{-\!\!*}\\
&\qquad\quad {\mid\!\Rrightarrow}_{\emptyset;\mathcal{E}}(I\,\sigma_2 * \mathsf{wp}_{\mathcal{E}}\, e_2\,\{Q\} * \mathop{\Large\ast}_{e' \in \vec{e}_f} \mathsf{wp}_{\mathcal{E}}\, e'\,\{\top\}))
\end{aligned}
$$

Accordingly, assume we have σ. We note that $\mathsf{red}\, e\,\sigma = \top$ without assumption on σ, so we can equationally discharge this obligation without further burden. After using monotonicity and the point for $\triangleright$, it remains to show the following:

$$
\begin{aligned}
&\Psi * I\,\sigma\\
&\quad \vdash {\mid\!\Rrightarrow}_{\mathcal{E};\emptyset}(\forall e_2, \sigma_2, \vec{e}_f.\, ((e,\sigma) \mapsto^{\dagger} (e_2, \sigma_2, \vec{e}_f))\\
&\qquad \mathrel{-\!\!*} {\mid\!\Rrightarrow}_{\emptyset;\mathcal{E}}(I\,\sigma_2 * \mathsf{wp}_{\mathcal{E}}\, e_2\,\{Q\} * \mathop{\Large\ast}_{e' \in \vec{e}_f} \mathsf{wp}_{\mathcal{E}}\, e'\,\{\top\}))
\end{aligned}
$$

At this point, we note the following equality:

$$\forall e_2, \sigma_2, \vec{e}_f.\, \ulcorner (e, \sigma) \mapsto^{\dagger} (e_2, \sigma_2, \vec{e}_f) \urcorner \mathrel{-\!\!*} \Xi(e_2, \sigma_2, \vec{e}_f) = \Xi(e_0[\mathsf{id}.e.v], \sigma, \epsilon)$$

This follows for the inversion principle for $\mapsto$ in mode s, which induces the following equation in Ω:

$$(e, \sigma) \mapsto^{\dagger} (e_2, \sigma_2, \vec{e}_f) \to (e_0[\mathsf{id}.e.v], \sigma, \epsilon) = (e_2, \sigma_2, \vec{e}_f)$$

At this point, it remains to show the following:

$$\Psi * I\,\sigma \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\emptyset} \mathrel{|\!\!\Rrightarrow}_{\emptyset;\mathcal{E}} (I\,\sigma * \mathsf{wp}_{\mathcal{E}}\, e_0[\mathsf{id}.e.v]\,\{Q\})$$

This then follows by assumption and the rule $\Phi \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\emptyset} \mathrel{|\!\!\Rrightarrow}_{\emptyset;\mathcal{E}} \Phi$. □

Similar proofs hold for other reductions. We show only one other to illustrate the point.

Lemma 10.5.8. *If* $e = {!\,\ell}$ *then the following holds:*

$$(\ell \mapsto v * \mathsf{wp}_{\mathcal{E}}\, v\,\{Q\}) \mathrel{-\!\!*} \mathsf{wp}_{\mathcal{E}}\, e\,\{Q\}$$

Proof. As before, we immediately focus on the right-hand disjunct in the definition of weakest preconditions. Let us write Ψ for $\ell \mapsto v * \mathsf{wp}_{\mathcal{E}}\, v\,\{Q\}$.

It suffices to show the following:

$$\begin{aligned}&\Psi \vdash \forall \sigma.\, I\,\sigma \mathrel{-\!\!*} \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\emptyset} (\ulcorner \mathsf{red}\, e\, \sigma \urcorner \wedge \triangleright \forall e_2, \sigma_2, \vec{e}_f.\, \ulcorner (e, \sigma) \mapsto^{\dagger} (e_2, \sigma_2, \vec{e}_f) \urcorner \mathrel{-\!\!*} \\ &\qquad \mathrel{|\!\!\Rrightarrow}_{\emptyset;\mathcal{E}} (I\,\sigma_2 * \mathsf{wp}_{\mathcal{E}}\, e_2\,\{Q\} * \mathop{\Large\ast}_{e' \in \vec{e}_f} \mathsf{wp}_{\mathcal{E}}\, e'\,\{\top\}))\end{aligned}$$

Accordingly, we fix σ. Let us note that $I\,\sigma * \ell \mapsto v$ implies that $\ulcorner \sigma\,\ell = v \urcorner$. We then are left with the following sequent:

$$\begin{aligned}&\Psi * I\,\sigma * \ulcorner \sigma\,\ell = v \urcorner \vdash \\ &\qquad \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\emptyset} (\ulcorner \mathsf{red}\, e\, \sigma \urcorner \wedge \triangleright \forall e_2, \sigma_2, \vec{e}_f.\, \ulcorner (e, \sigma) \mapsto^{\dagger} (e_2, \sigma_2, \vec{e}_f) \urcorner \mathrel{-\!\!*} \\ &\qquad \mathrel{|\!\!\Rrightarrow}_{\emptyset;\mathcal{E}} (I\,\sigma_2 * \mathsf{wp}_{\mathcal{E}}\, e_2\,\{Q\} * \mathop{\Large\ast}_{e' \in \vec{e}_f} \mathsf{wp}_{\mathcal{E}}\, e'\,\{\top\}))\end{aligned}$$

In particular, under this last assumption $\mathsf{red}\, e\, \sigma = \top$ so we may discharge this immediately. Using the rules for fancy updates and the later modality as well as the same inversion technique as the previous lemma, therefore, we further simplify:

$$\Psi * I\,\sigma * \ulcorner \sigma\,\ell = v \urcorner \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\emptyset} \mathrel{|\!\!\Rrightarrow}_{\emptyset;\mathcal{E}} (I\,\sigma * \mathsf{wp}_{\mathcal{E}}\, v\,\{Q\})$$

The conclusion now follows immediately from monotonicity. □

While proving a full version of adequacy is the subject of the next section, we are already to prove one fragment of desired conclusion: if $\mathsf{wp}_{\mathcal{E}}\, e\,\{Q\}$ holds then e is either a value or reduces.

Lemma 10.5.9. *The following entailment holds*

$$\mathsf{wp}_{\mathcal{E}}\, e\,\{Q\} * I\,\sigma \vdash \mathrel{|\!\!\Rrightarrow}_{\mathcal{E};\emptyset} ((\exists v.\, v = e * Q\, v) \vee \ulcorner \mathsf{red}(e, \sigma) \urcorner)$$

Proof. This is an immediate consequence of weakening after unfolding $\mathsf{wp}_{\mathcal{E}}\, e\,\{Q\}$. □

10.6 Adequacy

The final stage of our development of synthetic Iris is the crucial *adequacy* theorem. Recall that when $\mathsf{wp}_{\mathcal{E}}\, e\, \{Q\}$ was defined, the rough intention was that it ought to signify that (1) e was not stuck and (2) if e terminated, the result satisfied Q. However, the definition of $\mathsf{wp}_{\mathcal{E}}\, e\, \{Q\}$ is far from simple and it is non-obvious that this is actually the case. The role of the adequacy theorem is to turn the imprecise intuition above into a fully precise theorem.

A crucial caveat to the adequacy theorem, however, is the fact that it cannot be stated without something akin to $\langle g \mid - \rangle$. In particular, the presence of $\blacktriangleright$ (in the guise of $\triangleright$) ensures that adequacy cannot be detected except on global elements. Accordingly, this is the place where truly working with two distinct modes is necessary: $\mathsf{wp}_{\mathcal{E}}\, e\, \{Q\}$ must be defined at mode t and adequacy must be stated and proven at mode s.

We decompose the adequacy proof into two stages. First, working within Ω^* and mode t, we show that (1) if a program satisfies $\mathsf{wp}_{\mathcal{E}}\, e\, \{Q\}$ it is not stuck and (2) given an execution trace for a program e executing to e' in k steps, one can "advance" $\mathsf{wp}_{\mathcal{E}}\, e\, \{Q\}$ to obtain $\mathsf{wp}_{\mathcal{E}}\, e'\, \{Q\}$ under some combination of modalities. After this, we switch to working with Ω in mode s where we show that the aforementioned results are sufficient to derive a full soundness result.

10.6.1 Stepping weakest preconditions

Convention 10.6.1. Within this subsection, we continue to work in mode t and assume that we are given $\Sigma : \mathsf{GhostNames}$.

Notation 10.6.2. When writing large entailments of the form $\Phi_0 * \Phi_1 \cdots \vdash \Psi_1 * \Psi_1 \ldots$ we will use occasionally use inference notation for clarity and write the following instead:

$$\frac{\Phi_0 \qquad \Phi_1 \qquad \ldots}{\Psi_0 \qquad \Psi_1 \qquad \ldots}$$

Our first step towards adequacy will be a result showing that given the necessary hypotheses, we can "advance" a weakest precondition. In particular, we will show the following entailment is valid:

$$\frac{I\,\sigma \qquad \mathsf{wp}_{\mathcal{E}}\, e_1\, \{Q\} \qquad \mathop{\Huge\ast}_{i>1} \mathsf{wp}_{\mathcal{E}}\, e_i\, \{\top\} \qquad \ulcorner((e_1, \vec{e_i}), \sigma) \rightsquigarrow^{\dagger} ((e_1', \vec{e_j'}), \sigma')\urcorner}{\Rrightarrow_{\mathcal{E};\emptyset} \triangleright \Rrightarrow_{\emptyset;\mathcal{E}} (I\,\sigma' * \mathsf{wp}_{\mathcal{E}}\, e_1'\, \{Q\} * \mathop{\Huge\ast}_{j>1} \mathsf{wp}_{\mathcal{E}}\, e_j'\, \{\top\})} \tag{10.1}$$

Lemma 10.6.3. *Eq. (10.1) holds.*

Proof. Using the rules for $\ulcorner - \urcorner$, it suffices to assume $((e_1, \vec{e_i}), \sigma) \rightsquigarrow^{\dagger} ((e_1', \vec{e_j'}), \sigma') : \Omega$ holds and show that the remainder of premises of Eq. (10.1) imply the conclusion. Having assumed this step, we note that $\rightsquigarrow^{\dagger}$ is the image of the inductively-defined proposition $\rightsquigarrow$ under $\langle d \mid - \rangle$. Using the standard inversion lemma for $\rightsquigarrow$ as well as the fact that $\langle d \mid - \rangle$ preserves quotients, products, and equality, we conclude that there are two cases to consider:

1. either $(e_1, \sigma) \mapsto (e_1', \vec{e}_f, \sigma')$ and $((e_1', \vec{e}_j, \vec{e}_f), \sigma') = ((e_1', \vec{e_j'}), \sigma')$,

2. or there exists some $i > 1$ and $(e_i, \sigma) \mapsto (e'_i, \vec{e}_f, \sigma')$ and $((e_1, \vec{e}_{j<i}, e'_i \vec{e}_{j>i}, \vec{e}_f), \sigma') = ((e'_1, \vec{e}'_j), \sigma')$.

We prove the entailment assuming the first case holds as the proof under the assumption of the second is essentially the same. We begin by unfolding $\mathsf{wp}_{\mathcal{E}}\, e_1\, \{Q\}$. Inspecting the disjunction, we note that we can immediately discard the first disjunct where e_1 is a value since e_1 is known to take a step. Accordingly, and using the additional premise $I\, \sigma$, we are able to reduce to the following entailment:

$$\dfrac{\begin{array}{c}\Rrightarrow_{\mathcal{E};\emptyset}(\triangleright \forall e_2, \sigma_2, \vec{e}_f.\, \ulcorner(e, \sigma) \mapsto^{\dagger} (e_2, \sigma_2, \vec{e}_f)\urcorner \mathrel{-\!\!*} \Rrightarrow_{\emptyset;\mathcal{E}}(I\, \sigma_2 * \mathsf{wp}_{\mathcal{E}}\, e_2\, \{Q\} * \mathop{\Large\ast}_{e' \in \vec{e}_f} \mathsf{wp}_{\mathcal{E}}\, e'\, \{\top\})) \\ \mathop{\Large\ast}_{i>1} \mathsf{wp}_{\mathcal{E}}\, e_i\, \{\top\}\end{array}}{\Rrightarrow_{\mathcal{E};\emptyset} \triangleright \Rrightarrow_{\emptyset;\mathcal{E}}(I\, \sigma' * \mathsf{wp}_{\mathcal{E}}\, e'_1\, \{Q\} * \mathop{\Large\ast}_{j>1} \mathsf{wp}_{\mathcal{E}}\, e'_j\, \{\top\})}$$

Using monotonicity along with our assumption $(e_1, \sigma) \mapsto^{\dagger} (e'_1, \vec{e}_f, \sigma')$, we may further reduce to the following:

$$\dfrac{\Rrightarrow_{\mathcal{E};\emptyset}(\triangleright \Rrightarrow_{\emptyset;\mathcal{E}}(I\, \sigma' * \mathsf{wp}_{\mathcal{E}}\, e'_1\, \{Q\} * \mathop{\Large\ast}_{e' \in \vec{e}_f} \mathsf{wp}_{\mathcal{E}}\, e'\, \{\top\})) \qquad \mathop{\Large\ast}_{i>1} \mathsf{wp}_{\mathcal{E}}\, e_i\, \{\top\}}{\Rrightarrow_{\mathcal{E};\emptyset} \triangleright \Rrightarrow_{\emptyset;\mathcal{E}}(I\, \sigma' * \mathsf{wp}_{\mathcal{E}}\, e'_1\, \{Q\} * \mathop{\Large\ast}_{j>1} \mathsf{wp}_{\mathcal{E}}\, e'_j\, \{\top\})}$$

Capitalizing on the framing rules for $\Rrightarrow_{\mathcal{E}_1;\mathcal{E}_2}$ and $\triangleright$, we conclude that it suffices to show that the following tautological inference holds::

$$\dfrac{\Rrightarrow_{\mathcal{E};\emptyset} \triangleright \Rrightarrow_{\emptyset;\mathcal{E}}(I\, \sigma' * \mathsf{wp}_{\mathcal{E}}\, e'_1\, \{Q\} * \mathop{\Large\ast}_{j>1} \mathsf{wp}_{\mathcal{E}}\, e'_j\, \{\top\})}{\Rrightarrow_{\mathcal{E};\emptyset} \triangleright \Rrightarrow_{\emptyset;\mathcal{E}}(I\, \sigma' * \mathsf{wp}_{\mathcal{E}}\, e'_1\, \{Q\} * \mathop{\Large\ast}_{j>1} \mathsf{wp}_{\mathcal{E}}\, e'_j\, \{\top\})}$$

□

The next step is to generalize this result to generalize from advancing the thread pool by once to step advancing it k steps. To state this precisely, some care is required: should k be drawn from $\langle d \mid \mathsf{Nat}\rangle$ or Nat? Fortunately, these two types are isomorphic, so the difference is only one of convenience. We will opt for $\langle d \mid \mathsf{Nat}\rangle$ as it will simplify later results slightly.

Given $k :_d \mathsf{Nat}$, our goal is to prove the following:

$$\dfrac{I\, \sigma \qquad \mathsf{wp}_{\mathcal{E}}\, e_1\, \{Q\} \qquad \mathop{\Large\ast}_{i>1} \mathsf{wp}_{\mathcal{E}}\, e_i\, \{\top\} \qquad \ulcorner((e_1, \vec{e}_i), \sigma) \rightsquigarrow^{\dagger}_{k} ((e'_1, \vec{e}'_j), \sigma')\urcorner}{(\Rrightarrow_{\mathcal{E};\emptyset} \triangleright \Rrightarrow_{\emptyset;\mathcal{E}})^{k}(I\, \sigma' * \mathsf{wp}_{\mathcal{E}}\, e'_1\, \{Q\} * \mathop{\Large\ast}_{j>1} \mathsf{wp}_{\mathcal{E}}\, e'_j\, \{\top\})} \tag{10.2}$$

Lemma 10.6.4. *Eq. (10.2) holds.*

Proof. The crisp induction principle for Nat enables us to proceed by induction, whereby this reduces to repeated application of Eq. (10.1). □

10.6.2 Removing fancy update modalities

The inference Eq. (10.2) is a good step towards the final adequacy theorem, but the continued presence of $\Rrightarrow_{\mathcal{E}_1;\mathcal{E}_2}$ is an issue. Indeed, eventually, we will have to unfold the definitions of various connectives in Ω^* as given in Section 10.4 and use $\langle g \mid -\rangle$ to obtain a concrete proof out of $\mathsf{wp}_{\mathcal{E}}\, e\, \{Q\}$. Doing this with the fancy update modality is not appealing: it is built out of several complex components and involves numerous pieces of ghost state. We therefore set out to unfold the definition of $\Rrightarrow_{\mathcal{E}_1;\mathcal{E}_2}$ and simplify Eq. (10.2).

Convention 10.6.5. We continue to work in mode t, but we do not assume $\Sigma : \mathsf{GhostNames}$. Accordingly, we will explicitly subscript connectives like $\mathsf{wp}_{\Sigma;\mathcal{E}}\, e\, \{Q\}$ with some variable Σ to avoid confusion.

Lemma 10.6.6. *Given $\sigma : \langle d \mid \mathsf{Heap}\rangle$ and $\mathcal{E}$, the following implication holds in Ω^*:*

$$\top \vdash \exists \Sigma : \mathsf{GhostNames}.\, I_\Sigma(\sigma) * \mathsf{WSat}_\Sigma * \boxed{\mathcal{E}}^{\Sigma.\gamma_{\mathsf{en}}}$$

Proof. This essentially follows from repeated application of Lemma 10.4.38. In particular, this lemma yields the following entailments:

$$\begin{aligned}
&\top \vdash \exists \gamma_{\mathsf{en}}.\, \boxed{\mathcal{E}}^{\gamma_{\mathsf{en}}}\\
&\top \vdash \exists \gamma_{\mathsf{heap}}.\, \boxed{\bullet\sigma}^{\gamma_{\mathsf{heap}}}\\
&\top \vdash \exists \gamma_{\mathsf{dis}}.\, \boxed{\emptyset}^{\gamma_{\mathsf{dis}}}\\
&\top \vdash \exists \gamma_{\mathsf{inv}}.\, \boxed{\bullet\emptyset}^{\gamma_{\mathsf{inv}}}
\end{aligned}$$

Repackaging this using the fact that $\top * \top = \top$, we obtain the following:

$$\top \vdash \exists \Sigma.\, \boxed{\mathcal{E}}^{\Sigma.\gamma_{\mathsf{en}}} * \boxed{\bullet\sigma}^{\Sigma.\gamma_{\mathsf{heap}}} * \boxed{\emptyset}^{\Sigma.\gamma_{\mathsf{dis}}} * \boxed{\bullet\emptyset}^{\Sigma.\gamma_{\mathsf{inv}}}$$

The conclusion then follows immediately by unfolding the definitions of I_Σ and WSat_Σ. □

Lemma 10.6.7. *Given $\Sigma : \mathsf{GhostNames}$, the following entailment holds:*

$$\mathsf{WSat}_\Sigma * \boxed{\mathcal{E}_1}^{\Sigma.\gamma_{\mathsf{en}}} * \mathrel{\vert\!\!\Rrightarrow}_{\mathcal{E}_1;\mathcal{E}_2} \Phi \vdash \mathrel{\vert\!\!\Rrightarrow}(\mathsf{WSat}_\Sigma * \boxed{\mathcal{E}_2}^{\Sigma.\gamma_{\mathsf{en}}} * \Phi)$$

Proof. This follows immediately after unfolding the definition of $\mathrel{\vert\!\!\Rrightarrow}_{\mathcal{E}_1;\mathcal{E}_2}$. □

Lemma 10.6.8. *Given $k :_d \mathsf{Nat}$, the following entailment holds:*

$$\begin{aligned}
&(\forall \Sigma : \mathsf{GhostNames}.\, \mathsf{wp}_{\Sigma;\mathcal{E}}\, e_1\, \{Q\}) * \ulcorner (e,\sigma) \rightsquigarrow^{\dagger}_{k} ((e_1, \vec{e_j}), \sigma')\urcorner\\
&\vdash \exists \Sigma.\, (\mathrel{\vert\!\!\Rrightarrow} \triangleright \mathrel{\vert\!\!\Rrightarrow})^k(\mathsf{WSat}_\Sigma * \boxed{\mathcal{E}}^{\Sigma.\gamma_{\mathsf{en}}} * I_\Sigma\, \sigma' * \mathsf{wp}_{\Sigma;\mathcal{E}}\, e_1\, \{Q\} * \mathop{\Large\ast}_{j>1} \mathsf{wp}_{\Sigma;\mathcal{E}}\, e_j\, \{\top\})
\end{aligned}$$

Proof. This follows by combining Eq. (10.2) with Lemmas 10.6.6 and 10.6.7. □

Let us introduce the following notation to signify that a program either reduces or satisfies its post-condition:

$$\begin{aligned}
&\mathsf{sat} : \langle d \mid \mathsf{Exp}\rangle \to \langle d \mid \mathsf{Heap}\rangle \to (\langle d \mid \mathsf{Val}\rangle \to \Omega^*) \to \Omega^*\\
&\mathsf{sat}\, e\, \sigma\, Q = (\exists v.\, v = e * Q\, v) \vee \ulcorner \mathsf{red}(e,\sigma)\urcorner
\end{aligned}$$

Theorem 10.6.9. *Given $k :_d \mathsf{Nat}$, the following proposition is equal to $\top$ in Ω^*:*

$$\begin{aligned}
&(\forall \Sigma : \mathsf{GhostNames}.\, \mathsf{wp}_{\Sigma;\mathcal{E}}\, e_1\, \{Q\}) * \ulcorner (e,\sigma) \rightsquigarrow^{\dagger}_{k} ((e_1, \vec{e_j}), \sigma')\urcorner\\
&\quad \mathrel{-\!\!*} \exists \Sigma : \mathsf{GhostNames}.\, (\mathrel{\vert\!\!\Rrightarrow} \triangleright \mathrel{\vert\!\!\Rrightarrow})^k \mathsf{sat}(e_l, \sigma, P_l)
\end{aligned}$$

In the above formula, $1 \leq l \leq j$ and $P_l = Q$ if $l = 1$ and $\top$ otherwise.

Proof. This follows from Lemmas 10.5.9 and 10.6.8. □

10.6.3 The adequacy theorem

Theorem 10.6.9 is nearly the desired adequacy theorem: it states that any execution of a thread pool either continues to execute or satisfies the necessary post-condition. Unfortunately, this is still a statement within the logic of Ω^* and when applied to a k-step trace yields a result under $3k$ modalities. In this section, we take advantage of the equivalence $\langle g \mid \blacktriangleright A\rangle \simeq \langle g \mid A\rangle$ to address this last point. In particular, we show in the situation where the post-conditions are defined using propositions from Ω in mode s, one may conclude that after k steps of execution, the resulting thread pool either can continue to run or satisfies the expected post-conditions in mode s.

Convention 10.6.10. Within this subsection, we work in mode s.

Definition 10.6.11. Define psat (pronounced *pure satisfies*) as follows:

$$\begin{aligned}&\mathsf{psat} : \mathsf{Exp} \to \mathsf{Heap} \to (\mathsf{Val} \to \Omega) \to \Omega\\ &\mathsf{psat}\, e\, \sigma\, Q = (\exists v.\, v = e \wedge Q\, v) \vee ((e, \sigma) \mapsto \_)\end{aligned}$$

Lemma 10.6.12. *Given $Q : \mathsf{Val} \to \Omega$, the following identification holds:*

$$\langle g \mid \ulcorner\langle d \mid \mathsf{psat}(e, \sigma, Q)\rangle\urcorner = \mathsf{sat}(\mathsf{mod}_d(e), \mathsf{mod}_d(\sigma), \lambda v.\, \mathsf{let}\ \mathsf{mod}_d(-) \leftarrow v\ \mathsf{in}\ v_0\langle d \mid Q\, v_0\rangle)\rangle$$

Proof. By propositional univalence of Ω^*, it suffices to show that these two propositions imply each other. Unfolding psat and sat, this amounts to the fact that $\langle d \mid -\rangle$ and $\ulcorner - \urcorner$ commute with disjunction, conjunction, and existential quantification. □

We now set out to remove the $\Rrightarrow$ and $\triangleright$ modalities annotating the conclusion of Theorem 10.6.9. We begin with the following observation:

Lemma 10.6.13. *The following identifications hold for any $\phi : \Omega$:*

- $\langle g \mid \top \vdash \triangleright\ulcorner\phi\urcorner\rangle = \langle g \mid \top \vdash \ulcorner\blacktriangleright\langle d \mid \phi\rangle\urcorner\rangle$
- $\langle g \mid \top \vdash \ulcorner\phi\urcorner\rangle = \langle g \mid \langle d \mid \phi\rangle\rangle = \phi$
- $\langle g \mid \top \vdash \Rrightarrow\ulcorner\phi\urcorner\rangle = \langle g \mid \top \vdash \ulcorner\langle d \mid \phi\rangle\urcorner\rangle$

Proof. All three points hold by computation. In particular, because $\ulcorner\langle d \mid \phi\rangle\urcorner$ is a constant function it is unaffected by $\Rrightarrow$. □

Theorem 10.6.14. *Fix $e : \mathsf{Exp}$ along with a proposition $Q : \mathsf{Val} \to \Omega$. Suppose that we are given a proof of the following proposition W:*

$$\begin{aligned}&Q' :_g \langle d \mid \mathsf{Val}\rangle \to \Omega^*\\ &Q'\, v = \mathsf{let}\ \mathsf{mod}_d(v_0) \leftarrow v\ \mathsf{in}\ \ulcorner\langle d \mid Q\, v_0\rangle\urcorner\end{aligned}$$

$$W = \langle g \mid \top \vdash \forall\Sigma : \mathsf{GhostNames}.\, \mathsf{wp}_{\Sigma;\top}\, \mathsf{mod}_d(e)\, \{Q'\}\rangle$$

Let us further suppose that we are given some k : Nat along with a proof of the following $(e, \emptyset) \rightsquigarrow_k (e_1, \vec{e}_f, \sigma)$. In this case, the following holds:

$$\mathsf{psat}(e_1, \sigma, Q) \wedge \bigwedge_{j>1} \mathsf{psat}(e_j, \sigma, \top)$$

Proof. It suffices to show that $\mathsf{psat}(e_l, \sigma, P_l)$ holds for all j, where $P_l = Q$ if $j = 1$ and $\top$ otherwise. We will show only the case for $l = 1$, as the other cases are similar. Using axiom K on Theorem 10.6.9 together with our assumptions of W and $(e, \emptyset) \rightsquigarrow_k (e_1, \vec{e}_f, \sigma)$, we conclude that the following proposition holds:

$$\langle g \mid \top \vdash (\mathrel{|\!\!\Rrightarrow} \triangleright \mathrel{|\!\!\Rrightarrow})^k(\mathsf{sat}(\mathsf{mod}_d(e_1), \mathsf{mod}_d(\sigma), \mathsf{mod}_d(\sigma), Q')) \rangle$$

Note that, in particular, we are able to discard the $\exists \Sigma : \mathsf{GhostNames}$ as none of the sat predicates mention this connective. Using Lemma 10.6.12, we are able to replace all occurrences of sat with psat to obtain the following:

$$\langle g \mid \top \vdash (\mathrel{|\!\!\Rrightarrow} \triangleright \mathrel{|\!\!\Rrightarrow})^k(\ulcorner \langle d \mid \mathsf{psat}(e_1, \sigma, Q) \rangle \urcorner) \rangle$$

Next, we use Lemma 10.6.13 $3k + 1$ times to remove all mention of Ω^* and obtain the following:

$$\langle g \mid \blacktriangleright^k \langle d \mid \mathsf{psat}(e_1, \sigma, Q) \rangle \rangle$$

Finally, using the identity $\langle g \mid \blacktriangleright A \rangle \simeq \langle g \mid A \rangle$, we obtain the adequacy theorem. □

10.7 Conclusions

This chapter has introduced a reconstruction of the Iris program logic internal to MTT tuned for guarded recursion. While the definitions in this section closely follow those presented in Iris, our synthetic approach *automatically* yields new models of Iris. For instance, while Spies et al. [Spi+21] showed that one can interpret Iris in (essentially) sheaves over ordinals other than ω, synthetic Iris can be interpreted into sheaves over any well-founded Heyting algebra. While these additional models may prove to be important in the future, presently they are mostly a tangential benefit.

The main motivation and pay-off of synthetic Iris is pedagogical: working synthetically allows us to crisply and systematically avoid technical details. For instance, Section 10.4 defines the highly recursive type of propositions in Iris but without recourse to domain equations or reasoning about steps or anything similar. Every construction and definition is carried out as it would be in ordinary mathematics, and Löb induction is used once to tie things together. The presence of propositional univalence allows us to simplify various arguments without attempting to reconstruct the complex machinery of type classes and setoids used in Iris to mimic this principle.

Working at this level also highlights and clarifies the odd behavior of cameras in Iris. The category of resource algebras introduced in this chapter is new, as is the recognition of the universal properties of the agreement and exclusive resource algebras. We are also able to crystallize the deficiencies of the present definition of resource algebra: for instance, the lack of unital completions and the failure of $\bigoplus$ to even be functorial. Further exploration within synthetic Iris offers a path to finding a more uniform and well-behaved definition of resource algebras, which we believe will be of substantial benefit to both synthetic and classical Iris.

Presently, the lack of a good implementation of modal type theory precludes this approach from replacing the Coq development of Iris. If a comparable implementation of MTT existed, we also believe that these simpler definitions would lend themselves to a more tractable formalization.

Part IV

Conclusions and outlook

11 Conclusions

> This is very interesting, but not every functor is a right adjoint.
>
> Paige Randall North
> 2019

We have introduced MTT the first fully general modal dependent type theory capable of encapsulating arbitrary interacting modalities. In so doing, we have laid out a systematic and general approach to modal type theory centered around *weak dependent right adjoints*. We have also presented evidence for the usability and flexibility of MTT and weak dependent right adjoints; both by proving important metatheorems (canonicity, normalization) and providing numerous in-depth case studies applying MTT to adjoint modalities, guarded recursion, and a synthetic account of Iris.

11.1 Related Work

While we have discussed related work in each chapter, we now provide a more thorough discussion of the related work most closely connected to MTT.

11.1.1 Multimodal adjoint type theory

By far the most closely related type theory to MTT is *multimodal adjoint type theory* or MATT [Shu23]. This is no accident: MATT was designed to be a refinement of MTT which provides systematic control over the addition of *strict dependent right adjoints* as described in Section 6.3 while also allowing the user to exclude certain crisp modal induction principles. Thus, while in this thesis we have adopted an ad-hoc attitude towards these extensions, in MATT they are described alongside the mode theory when instantiating the type theory.

These considerations are reverse-engineered from the remarkable strictification theorem proven in the same paper introducing MATT. As was shown in Section 7.2.4, one can build a model of MTT with the adjoint mode theory from just a pair of lex adjoint functors by first gluing along the left adjoint—provided one is willing to sacrifice the elimination for the left adjoint modality when framed by the right adjoint. Shulman [Shu23] introduces *codextrification* a vast generalization of this construction that applies not just to a single adjoint, but to an arbitrary 2-functor $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ provided that $F(\mu)$ is suitably continuous.

Theorem 11.1.1 (Section 4 [Shu23]). *Fix a κ-small 2-category $\mathcal{M}$ for some regular cardinal κ and denote the 2-category of κ-complete categories and κ-continuous functors by $\mathbf{Lex}_\kappa$. Given a 2-functor $F : \mathcal{M} \longrightarrow \mathbf{Lex}_\kappa$, there exists a 2-functor $G : \mathcal{M} \longrightarrow \mathbf{Lex}_\kappa$ and a 2-natural transformation $\pi : G \longrightarrow F$ satisfying the following properties:*

1. *For each $m : \mathcal{M}$, the functor $\pi(m)$ is κ-continuous and has a fully-faithful right adjoint.*

2. *For each morphism μ, the functor $G(\mu)$ is a right adjoint.*

Remark 11.1.2. More than this is true: if each $F(m)$ is locally Cartesian closed, presentable, an elementary topos etc. then the same will be true of each $G(m)$. Moreover, G can be given a universal property but it is not required for what follows. $\diamond$

Remark 11.1.3. The precise details of the construction are not relevant for the following discussion, but it may be helpful to know that codextrification is realized by a collection of op-lax limits analogous to how $\mathbf{Gl}(F)$ may be realized as an op-lax limit. $\diamond$

In Section 7.2.3, we described a special case of an extension of the local universes coherence theorem for MTT from Shulman [Shu23]. The general case is precisely designed to work with codextrification.

Theorem 11.1.4 (Section 5 [Shu23]). *Under the same assumptions as Theorem 11.1.1, if $F : \mathcal{M} \longrightarrow \mathbf{Lex}_\kappa$ satisfies the assumptions of Theorem 7.2.21, then the codextrification G extends to a model of MTT.*

We note, crucially, that unlike in Theorem 7.2.21 we do not require $F(\mu)$ to have a left adjoint. Instead, we use codextrification to (co)freely add a right adjoint and interpret MTT in the resulting 2-functor.

All of the results discussed thus far apply to MTT (and therefore to a special case of MATT). The motivation for the generality afforded by MATT comes from the requirement that the functors involved must be κ-continuous for some cardinal κ such that $\mathcal{M}$ is κ-small. If $\mathcal{M}$ is finite, this requirement is quite reasonable but it does fail if $\mathcal{M}$ is infinite. Unfortunately, most mode theories of interest (e.g., the adjoint mode theory and its sub-mode theories) are at least countably infinite.

Shulman observed that in many circumstances one did not need to apply codextrification to the entirety of $\mathcal{M}$ if one was willing to eschew certain crisp modal induction principles. In particular, if a modality $\mu \in \mathcal{M}$ was right adjoint to some ν one could omit μ when codextrifying.

Definition 11.1.5. Given a 2-category $\mathcal{C}$ and a subset S of 1-cells, we denote by $\mathcal{C}[S_*]$ the 2-category which freely adds right adjoints to each of the 1-cells in S. We write s_* for the freely added right adjoint to $s \in S$.

Shulman [Shu23] then shows that one may take a functor $F : \mathcal{C}[S_*] \longrightarrow \mathbf{Lex}_\kappa$ and then consider the codextrification G of the restriction of F to $\mathcal{C}$. Remarkably, the model of MTT on G provided by Theorem 11.1.4 automatically inherits strict dependent right adjoints for each of the morphisms with S and so G interprets a version of MTT with mode theory $\mathcal{C}[S_*]$ but without e.g. crisp induction principles using s_* as a framing modality. Importantly, many infinite mode theories can be presented as $\mathcal{C}[S_*]$ for some *finite* $\mathcal{C}$ (e.g., $\mathcal{M}_{\mathsf{adj}}$), thereby alleviating much of the burden imposed by κ-continuity. For instance, this allows MATT to capture the multiplicity of cohesive structures considered by Myers and Riley [MR23].

The extra features of MATT are precisely designed to capture these additional right adjoints which exist prior to codextrification. As Shulman [Shu23] introduced MATT during the preparation of this thesis, we have studied where some of the extensions and features provided by MATT can be included into MTT without undue alteration. For instance, in Section 8.7, we have shown that normalization algorithm for MTT can be extended to support the strict dependent right adjoints present in MATT without additional effort. Similarly, in Section 6.4, we analyzed some of the behavior of internal right adjoints without assuming certain crisp modal induction principles; showing that up to definitional equality, the stricter rules provided by MATT can be derived from the weaker ones available in MTT.

Thus, it appears that much of MATT can be recovered by either considering an extension of MTT with a smaller mode theory as was done by Shulman [Shu23] *or* by considering a restriction of MTT with a larger mode theory (in the style of Section 7.2.4). Further study is needed to more precisely characterize the relationship between MTT and MATT. We expect, however, that "MTT 2.0" will include at least some of the additional flexibility afforded by MATT to fully capitalize on codextrification and the new models such it offers [MR23].

11.1.2 The Licata-Shulman-Riley framework

While MTT was the first general framework for modal dependent type theory with multiple modalities, prior work by Licata et al. [LSR17] introduced another framework (referred to simply as LSR) for a simply-typed system that allowed for arbitrary collections of modalities along with arbitrary variation of the structural principles at each mode. While each mode in MTT behaves like ordinary Martin-Löf type theory, each mode in LSR can behave like the simply-typed lambda calculus, an intuitionistic linear calculus, or some more exotic substructural theory. More than the remarkable flexibility in each mode theory, LSR also internalizes more modalities than MTT. Roughly, while the semantics of MTT are easiest to set up using adjunctions and interpreting modalities using the right adjoints, LSR captures both the left and the right adjoints as modalities.

To accommodate this generality, the syntax for LSR is far more cumbersome than that of MTT. Much of the effort in MTT was to ensure that modalities did not require cumbersome annotations or delayed substitutions, but LSR must use more extensive notations both for variable access and modal rules. In fact, Licata et al. [LSR17] explained that one might use LSR to produce a reference calculus for which one then proves equivalent to a more convenient hand-crafted syntax. This goal is quite different from MTT, which aims to be usable out-of-the-box.

It therefore seems that LSR and MTT occupy incomparable points in the design space (even setting aside the question of dependence): the former is more general while the latter is more usable.

11.1.3 Fitch/Kripke-style modal type theories & uniform modal transformations

In Chapter 5, we spent a great deal of time analyzing the poor behavior of the naïve elimination rule in Fitch/Kripke-style type theories. Given our focus on constructing a general modal type theory, we did not focus on any of the specific workarounds used for particular modalities. Given the convenience of the elimination rule however, such explorations may well be worth it when analyzing modal type theories that are likely to be used again and again [Bir+20; GSB19a; HP23; VRT22].

This process is often arduous and syntactic. The essential goal is to prove that the adjoint action on the syntactic category of contexts is a parametric right adjoint and this necessitates careful analysis of the modality's adjoint action and the structure of the syntactic category of contexts. However, this process has been streamlined since the work of Birkedal et al. [Bir+20] and Gratzer et al. [GSB19a] which proved these results for modal type theories with a plain dependent right adjoint or an idempotent comonad, respectively. In particular, Valliappan et al. [VRT22] and Hu and Pientka [HP23] both gave parameterized Fitch-style type theories allowing the $\Box$ modality included in their type theories to be tuned to satisfy additional properties.

Hu and Pientka [HP23], in particular, showed that substitutions could be placed in a certain normal form—a uniform modal transformation (UMoT)—which ensured that each substitution was organized into a list of terms for each modal zone. This normal form enables a vastly more streamlined construction of the elimination rule and the proof of its substitution lemma since the relevant PRA structure is then easy to define. Hu and Pientka [HP23] and Valliappan et al. [VRT22] consider only calculi with a single modality which makes them unsuitable for the same goals as MTT. That said, the notion of UMoT described by Hu and Pientka [HP23] is worth careful investigation in its own right.

In particular, op. cit. has shown that these UMoTs can be used to adapt classical untyped normalization-by-evaluation [Abe13] to work with a modal calculus with a non-idempotent comonad. This approach seems useful for circumventing the complications encountered by Stassen et al. [SGB23a] which forced op. cit. to limit their implementation of modalities to poset-enriched mode theories.

11.1.4 Clocked Type Theory

In a separate strand of work, Bahr et al. [BGM17] and Kristensen et al. [KMV22] have introduced and refined clocked type theory CloTT, a Fitch-style calculus for guarded recursion. Once again, their calculus is limited to one modality and therefore oriented in a different direction than MTT but again several features of CloTT bear further study.

It remains an open and interesting question where the convenient *tick syntax* of CloTT could be adapted to MTT. This could alleviate some of the pain of writing programs in MTT which use complex 2-cell manipulations, such as those in Section 6.4. Preliminary steps in this direction have been undertaken by Nuyts [Nuy23]. In addition, clocked type theory features *indexed* modalities—each $\blacktriangleright$ modality is indexed by a clock which may be abstracted over and instantiated. There is no corresponding feature in MTT where modalities are fixed in the mode theory and do not change throughout a proof. Many natural situations in programming languages are best explained by first-class or indexed modalities and a proper theory of these phenomena remains elusive. The semantics of CloTT [MMV20] suggest that a necessary criterion for indexed modalities is "independence" between the indexing type and the modalities; concretely, the semantics dependent crucially on the commutation of the left adjoint to $\blacktriangleright$ and the type of clocks.

11.2 Open questions and future work

Having made it this far in the thesis, the reader may be forgiven for having had their fill of modal type theory. If however, they have the stomach for it, many interesting questions and conjectures remain. In this final section, we detail a few open questions and conjectures regarding MTT, weak dependent right adjoints, and applications thereof.

11.2.1 Open questions on weak dependent right adjoints

Despite the results established in this thesis and those covered by Shulman [Shu23], questions about weak dependent right adjoints remain.

Conjecture 11.2.1. *(Extensional) DRA (Section 5.5) is a conservative extension of (extensional) MTT with a single endomodality when restricted to closed terms. In other words, given a closed type A in MTT with one endomodality, A inhabited just when $[\![A]\!]$ is inhabited in DRA.*

This conjecture is essentially answered in the affirmative for the simply-typed case by Davies and Pfenning [DP01]. However, adapting their translation procedure to accommodate dependent types is far from trivial. It seems almost certain that additional commuting conversions must be added to MTT, so an intermediate step would be to study the extensional case. We note that it seems likely that the conjecture is *false* in the presence of infinitary products (see Lemma 6.4.9).

Conjecture 11.2.2. *Consider MTT instantiated with $\mathcal{M}_{\mathsf{adj}}$ and extended with univalence [Uni13] at each mode. The internal right adjoint satisfies crisp identity induction.*

Recall that (1) the left adjoint satisfies crisp identity induction (Lemma 6.4.13) and (2) any modality satisfies this property in the presence of extensional equality (Lemma 6.3.4). In other work, Aagaard et al. [Aag+22] shows that every modality in cubical MTT satisfies crisp identity induction. However, it is non-obvious if this will continue to hold in MTT extended with univalence. Note that while Lemma 7.3.2 appears to refute this conjecture, only one of the two modes in this model satisfies univalence.

Conjecture 11.2.3. *There exists an infinite mode theory $\mathcal{M}$ and a strict 2-functor $F : \mathcal{M} \longrightarrow \mathbf{Lex}_\omega$ such that there does not exist any $G : \mathcal{M} \longrightarrow \mathbf{Lex}_\omega$ and $\pi : G \longrightarrow F$ satisfying the conclusion of Theorem 11.1.1.*

This conjecture is meant to express that Shulman [Shu23] is *tight* in a certain sense; one cannot relax the continuity hypotheses in general. This question is purely categorical, but it has an impact on the construction of models of MTT in light of Theorem 11.1.4.

Conjecture 11.2.4. *Given a closed (pre)type $\mathfrak{C}$ and a modality μ, MTT extended with the definitional equality $\Gamma.\{\mu\} = \Gamma.\mathfrak{C}$ satisfies canonicity and normalization.*

Surprisingly, it turns out that applications of MTT have models which satisfy such equalities [Cav21; ND21] and these examples would be improved if such equalities could be conveniently internalized. More generally, a theory of MTT extended with computational left adjoints appears to have many different uses. For instance, Kolomatskaia and Shulman and Gratzer, Sterling, and Birkedal are making use of use of other "computational modalities" to better reason about augmented semisimplicial spaces and guarded recursion, respectively.

Finally, while it is vaguer than previous conjectures, we note that there appear to be interesting applications of some form of *modal extension types*. These extension types ought to generalize ordinary extension types [RS17] from the modality $\mathbb{I} \to -$ to apply to arbitrary modalities.

Conjecture 11.2.5. *There exists a generalization of extension types to arbitrary modalities and the corresponding extension of MTT satisfies canonicity and normalization.*

Presently no general proposal of such types exist, but important special cases are described by Cavallo [Cav21] and Nuyts and Devriese [ND21].

11.2.2 Substructural modal type theories

By design, we have chosen to focus only on modalities but in so doing we have ignored the rich and fruitful interaction between modalities and substructural type theories. The reason for the omission is entirely pragmatic: substructural dependent type theory is a complex topic even without modalities with applications both within computer science and mathematics [CP02; KPB15; Ril22].[1] However, many approaches to substructural type theory naturally lead to modalities. For instance, quantitative type theory, one promising approach to substructural dependent type theory crucially uses modalities to handle the linearity [Atk18]. Just as MTT annotates each variable with a modality to control its usage, quantitative type theory annotates each variable with a quantity for the same purpose.

A variety of calculi exploring similar ideas in the form of *graded modalities* have been proposed [AB20; ADE23; MEO21]. These theories focus on using modalities to control the use of variables within a program; whether they can be erased, are used parametrically, linearly, etc. In particular, a fully-featured dependent type theory based on these ideas is presented in detail by Moon et al. [MEO21].

Presently the scope of these graded modalities is almost disjoint from the modalities studied in this thesis. Given that both have numerous interesting applications, however, it is natural to wonder whether it is possible to present a single theory that includes both. We note that substructural modalities have numerous applications within mathematics [LSR17; Ril22] and a "substructural MTT" may be quite useful.

11.2.3 Applications to synthetic mathematics

As continuously mentioned, modal types theories are best judged through their applications. While we have focused primarily on guarded recursion in Part III, we highlight two other promising potential applications of MTT to synthetic mathematics.

[1] In particular, we refer the reader to Section 1.7 of Riley [Ril22] for a discussion of linear dependent type theory with an eye towards mathematical applications.

Synthetic algebraic geometry Cherubini et al. [CCH23] have proposed a new foundation for synthetic algebraic geometry. This work extends that of Blechschmidt [Ble17] by generalizing an internal logic to an internal (univalent) type theory. Given that Cherubini et al. [CCH23] strive to axiomatize the little Zariski topos over a commutative ring k, there appear to be many natural modalities one might consider:

- One might consider a version of cubical MTT [Aag+22] where one mode plays the role of the little Zariski ∞-topos and the other models ∞-groupoids. Modalities representing e.g. the global sections functor and the discrete functor should then be useful in relating the internal construction of cohomology proposed by Blechschmidt et al. [BCW] to a more classical external definition.
- Given two commutative rings linked by a homomorphism $k_0 \longrightarrow k_1$, there is an induced geometric morphism between Zariski ∞-topoi. Capturing some of these adjoints offers the possibility of axiomatizing or even proving some of the descent properties commonly used within algebraic geometry.

Synthetic ∞-category theory Given the complexity of various models of ∞-categories, a synthetic account of ∞-category theory is an appealing prospect. We will single out one strand of research in this direction: *simplicial type theory* [RS17; Wei22] and its cubical variant [WL20]. Both of these extend normal homotopy type theory a *directed interval* Δ^1. This interval mirrors the interval $\mathbb{I}$ crucially featured in cubical type theory [Coh+17] but, while $\mathbb{I}$ is meant to axiomatize the topological interval $[0, 1]$, simplicial type theory uses Δ^1 to axiomatize the category generated by a single arrow $\{\bullet \longrightarrow \bullet\}$. Just as was done in cubical type theory, one may use Δ^1 to carve out ∞-categories from types as those which are suitably fibrant.

A central contribution of Cohen et al. [Coh+17] is to construct a fibrant universe that classifies small fibrant types. Constructing a similar type in the context of simplicial type theory remains an important and interesting question. In plain terms, such a type $\mathcal{U}$ would be an ∞-category of small ∞-categories such the space of morphisms[2] from A to B in $\mathcal{U}$ corresponds to the ∞-groupoid of functors $A \longrightarrow B$. This last property is often termed *directed univalence.*

A promising first step towards constructing $\mathcal{U}$ is given by Weaver and Licata [WL20] for the cubical version of simplicial type theory. They adapt the methodology of Licata et al. [Lic+18] to construct an ∞-category of small ∞-groupoids satisfying directed univalence. There is ongoing work to adapt this construction to account for all small ∞-categories. There is still reason to hope for such construction for ordinary simplicial type theory rather than the cubical variant: the former is known to have models in which synthetic ∞-categories precisely correspond to classical ∞-categories but this remains an open question for the latter.

Unfortunately, adapting Licata et al. [Lic+18] in the context of ordinary simplicial type theory faces substantial technical obstacles. In the standard models of simplicial type theory, Δ^1 is not tiny and the tininess of $\mathbb{I}$ was essential for Licata et al. [Lic+18]. Weinberger, Buchholtz, Riehl, and collaborators have proposed to circumvent this by working not only with simplicial spaces—the standard model of simplicial type theory—but simultaneously with the category of cubical spaces into which simplicial spaces embed.

[2] Recall that we are speaking of ∞-categories so there is a proper space of morphisms

The interval is tiny in cubical spaces (this was used by Weaver and Licata [WL20]) and one could use the essential geometric morphism relating these two categories to carry out pieces of the construction in cubical spaces where necessary.

It seems likely that MTT would be useful in this situation: a version of MTT extended with univalence could internalize this adjunction along with the global sections functor and the amazing right adjoint to Δ^1 in bicubical sets. Taken all together, this structure should permit one to carry out the construction of $\mathcal{U}$ for simplicial type theory purely internally; generalizing what was done by Licata et al. [Lic+18].

Moreover, a central complication of synthetic category theory is that categories—unlike sets or ∞-groupoids—have a non-trivial automorphism in the form of $\mathcal{C} \mapsto \mathcal{C}^{\mathsf{op}}$. In simplicial type theory, it seems useful to have this available not only on fibrant types (proper ∞-categories) but on all types. This is straightforward to accomplish with MTT, as in the standard models $-^{\mathsf{op}}$ is realized by a morphism which is immediately seen to be a dependent right adjoint.

Taken together, this suggests that a fusion of MTT with simplicial type theory may be well-adapted both to construct the ∞-category of ∞-categories and carrying out arguments in synthetic category theory.

11.2.4 A logical framework for modal type theories

At several points throughout this thesis, we have had to apologize for the technicalities introduced by the lack of a good logical framework for modal type theory. It has led to technical excursions in our proofs of normalization and guarded canonicity as well as a great deal of hand-wringing about the admissibility of substitution in Chapter 6. In an ideal world, these issues would have been cleanly solved by our logical framework, but LFs that provide such guarantees [HB21; Uem21] do not incorporate modal type theory.

Part of the reason for this oversight is the comparative lack of experience with modal type theories. When different modal type theories used such disparate assumptions, it was difficult to settle on a particular logical framework (which would then almost certainly exclude some examples). If—as this thesis argues—wDRAs suffice, it becomes possible to propose logical frameworks for modal type theory.

Recently, Uemura [Uem23] has proposed such an LF based on *fibered* representable map categories. Prior work by Uemura has argued for realizing type theories as lex categories with a class of *representable* maps and his proposal extends this philosophy to multimodal type theories by considering fibrations over the mode theory $\mathcal{E} \longrightarrow \mathcal{M}$. The reindexing maps between fibers $\mathcal{E}_m \longrightarrow \mathcal{E}_n$ are the manifestation of modalities at the level of the logical framework—in much the same way that dependent products in $\mathcal{E}_m$ realize the hypothetical judgment.

This approach appears promising, but much remains to be done. For instance, it is unclear what structure (if any) the reindexing maps ought to preserve. Absent the requirement that reindexing functors preserve any structure, it is also open whether one can syntactically present such fibrations. Some concrete syntax appears necessary, if only for adequacy. Finally, it is unknown what structure one may reasonably postulate on $\mathcal{E} \longrightarrow \mathcal{M}$ while automatically maintaining the admissiblity of substitution. More provocatively, it remains unclear how one distinguishes between DRA and MTT in this setup, in spite of the fact that the former has a rather poor substitution lemma. We hope to see this topic explored further in the future.

Bibliography

[Aag+22] Frederik Lerbjerg Aagaard, Magnus Baunsgaard Kristensen, Daniel Gratzer, and Lars Birkedal. *Unifying cubical and multimodal type theory.* 2022. arXiv: 2203.13000 [cs.LO].

[Abe13] Andreas Abel. "Normalization by Evaluation: Dependent Types and Impredicativity". Habilitation. Ludwig-Maximilians-Universität München, 2013.

[AB20] Andreas Abel and Jean-Philippe Bernardy. "A Unified View of Modalities in Type Systems". In: *Proc. ACM Program. Lang.* 4.ICFP (Aug. 2020). DOI: 10.1145/3408972.

[ADE23] Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. *A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized.* To appear in the proceedings of the International Conference on Functional Programming 2023 (ICFP23). 2023. URL: https://www.cse.chalmers.se/~nad/publications/abel-danielsson-eriksson-graded-type-theory.pdf.

[AR94] Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories.* London Mathematical Society Lecture Note Series. Cambridge University Press, 1994. DOI: 10.1017/CBO9780511600579.

[Ahm04] Amal Jamil Ahmed. "Semantics of Types for Mutable State". AAI3136691. PhD thesis. USA: Princeton University, 2004.

[All87] Stuart Frazier Allen. "A non-type-theoretic semantics for type-theoretic language". PhD thesis. Ithaca, NY, USA: Cornell University, 1987.

[AHS95] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. "Categorical reconstruction of a reduction free normalization proof". In: *Category Theory and Computer Science.* Ed. by David Pitt, David E. Rydeheard, and Peter Johnstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 182–199. ISBN: 978-3-540-44661-3.

[AK16] Thorsten Altenkirch and Ambrus Kaposi. "Normalisation by Evaluation for Dependent Types". In: *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).* Ed. by Delia Kesner and Brigitte Pientka. Vol. 52. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 6:1–6:16. ISBN: 978-3-95977-010-1. DOI: 10.4230/LIPIcs.FSCD.2016.6. URL: http://drops.dagstuhl.de/opus/volltexte/2016/5972.

[AR89] Pierre America and Jan Rutten. “Solving reflexive domain equations in a category of complete metric spaces”. In: *Journal of Computer and System Sciences* 39.3 (1989), pp. 343–375. ISSN: 0022-0000. DOI: 10.1016/0022-0000(89)90027-5.

[AM01] Andrew W. Appel and David McAllester. “An indexed model of recursive types for foundational proof-carrying code”. In: *ACM Transactions on Programming Languages and Systems* 23.5 (2001), pp. 657–683. DOI: 10.1145/504709.504712.

[AGV72] Michael Artin, Alexander Grothendieck, and Jean-Louis Verdier. *Théorie des topos et cohomologie étale des schémas.* Séminaire de Géométrie Algébrique du Bois-Marie 1963–1964 (SGA 4), Dirigé par M. Artin, A. Grothendieck, et J.-L. Verdier. Avec la collaboration de N. Bourbaki, P. Deligne et B. Saint-Donat, Lecture Notes in Mathematics, Vol. 269, 270, 305. Berlin: Springer-Verlag, 1972.

[Atk18] Robert Atkey. “Syntax and Semantics of Quantitative Type Theory”. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science.* LICS ’18. Oxford, United Kingdom: Association for Computing Machinery, 2018, pp. 56–65. ISBN: 9781450355834. DOI: 10.1145/3209108.3209189.

[AM13] Robert Atkey and Conor McBride. “Productive Coprogramming with Guarded Recursion”. In: *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming.* ICFP ’13. Association for Computing Machinery, 2013, pp. 197–208. DOI: 10.1145/2500365.2500597. URL: https://doi.org/10.1145/2500365.2500597.

[Awo10] S. Awodey. *Category Theory.* Oxford Logic Guides. OUP Oxford, 2010. ISBN: 9780199587360.

[Awo18] Steve Awodey. “Natural models of homotopy type theory”. In: *Mathematical Structures in Computer Science* 28.2 (2018), pp. 241–286. ISSN: 09601295. DOI: 10.1017/S0960129516000268. eprint: 1406.3219.

[Awo22] Steve Awodey. *On Hofmann–Streicher universes.* 2022. arXiv: 2205.10917 [math.CT].

[BGM17] Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. “The clocks are ticking: No more delays!” In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS).* IEEE, 2017. DOI: 10.1109/LICS.2017.8005097. URL: http://www.itu.dk/people/mogel/papers/lics2017.pdf.

[BGM19] Patrick Bahr, Christian Uldal Graulund, and Rasmus Ejlers Møgelberg. “Simply RaTT: A Fitch-style Modal Calculus for Reactive Programming Without Space Leaks”. In: *Proc. ACM Program. Lang.* 3 (ICFP 2019), 109:1–109:27. ISSN: 2475-1421. DOI: 10.1145/3341713.

[BGM21] Patrick Bahr, Christian Uldal Graulund, and Rasmus Ejlers Møgelberg. “Diamonds Are Not Forever: Liveness in Reactive Programming with Guarded Recursion”. In: *Proc. ACM Program. Lang.* 5.POPL (Jan. 2021). DOI: 10.1145/3434283. URL: https://doi.org/10.1145/3434283.

[BL13] Andrej Bauer and Peter LeFanu Lumsdaine. “On the Bourbaki–Witt principle in toposes”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 155.1 (2013), pp. 87–99. DOI: 10.1017/S0305004113000108.

[BS18] Ulrich Berger and Anton Setzer. “Undecidability of Equality for Codata Types”. In: *Coalgebraic Methods in Computer Science.* Ed. by Corina Cîrstea. Cham: Springer International Publishing, 2018, pp. 34–55. ISBN: 978-3-030-00389-0.

[Bd00] G. M. Bierman and V. C. V. de Paiva. “On an Intuitionistic Modal Logic”. In: *Studia Logica* 65.3 (2000). DOI: 10.1023/A:1005291931660.

[Bir00] Lars Birkedal. “Developing Theories of Types and Computability via Realizability”. In: *Electronic Notes in Theoretical Computer Science* 34 (2000).

[Bir+19] Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. “Guarded Cubical Type Theory”. In: *Journal of Automated Reasoning* 63 (2019), pp. 211–253.

[Bir+20] Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. “Modal dependent type theory and dependent right adjoints”. In: *Mathematical Structures in Computer Science* 30.2 (2020), pp. 118–138. DOI: 10.1017/S0960129519000197. eprint: 1804.05236.

[Bir+12] Lars Birkedal, Rasmus Møgelberg, Jan Schwinghammer, and Kristian Støvring. “First steps in synthetic guarded domain theory: step-indexing in the topos of trees”. In: *Logical Methods in Computer Science* 8.4 (2012). Ed. by Patrick Baillot. DOI: 10.2168/LMCS-8(4:1)2012.

[BM13] Lars Birkedal and Rasmus Ejlers Møgelberg. “Intensional Type Theory with Guarded Recursive Types qua Fixed Points on Universes”. In: *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science.* LICS ’13. USA: IEEE Computer Society, 2013, pp. 213–222. ISBN: 9780769550206. DOI: 10.1109/LICS.2013.27.

[BBM14] Ales Bizjak, Lars Birkedal, and Marino Miculan. “A Model of Countable Nondeterminism in Guarded Type Theory”. In: *Rewriting and Typed Lambda Calculi – Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings.* Ed. by Gilles Dowek. Vol. 8560. Lecture Notes in Computer Science. Springer, 2014, pp. 108–123. ISBN: 978-3-319-08917-1. DOI: 10.1007/978-3-319-08918-8_8.

[BB22] Aleš Bizjak and Lars Birkedal. *Lecture Notes on Iris: Higher-Order Concurrent Separation Logic.* Online. https://iris-project.org/tutorial-pdfs/iris-lecture-notes.pdf. 2022.

[Biz+16] Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus E. Møgelberg, and Lars Birkedal. “Guarded Dependent Type Theory with Coinductive Types”. In: *Foundations of Software Science and Computation Structures.* Ed. by Bart Jacobs and Christof Löding. Springer Berlin Heidelberg, 2016, pp. 20–35.

[Biz+19] Aleš Bizjak, Daniel Gratzer, Robbert Krebbers, and Lars Birkedal. “Iron: Managing Obligations in Higher-order Concurrent Separation Logic”. In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019), 65:1–65:30. ISSN: 2475-1421. DOI: 10.1145/3290378. URL: http://doi.acm.org/10.1145/3290378.

[BM15] Aleš Bizjak and Rasmus Ejlers Møgelberg. “A Model of Guarded Recursion With Clock Synchronisation”. In: *Electronic Notes in Theoretical Computer Science* 319 (2015). The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI)., pp. 83–101. DOI: https://doi.org/10.1016/j.entcs.2015.12.007. URL: http://www.sciencedirect.com/science/article/pii/S1571066115000742.

[BM20] Aleš Bizjak and Rasmus Ejlers Møgelberg. “Denotational semantics for guarded dependent type theory”. In: *Mathematical Structures in Computer Science* 30.4 (2020), pp. 342–378. DOI: 10.1017/S0960129520000080.

[Ble17] Ingo Blechschmidt. “Using the internal language of toposes in algebraic geometry”. PhD thesis. Universität Augsburg, 2017.

[BCW] Ingo Blechschmidt, Felix Cherubini, and David Wärn. *Čech Cohomology in Homotopy Type Theory.* To appear.

[BKS21] Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. *Relative induction principles for type theories.* 2021. DOI: 10.48550/ARXIV.2102.11649.

[BKS23] Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. *For the Metatheory of Type Theory, Internal Sconing Is Enough.* 2023. arXiv: 2302.05190 [cs.LO].

[Bor94] Francis Borceux. *Handbook of Categorical Algebra.* Vol. 1. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994. DOI: 10.1017/CBO9780511525858.

[BH01] Sylvain Boulmé and Grégoire Hamon. “Certifying Synchrony for Free”. In: *Logic for Programming, Artificial Intelligence, and Reasoning.* Ed. by Robert Nieuwenhuis and Andrei Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 495–506. ISBN: 978-3-540-45653-7.

[CJ95] Aurelio Carboni and Peter Johnstone. “Connected limits, familial representability and Artin glueing”. In: *Mathematical Structures in Computer Science* 5.4 (1995), pp. 441–459. DOI: 10.1017/S0960129500001183.

[Car78] John Cartmell. “Generalised Algebraic Theories and Contextual Categories”. PhD thesis. University of Oxford, 1978.

[CCD20] Simon Castellan, Pierre Clairambault, and Peter Dybjer. *Categories with Families: Unityped, Simply Typed, and Dependently Typed.* 2020. arXiv: 1904.00827 [cs.LO].

[Cav21] Evan Cavallo. “Higher inductive types and internal parametricity for cubical type theory”. PhD thesis. Carnegie Mellon Univesity, 2021.

[CP02] Iliano Cervesato and Frank Pfenning. “A Linear Logical Framework”. In: *Information and Computation* 179.1 (2002), pp. 19–75. ISSN: 0890-5401. DOI: 10.1006/inco.2001.2951.

[CCH23] Felix Cherubini, Thierry Coquand, and Matthias Hutzler. *A Foundation for Synthetic Algebraic Geometry*. 2023. arXiv: 2307.00073 [math.AG].

[Cis19] Denis-Charles Cisinski. *Higher Categories and Homotopical Algebra*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2019. DOI: 10.1017/9781108588737. URL: http://www.mathematik.uni-regensburg.de/cisinski/CatLR.pdf.

[CD14] Pierre Clairambault and Peter Dybjer. "The biequivalence of locally cartesian closed categories and Martin-Löf type theories". In: *Mathematical Structures in Computer Science* 24.6 (2014). DOI: 10.1017/S0960129513000881.

[Clo18] Ranald Clouston. "Fitch-Style Modal Lambda Calculi". In: *Foundations of Software Science and Computation Structures*. Ed. by Christel Baier and Ugo Dal Lago. Springer International Publishing, 2018, pp. 258–275.

[Clo+15] Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. "Programming and Reasoning with Guarded Recursion for Coinductive Types". In: *Foundations of Software Science and Computation Structures*. Ed. by Andrew Pitts. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 407–421. ISBN: 978-3-662-46678-0.

[Coh+17] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical Type Theory: a constructive interpretation of the univalence axiom". In: *IfCoLog Journal of Logics and their Applications* 4.10 (2017), pp. 3127–3169. arXiv: 1611.02108 [cs.LO].

[Coq19] Thierry Coquand. "Canonicity and normalization for dependent type theory". In: *Theoretical Computer Science* 777 (2019), pp. 184–191. DOI: 10.1016/j.tcs.2019.01.015.

[CMR17] Thierry Coquand, Bassel Mannaa, and Fabien Ruch. "Stack semantics of type theory". In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. June 2017, pp. 1–11. DOI: 10.1109/LICS.2017.8005130.

[DP01] Rowan Davies and Frank Pfenning. "A Modal Analysis of Staged Computation". In: *Journal of the ACM* 48.3 (May 2001), pp. 555–604.

[Day70] Brian Day. "On closed categories of functors". In: *Reports of the Midwest Category Seminar IV*. Ed. by S. MacLane, H. Applegate, M. Barr, B. Day, E. Dubuc, Phreilambud, A. Pultr, R. Street, M. Tierney, and S. Swierczkowski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1970, pp. 1–38. ISBN: 978-3-540-36292-0.

[dR15] Valeria de Paiva and Eike Ritter. "Fibrational Modal Type Theory". In: *Proceedings of the Tenth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2015)*. 2015. DOI: 10.1016/j.entcs.2016.06.010.

[DAB11] Derek Dreyer, Amal Ahmed, and Lars Birkedal. "Logical Step-Indexed Logical Relations". In: *Logical Methods in Computer Science* Volume 7, Issue 2 (June 2011). DOI: 10.2168/lmcs-7(2:16)2011.

[DNB12] Derek Dreyer, Georg Neis, and Lars Birkedal. "The impact of higher-order state and control effects on local relational reasoning". In: *Journal of Functional Programming* 22.4-5 (2012), pp. 477–528. DOI: 10.1017/S095679681200024X.

[Dyb96] Peter Dybjer. "Internal type theory". In: *Types for Proofs and Programs*. Ed. by Stefano Berardi and Mario Coppo. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 120–134. DOI: 10.1007/3-540-61780-9_66.

[Fio02] Marcelo Fiore. "Semantic Analysis of Normalisation by Evaluation for Typed Lambda Calculus". In: *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*. PPDP '02. Pittsburgh, PA, USA: ACM, 2002, pp. 26–37. ISBN: 1-58113-528-9. DOI: 10.1145/571157.571161.

[Fio12] Marcelo Fiore. *Discrete generalised polynomial functors*. Slides from talk given at ICALP 2012. 2012. URL: https://www.cl.cam.ac.uk/~mpf23/talks/ICALP2012.pdf.

[Fre78] Peter Freyd. "On proving that **1** is an indecomposable projective in various free categories". 1978.

[GK13] Nicola Gambino and Joachim Kock. "Polynomial functors and polynomial monads". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 154.1 (2013), pp. 153–192. DOI: 10.1017/S0305004112000394.

[Gir11] Jean-Yves Girard. *The Blind Spot: Lectures on Logic*. European Mathematical Society, 2011.

[Goo+21] Sam van Gool, Adrien Guatto, George Metcalfe, and Simon Santschi. "Time Warps, from Algebra to Algorithms". In: *Relational and Algebraic Methods in Computer Science: 19th International Conference, RAMiCS 2021, Marseille, France, November 2–5, 2021, Proceedings*. Marseille, France: Springer-Verlag, 2021, pp. 309–324. ISBN: 978-3-030-88700-1. DOI: 10.1007/978-3-030-88701-8_19.

[Gra22] Daniel Gratzer. "Normalization for Multimodal Type Theory". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '22. Haifa, Israel: Association for Computing Machinery, 2022. ISBN: 9781450393515. DOI: 10.1145/3531130.3532398. URL: https://doi.org/10.1145/3531130.3532398.

[Gra23] Daniel Gratzer. *Normalization for multimodal type theory*. 2023. arXiv: 2301.11842 [cs.LO].

[GB22] Daniel Gratzer and Lars Birkedal. "A Stratified Approach to Löb Induction". In: *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*. Ed. by Amy Felty. Vol. 228. Leibniz International Proceedings in Informatics (LIPIcs). Saarbrücken, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2022. DOI: 10.4230/LIPIcs.FSCD.2022.3. URL: https://jozefg.github.io/papers/a-stratified-approach-to-lob-induction.pdf.

[Gra+22] Daniel Gratzer, Evan Cavallo, G. A. Kavvos, Adrien Guatto, and Lars Birkedal. "Modalities and Parametric Adjoints". In: *ACM Trans. Comput. Logic* 23.3 (Apr. 2022). ISSN: 1529-3785. DOI: 10.1145/3514241. URL: https://doi.org/10.1145/3514241.

[Gra+21] 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). DOI: 10.46298/lmcs-17(3:11)2021. URL: https://lmcs.episciences.org/7713.

[Gra+20a] Daniel Gratzer, G.A. Kavvos, Andreas Nuyts, and Lars Birkedal. "Multimodal Dependent Type Theory". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. ACM, 2020. DOI: 10.1145/3373718.3394736.

[Gra+20b] Daniel Gratzer, G.A. Kavvos, Andreas Nuyts, and Lars Birkedal. *Type Theory à la Mode*. Technical Report for the LICS paper "Multimodal Dependent Type Theory". 2020. URL: https://jozefg.github.io/papers/type-theory-a-la-mode.pdf.

[GSS22] Daniel Gratzer, Michael Shulman, and Jonathan Sterling. *Strict universes for Grothendieck topoi*. 2022. arXiv: 2202.12012 [math.CT].

[GS20] Daniel Gratzer and Jonathan Sterling. *Syntactic categories for dependent type theory: sketching and adequacy*. 2020. arXiv: 2012.10783 [cs.LO].

[Gra+23] Daniel Gratzer, Jonathan Sterling, Carlo Angiuli, Thierry Coquand, and Lars Birkedal. *Controlling unfolding in type theory*. 2023.

[GSB19a] Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. "Implementing a Modal Dependent Type Theory". In: *Proc. ACM Program. Lang.* 3 (ICFP 2019). DOI: 10.1145/3341711.

[GSB19b] Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. *Normalization-by-Evaluation for Modal Dependent Type Theory*. Technical Report for the ICFP paper by the same name. 2019. URL: https://jozefg.github.io/papers/2019-implementing-modal-dependent-type-theory-tech-report.pdf.

[Gro+17] Jacob A Gross, Daniel R Licata, Max S New, Jennifer Paykin, Mitchell Riley, Michael Shulman, and Felix Wellen. "Differential Cohesive Type Theory (Extended Abstract)". In: *Extended abstracts for the Workshop "Homotopy Type Theory and Univalent Foundations"*. 2017. URL: https://hott-uf.github.io/2017/abstracts/cohesivett.pdf.

[Gua18] Adrien Guatto. "A Generalized Modality for Recursion". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '18. ACM, 2018. DOI: 10.1145/3209108.3209148.

[Hal+91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. "The synchronous data flow programming language LUSTRE". In: *Proceedings of the IEEE* 79.9 (1991), pp. 1305–1320. DOI: 10.1109/5.97300.

[HB21] Philipp G. Haselwarter and Andrej Bauer. *Finitary type theories with and without contexts*. 2021. arXiv: 2112.00539 [math.LO].

[Hof99] Martin Hofmann. “Semantical Analysis of Higher-Order Abstract Syntax”. In: *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*. LICS ’99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 204–. ISBN: 0-7695-0158-3. URL: http://dl.acm.org/citation.cfm?id=788021.788940.

[HS97] Martin Hofmann and Thomas Streicher. “Lifting Grothendieck Universes”. Unpublished note. 1997. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf.

[HP23] Jason Z. S. Hu and Brigitte Pientka. “A Categorical Normalization Proof for the Modal Lambda-Calculus”. In: *Electronic Notes in Theoretical Informatics and Computer Science* Volume 1 - Proceedings of MFPS XXXVIII (Feb. 2023). DOI: 10.46298/entics.10360.

[HPS22] Jason Z. S. Hu, Brigitte Pientka, and Ulrich Schöpp. “A Category Theoretic View of Contextual Types: From Simple Types to Dependent Types”. In: *ACM Trans. Comput. Logic* 23.4 (Oct. 2022). ISSN: 1529-3785. DOI: 10.1145/3545115. URL: https://doi.org/10.1145/3545115.

[Hyl82] J.M.E. Hyland. “The Effective Topos”. In: *The L. E. J. Brouwer Centenary Symposium*. Ed. by A.S. Troelstra and D. van Dalen. Vol. 110. Studies in Logic and the Foundations of Mathematics. Elsevier, 1982, pp. 165–216. DOI: 10.1016/S0049-237X(09)70129-6.

[Jan+22] Junyoung Jang, Samuel Gélineau, Stefan Monnier, and Brigitte Pientka. “Mundefinedbius: Metaprogramming Using Contextual Types: The Stage Where System f Can Pattern Match on Itself”. In: *Proc. ACM Program. Lang.* 6.POPL (Jan. 2022). DOI: 10.1145/3498700.

[Jef12] Alan Jeffrey. “LTL Types FRP: Linear-Time Temporal Logic Propositions as Types, Proofs as Functional Reactive Programs”. In: *Proceedings of the Sixth Workshop on Programming Languages Meets Program Verification*. PLPV ’12. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 2012, pp. 49–60. ISBN: 9781450311250. DOI: 10.1145/2103776.2103783.

[Jef14] Alan Jeffrey. “Functional Reactive Types”. In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. CSL-LICS ’14. Vienna, Austria: Association for Computing Machinery, 2014. ISBN: 9781450328869. DOI: 10.1145/2603088.2603106. URL: https://doi.org/10.1145/2603088.2603106.

[Jel13] Wolfgang Jeltsch. “Temporal Logic with ”Until”, Functional Reactive Programming with Processes, and Concrete Process Categories”. In: *Proceedings of the 7th Workshop on Programming Languages Meets Program Verification*. PLPV ’13. Rome, Italy: Association for Computing Machinery, 2013, pp. 69–78. ISBN: 9781450318600. DOI: 10.1145/2428116.2428128.

[Joh02] P.T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Oxford Logic Guides. Clarendon Press, 2002. ISBN: 9780198515982.

[Jun+16] Ralf Jung, Robbert Krebbers, Lars Birkedal, and Derek Dreyer. “Higher-Order Ghost State”. In: *SIGPLAN Not.* 51.9 (Sept. 2016), pp. 256–269. ISSN: 0362-1340. DOI: 10.1145/3022670.2951943.

[Jun+18] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. “Iris from the ground up: A modular foundation for higher-order concurrent separation logic”. In: *Journal of Functional Programming* 28 (2018). DOI: 10.1017/S0956796818000151.

[Jun+15] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. “Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning”. In: *SIGPLAN Not.* 50.1 (Jan. 2015), pp. 637–650. ISSN: 0362-1340. DOI: 10.1145/2775051.2676980.

[KHS19] Ambrus Kaposi, Simon Huber, and Christian Sattler. “Gluing for type theory”. In: *Proceedings of the 4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman Geuvers. Vol. 131. 2019.

[KKA19] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. “Constructing Quotient Inductive-inductive Types”. In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019), 2:1–2:24. ISSN: 2475-1421. DOI: 10.1145/3290315.

[KL21] Krzysztof Kapulkin and Peter LeFanu Lumsdaine. “The simplicial model of Univalent Foundations (after Voevodsky)”. In: *Journal of the European Mathematical Society* 6 (2021), pp. 2071–2126. DOI: 10.4171/JEMS/1050.

[Kav17] G. A. Kavvos. “Dual-context calculi for modal logic”. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2017, pp. 1–12. DOI: 10.1109/LICS.2017.8005089. arXiv: 1602.04860.

[Kav19] G. A. Kavvos. “Modalities, Cohesion, and Information Flow”. In: *Proceedings of the ACM on Programming Languages* 3 (POPL 2019), 20:1–20:29. DOI: 10.1145/3290333.

[KG23] G. A. Kavvos and Daniel Gratzer. “Under Lock and Key: A Proof System for a Multimodal Logic”. In: *Bulletin of Symbolic Logic* (2023), pp. 1–30. DOI: 10.1017/bsl.2023.14.

[KI19] Akira Kawata and Atsushi Igarashi. “A Dependently Typed Multi-stage Calculus”. In: *Programming Languages and Systems*. Ed. by Anthony Widjaja Lin. Cham: Springer International Publishing, 2019, pp. 53–72. ISBN: 978-3-030-34175-6.

[KPT99] Yoshiki Kinoshita, John Power, and Makoto Takeyama. “Sketches”. In: *Journal of Pure and Applied Algebra* 143.1 (1999), pp. 275–291. ISSN: 0022-4049. DOI: 10.1016/S0022-4049(98)00114-5.

[Kle50] S. C. Kleene. “A symmetric form of Gödel’s theorem”. In: *Ind. Math* (1950), 12:244–246. DOI: 10.2307/2266709.

[Kov22] András Kovács. “Staged Compilation with Two-Level Type Theory”. In: *Proc. ACM Program. Lang.* 6.ICFP (Aug. 2022). DOI: 10.1145/3547641.

[KB11] Neelakantan Krishnaswami and Nick Benton. "Ultrametric Semantics of Reactive Programs". In: *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society, June 2011.

[Kri13] Neelakantan R. Krishnaswami. "Higher-Order Functional Reactive Programming without Spacetime Leaks". In: *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming.* ICFP '13. Boston, Massachusetts, USA: Association for Computing Machinery, 2013, pp. 221–232. ISBN: 9781450323260. DOI: 10.1145/2500365.2500588. URL: https://doi.org/10.1145/2500365.2500588.

[KBH12] Neelakantan R. Krishnaswami, Nick Benton, and Jan Hoffmann. "Higher-Order Functional Reactive Programming in Bounded Space". In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* POPL '12. Philadelphia, PA, USA: Association for Computing Machinery, 2012, pp. 45–58. ISBN: 9781450310833. DOI: 10.1145/2103656.2103665.

[KPB15] Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton. "Integrating Linear and Dependent Types". In: *SIGPLAN Not.* 50.1 (Jan. 2015), pp. 17–30. ISSN: 0362-1340. DOI: 10.1145/2775051.2676969.

[KMV22] Magnus Baunsgaard Kristensen, Rasmus Ejlers Møgelberg, and Andrea Vezzosi. "Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science.* New York, NY, USA: Association for Computing Machinery, 2022. DOI: 10.1145/3531130.3533359.

[Lac09] Stephen Lack. "A 2-Categories Companion". In: *The IMA Volumes in Mathematics and its Applications* (Sept. 2009), pp. 105–191. ISSN: 0940-6573. DOI: 10.1007/978-1-4419-1524-5_4.

[LS88] Joachim Lambek and Philip J. Scott. *Introduction to Higher-Order Categorical Logic.* Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1988. ISBN: 9780521356534.

[Law07] F. William Lawvere. "Axiomatic cohesion". In: *Theory and Applications of Categories* 19.3 (2007), pp. 41–49. URL: http://www.tac.mta.ca/tac/volumes/19/3/19-03.pdf.

[Lic+18] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. "Internal Universes in Models of Homotopy Type Theory". In: *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).* Ed. by H. Kirchner. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018, 22:1–22:17. DOI: 10.4230/LIPIcs.FSCD.2018.22. eprint: 1801.07664.

[LS16] Daniel R. Licata and Michael Shulman. "Adjoint Logic with a 2-Category of Modes". In: *Logical Foundations of Computer Science.* Ed. by Sergei Artemov and Anil Nerode. Springer International Publishing, 2016, pp. 219–235. DOI: 10.1007/978-3-319-27683-0_16.

[LSR17] Daniel R. Licata, Michael Shulman, and Mitchell Riley. “A Fibrational Framework for Substructural and Modal Logics”. In: *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*. Ed. by Dale Miller. Vol. 84. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 25:1–25:22. DOI: 10.4230/LIPIcs.FSCD.2017.25.

[LW15] Peter Lefanu Lumsdaine and Michael A. Warren. “The local universes model, an overlooked coherence construction for dependent type theories”. In: *ACM Transactions on Computational Logic* 16.3 (2015). DOI: 10.1145/2754931.

[Lur09] Jacob Lurie. *Higher Topos Theory*. Princeton University Press, 2009. ISBN: 9780691140490.

[Lur22] Jacob Lurie. *Kerodon*. https://kerodon.net. 2022.

[MM92] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in geometry and logic : a first introduction to topos theory*. Universitext. New York: Springer, 1992. ISBN: 0-387-97710-4.

[MMV20] Bassel Mannaa, Rasmus Ejlers Møgelberg, and Niccolò Veltri. “Ticking clocks as dependent right adjoints: Denotational semantics for clocked type theory”. In: *Logical Methods in Computer Science* Volume 16, Issue 4 (Dec. 2020). DOI: 10.23638/LMCS-16(4:17)2020.

[Mar82] Per Martin-Löf. “Constructive Mathematics and Computer Programming”. In: *Logic, Methodology and Philosophy of Science VI*. Ed. by L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski. Vol. 104. Studies in Logic and the Foundations of Mathematics. Elsevier, 1982, pp. 153–175. DOI: 10.1016/S0049-237X(09)70189-2.

[Mar92] Per Martin-Löf. *Substitution calculus*. Notes from a lecture given in Göteborg. 1992.

[May01] J.P. May. “Picard Groups, Grothendieck Rings, and Burnside Rings of Categories”. In: *Advances in Mathematics* 163.1 (2001), pp. 1–16. ISSN: 0001-8708. DOI: 10.1006/aima.2001.1996.

[MP08] Conor McBride and Ross Paterson. “Applicative programming with effects”. In: *Journal of Functional Programming* 18.1 (2008). DOI: 10.1017/S0956796807006326. URL: http://www.staff.city.ac.uk/~ross/papers/Applicative.pdf.

[MS93] John C. Mitchell and Andre Scedrov. “Notes on sconing and relators”. In: *Computer Science Logic*. Ed. by E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. M. Richter. Springer Berlin Heidelberg, 1993, pp. 352–378. DOI: 10.1007/3-540-56992-8_21.

[Møg14] Rasmus Ejlers Møgelberg. “A Type Theory for Productive Coprogramming via Guarded Recursion”. In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. CSL-LICS ’14. 2014. DOI: 10.1145/2603088.2603132.

[MP16] Rasmus Ejlers Møgelberg and Marco Paviotti. “Denotational semantics of recursive types in synthetic guarded domain theory”. English. In: *LICS ’16 Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. United States: Association for Computing Machinery, 2016, pp. 317–326. ISBN: 978-1-4503-4391-6. DOI: 10.1145/2933575.2934516.

[MV19] Rasmus Ejlers Møgelberg and Niccolò Veltri. “Bisimulation as Path Type for Guarded Recursive Types”. In: *Proceedings of the ACM on Programming Languages* 3.POPL (Dec. 2019). DOI: 10.1145/3290317.

[MV21] Rasmus Ejlers Møgelberg and Andrea Vezzosi. “Two Guarded Recursive Powerdomains for Applicative Simulation”. In: *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics*. Vol. 351. Electronic Proceedings in Theoretical Computer Science, Dec. 2021, pp. 200–217. DOI: 10.4204/EPTCS.351.13.

[MEO21] Benjamin Moon, Harley Eades III, and Dominic Orchard. “Graded Modal Dependent Type Theory”. In: *Programming Languages and Systems*. Ed. by Nobuko Yoshida. Cham: Springer International Publishing, 2021, pp. 462–490. ISBN: 978-3-030-72019-3.

[Mur08] Tom Murphy VII. “Modal Types for Mobile Code”. Available as technical report CMU-CS-08-126. PhD thesis. Carnegie Mellon, Jan. 2008. URL: http://tom7.org/papers/.

[MCH05] Tom Murphy VII, Karl Crary, and Robert Harper. “Distributed Control Flow with Classical Modal Logic”. In: *Computer Science Logic, 19th International Workshop (CSL 2005)*. Ed. by Luke Ong. Lecture Notes in Computer Science. Oxford, UK: Springer, Aug. 2005.

[MCH07] Tom Murphy VII, Karl Crary, and Robert Harper. “Type-safe Distributed Programming with ML5”. In: *Trustworthy Global Computing 2007*. Sophia-Antipolis, France, Nov. 2007.

[Mur+04] Tom Murphy VII, Karl Crary, Robert Harper, and Frank Pfenning. “A Symmetric Modal Lambda Calculus for Distributed Computing”. In: *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*. Turku, Finland: IEEE Press, July 2004.

[MR23] David Jaz Myers and Mitchell Riley. *Commuting Cohesions*. 2023. arXiv: 2301.13780 [math.CT].

[Nak00] H. Nakano. “A modality for recursion”. In: *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*. IEEE Computer Society, 2000, pp. 255–266.

[NP05] Aleksandar Nanevski and Frank Pfenning. “Staged computation with names and necessity”. In: *Journal of Functional Programming* 15.6 (2005), pp. 893–939. DOI: 10.1017/S095679680500568X.

[NPP08] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. “Contextual Modal Type Theory”. In: *ACM Trans. Comput. Logic* 9.3 (June 2008). ISSN: 1529-3785. DOI: 10.1145/1352582.1352591.

[NU22] Hoang Kim Nguyen and Taichi Uemura. *∞-type theories*. 2022. arXiv: 2205.00798 [math.CT].

[Niu+22] Yue Niu, Jonathan Sterling, Harrison Grodin, and Robert Harper. "A Cost-Aware Logical Framework". In: *Proceedings of the ACM on Programming Languages* 6.POPL (Jan. 2022). DOI: 10.1145/3498670. arXiv: 2107.04663 [cs.PL].

[NPS90] Bengt Nordström, Kent Peterson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory*. Vol. 7. International Series of Monographs on Computer Science. NY: Oxford University Press, 1990.

[Nuy20] Andreas Nuyts. "Contributions to Multimode and Presheaf Type Theory". PhD thesis. KU Leuven, 2020.

[Nuy23] Andreas Nuyts. "A Lock Calculus for Multimode Type Theory". In: *28th International Conference on Types for Proofs and Programs (TYPES 2023)*. 2023.

[ND18] Andreas Nuyts and Dominique Devriese. "Degrees of Relatedness: A Unified Framework for Parametricity, Irrelevance, Ad-Hoc Polymorphism, Intersections, Unions and Algebra in Dependent Type Theory". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '18. ACM, 2018. DOI: 10.1145/3209108.3209119.

[ND21] Andreas Nuyts and Dominique Devriese. *Transpension: The Right Adjoint to the Pi-type*. 2021. URL: https://arxiv.org/abs/2008.08533.

[OP18] Ian Orton and Andrew M. Pitts. "Axioms for Modelling Cubical Type Theory in a Topos". In: *Logical Methods in Computer Science* 14.4 (2018). DOI: 10.23638/LMCS-14(4:23)2018. arXiv: 1712.04864.

[PS23] Daniele Palombi and Jonathan Sterling. "Classifying topoi in synthetic guarded domain theory". In: *Proceedings 38th Conference on Mathematical Foundations of Programming Semantics, MFPS 2022*. Feb. 2023. DOI: 10.46298/entics.10323.

[Pal93] Jens Palsberg. "Correctness of binding-time analysis". In: *Journal of Functional Programming* 3.3 (1993), pp. 347–363. DOI: 10.1017/S0956796800000770.

[PMB15] Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. "A Model of PCF in Guarded Type Theory". In: *Electronic Notes in Theoretical Computer Science* 319.Supplement C (2015). The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI), pp. 333–349. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2015.12.020.

[PD01] Frank Pfenning and Rowan Davies. "A Judgmental Reconstruction of Modal Logic". In: *Mathematical Structures in Computer Science* 11.4 (2001), pp. 511–540. DOI: 10.1017/S0960129501003322. URL: http://www.cs.cmu.edu/~fp/papers/mscs00.pdf.

[Rie16] Emily Riehl. *Category Theory in Context*. Aurora: Dover Modern Math Originals. Dover Publications, 2016. ISBN: 9780486809038.

[RS17] Emily Riehl and Michael Shulman. "A type theory for synthetic ∞-categories". In: *Higher Structures* 1 (1 2017), pp. 147–224. DOI: 10.21136/HS.2017.06.

[RSS20] Egbert Rijke, Michael Shulman, and Bas Spitters. “Modalities in homotopy type theory”. In: *Logical Methods in Computer Science* 16.1 (2020). eprint: 1706.07526.

[Ril22] Mitchell Riley. “A Bunched Homotopy Type Theory for Synthetic Stable Homotopy Theory”. PhD thesis. Wesleyan University, 2022. DOI: 10.14418/wes01.3.139.

[Ros36] Barkley Rosser. “Extensions of Some Theorems of Gödel and Church”. In: *The Journal of Symbolic Logic* 1.3 (1936), pp. 87–91. ISSN: 00224812. URL: http://www.jstor.org/stable/2269028 (visited on 08/08/2022).

[SS20] Hisham Sati and Urs Schreiber. *Proper Orbifold Cohomology*. 2020. arXiv: 2008.01101 [math.AT].

[SS86] Stephen Schanuel and Ross Street. “The free adjunction”. en. In: *Cahiers de Topologie et Géométrie Différentielle Catégoriques* 27.1 (1986), pp. 81–83. URL: http://www.numdam.org/item/CTGDC_1986__27_1_81_0.

[Sch13] Urs Schreiber. *Differential cohomology in a cohesive infinity-topos*. 2013. arXiv: 1310.7930 [math-ph].

[SS12] Urs Schreiber and Michael Shulman. “Quantum Gauge Field Theory in Cohesive Homotopy Type Theory”. In: *Proceedings 9th Workshop on Quantum Physics and Logic, QPL 2012, Brussels, Belgium, 10-12 October 2012*. Ed. by Ross Duncan and Prakash Panangaden. Vol. 158. EPTCS. 2012, pp. 109–126. DOI: 10.4204/EPTCS.158.8. URL: https://doi.org/10.4204/EPTCS.158.8.

[Sco70] Dana Scott. “Advice on Modal Logic”. In: *Philosophical Problems in Logic: Some Recent Developments*. Ed. by Karel Lambert. Dordrecht: Springer Netherlands, 1970, pp. 143–173. ISBN: 978-94-010-3272-8. DOI: 10.1007/978-94-010-3272-8_7.

[Shu15a] Michael Shulman. “The Univalence Axiom for Elegant Reedy Presheaves”. In: *Homology, Homotopy and Applications* 17 (2 2015), pp. 81–106.

[Shu15b] Michael Shulman. “Univalence for inverse diagrams and homotopy canonicity”. In: *Mathematical Structures in Computer Science* 25.5 (2015), pp. 1203–1277. DOI: 10.1017/S0960129514000565. eprint: 1203.3253.

[Shu18] Michael Shulman. “Brouwer’s fixed-point theorem in real-cohesive homotopy type theory”. In: *Mathematical Structures in Computer Science* 28.6 (2018), pp. 856–941. DOI: 10.1017/S0960129517000147. URL: https://doi.org/10.1017/S0960129517000147.

[Shu19] Michael Shulman. *All* $(\infty, 1)$*-toposes have strict univalent universes*. 2019. arXiv: 1904.07004 [math.AT].

[Shu23] Michael Shulman. *Semantics of multimodal adjoint type theory*. 2023. URL: https://arxiv.org/abs/2303.02572.

[SP82] M. B. Smyth and G. D. Plotkin. “The Category-Theoretic Solution of Recursive Domain Equations”. In: *SIAM Journal on Computing* 11.4 (1982), pp. 761–783. DOI: 10.1137/0211062.

[Spi+21] Simon Spies, Lennard Gäher, Daniel Gratzer, Joseph Tassarotti, Robbert Krebbers, Derek Dreyer, and Lars Birkedal. "Transfinite Iris: Resolving an Existential Dilemma of Step-Indexed Separation Logic". In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 80–95. ISBN: 9781450383912. URL: https://doi.org/10.1145/3453483.3454031.

[SGB23a] Philipp Stassen, Daniel Gratzer, and Lars Birkedal. "{mitten}: A Flexible Multimodal Proof Assistant". In: *28th International Conference on Types for Proofs and Programs (TYPES 2022)*. Ed. by Delia Kesner and Pierre-Marie Pédrot. Vol. 269. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 6:1–6:23. ISBN: 978-3-95977-285-3. DOI: 10.4230/LIPIcs.TYPES.2022.6.

[Ste21] Jonathan Sterling. "First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory". CMU technical report CMU-CS-21-142. PhD thesis. Carnegie Mellon University, 2021. DOI: 10.5281/zenodo.5709838.

[Ste22] Jonathan Sterling. "Naïve logical relations in synthetic Tait computability". Unpublished manuscript. June 2022.

[SA21] Jonathan Sterling and Carlo Angiuli. "Normalization for Cubical Type Theory". In: *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '21. New York, NY, USA: ACM, 2021.

[SAG19] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. "Cubical Syntax for Reflection-Free Extensional Equality". In: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman Geuvers. Vol. 131. Leibniz International Proceedings in Informatics (LIPIcs). 2019, 31:1–31:25. ISBN: 978-3-95977-107-8. DOI: 10.4230/LIPIcs.FSCD.2019.31. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10538.

[SAG22] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. "A Cubical Language for Bishop Sets". In: *Logical Methods in Computer Science* Volume 18, Issue 1 (Mar. 2022). DOI: 10.46298/lmcs-18(1:43)2022. URL: https://lmcs.episciences.org/9264.

[SGB23b] Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. *Denotational semantics of general store and polymorphism*. 2023. arXiv: 2210.02169 [cs.PL].

[SGB23c] Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. "Free theorems from univalent reference types: the impact of univalence on denotational semantics". 2023.

[SH21] Jonathan Sterling and Robert Harper. "Logical Relations as Types: Proof-Relevant Parametricity for Program Modules". In: *Journal of the ACM* 68.6 (2021). ISSN: 0004-5411. DOI: 10.1145/3474834. arXiv: 2010.08599 [cs.PL].

[SH22] Jonathan Sterling and Robert Harper. "Sheaf semantics of termination-insensitive noninterference". In: *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*. Ed. by Amy P. Felty. Vol. 228. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Aug. 2022, 5:1–5:19. ISBN: 978-3-95977-233-4. DOI: 10.4230/LIPIcs.FSCD.2022.5. arXiv: 2204.09421 [cs.PL].

[Str98] Thomas Streicher. "Categorical intuitions underlying semantic normalisation proofs". In: *Preliminary Proceedings of the APPSEM Workshop on Normalisation by Evaluation*. Ed. by O. Danvy and P. Dybjer. Department of Computer Science, Aarhus University, 1998.

[Str05] Thomas Streicher. "Universes in toposes". In: *From Sets and Types to Topology and Analysis: Towards practical foundations for constructive mathematics*. Ed. by Laura Crosilla and Peter Schuster. Vol. 48. Oxford Logical Guides. Oxford: Oxford University Press, 2005, pp. 78–90. ISBN: 978-0-19-856651-9. DOI: 10.1093/acprof:oso/9780198566519.001.0001.

[Tai67] W. W. Tait. "Intensional Interpretations of Functionals of Finite Type I". In: *Journal of Symbolic Logic* 32.2 (1967), pp. 198–212. DOI: 10.2307/2271658.

[Tra53] B. A. Trahtenbrot. "On Recursive Separability". In: *Dokl. Acad. Nauk* 88 (1953), pp. 953–956.

[TDB13] Aaron Turon, Derek Dreyer, and Lars Birkedal. "Unifying Refinement and Hoare-Style Reasoning in a Logic for Higher-Order Concurrency". In: *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*. ICFP '13. Boston, Massachusetts, USA: Association for Computing Machinery, 2013, pp. 377–390. ISBN: 9781450323260. DOI: 10.1145/2500365.2500600.

[Tur+13] Aaron J. Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer. "Logical Relations for Fine-Grained Concurrency". In: *SIGPLAN Not.* 48.1 (Jan. 2013), pp. 343–356. ISSN: 0362-1340. DOI: 10.1145/2480359.2429111.

[Uem19] Taichi Uemura. "A General Framework for the Semantics of Type Theory". In: (Apr. 2019). eprint: 1904.04097 (math.CT). URL: https://arxiv.org/abs/1904.04097.

[Uem21] Taichi Uemura. "Abstract and Concrete Type Theories". PhD thesis. Institute for Logic, Language and Computation, University of Amsterdam, 2021.

[Uem23] Taichi Uemura. *Towards modular proof of normalization for type theories*. Talk at the WG6 meeting in Vienna. 2023.

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

[VRT22] Nachiappan Valliappan, Fabian Ruch, and Carlos Tomé Cortiñas. "Normalization for Fitch-Style Modal Calculi". In: *Proc. ACM Program. Lang.* 6.ICFP (Aug. 2022). DOI: 10.1145/3547649.

[VV20] Niccolò Veltri and Andrea Vezzosi. “Formalizing π-calculus in guarded cubical Agda”. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2020, pp. 270–283.

[Vez18] Andrea Vezzosi. *agda-flat*. 2018. URL: https://github.com/agda/agda/tree/flat.

[Voe14] Vladimir Voevodsky. “A C-system defined by a universe category”. In: *Theory and Applications of Categories* 30 (Sept. 2014).

[Wat+04] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. “A Concurrent Logical Framework: The Propositional Fragment”. In: *Types for Proofs and Programs*. Ed. by Stefano Berardi, Mario Coppo, and Ferruccio Damiani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 355–377. DOI: 10.1007/978-3-540-24849-1_23.

[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. Saarbrücken, Germany: Association for Computing Machinery, 2020, pp. 915–928. ISBN: 9781450371049. DOI: 10.1145/3373718.3394794.

[Wei22] Jonathan Weinberger. “A Synthetic Perspective on $(\infty, 1)$-Category Theory: Fibrational and Semantic Aspects”. PhD thesis. TU Darmstadt, 2022. DOI: 10.26083/tuprints-00020716.

[XE16] Chuangjie Xu and Martín Escardó. *Universes in sheaf models*. Unpublished note. 2016. URL: https://cj-xu.github.io/notes/sheaf_universe.pdf.

[Zwa19] Colin Zwanziger. “Natural Model Semantics for Comonadic and Adjoint Type Theory: Extended Abstract”. In: *Preproceedings of Applied Category Theory Conference 2019*. 2019.

[Zwa22] Colin Zwanziger. “The Natural Display Topos of Coalgebras”. PhD thesis. Carnegie Mellon University, 2022.