Higher Inductive Types and Internal Parametricity for Cubical Type Theory

Evan Cavallo

CMU-CS-21-100

February 2021

Revised May 10, 2021

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Robert Harper, Chair
Stephen Brookes
Karl Crary
Daniel R. Licata (Wesleyan University)
Anders Mörtberg (Stockholm University)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the United States Air Force Office of Scientific Research award numbers FA9550-15-1-0053 and FA9550-19-1-0216. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: dependent type theory, computational semantics, cubical type theory, internal parametricity, cohesive type theory

Abstract

Recent innovation in the design of type theories—foundational systems of mathematics with a focus on constructivity—has produced the concept of interval variable, which can be used to capture relations between objects that carry computational content. We examine two such relationships in type theory: equality, in particular quotients, and arbitrary relations as applied in parametricity interpretations.

Cubical type theory, a system using an interval-based formulation of equality, enables a permissive kind of content-carrying equality that includes in particular isomorphism. Cubical type theory provides a constructive interpretation of homotopy type theory and the Univalent Foundations, formalisms that introduced the idea of isomorphism as equality but which lack intrinsic computational meaning. The cubical approach to equality also rectifies long-standing deficiencies in the behavior of quotients in type theory. We realize a system of generalized quotients for cubical type theory originally conceived in homotopy type theory, called higher inductive types, that merges the concepts of inductive type and quotient. Such a mutual generalization is particularly essential in the contentful equality setting, but also has significant applications to ordinary mathematics.

Parametricity is, among other things, a proof technique for deriving properties of constructions performed in type theories, based on the idea that constructions preserve all relations between objects. Traditionally a meta-theoretical tool, recent work has shown that parametricity properties can be integrated into a type theory itself using an interval-based system. We develop internal parametricity on top of cubical type theory, examining the similarities and distinctions between the two applications of intervals, finding that a background of cubical equality improves the behavior of internal parametricity, and applying internal parametricity as a tool to solve difficult problems in cubical type theory. We introduce a system of cohesion modalities to express the interaction between parametric and non-parametric type theory, enabling the use of parametricity results in a non-parametric setting.

Acknowledgments

I met Bob Harper in the first year of my undergraduate degree, when I sent him an email expressing interest in functional programming. At that point I had other plans—a physics degree, for one—but over the years, Bob and the Programming Languages group coaxed me first from physics into mathematics and finally computer science, entrancing me with tales of computational trinitarianism and elegant structure. We have always been an odd couple: Bob bombastic and firm in his convictions, me milquetoast and evasive. But I would not have had it any other way; I have learned and will surely continue to learn a great deal from Bob about integrity, commitment, and passion, in science and in life. Bob, thank you for your expectations and for your boundless faith in me to meet them.

My career owes its foundation to Carlo Angiuli, who took me under his wing in my undergraduate years, has always been ready to listen to my latest madcap idea, and whose cubical type theory became the foundation for the work of this dissertation. Jon Sterling has revolutionized my world more times than I can count. I thank the many others who have passed through Bob's group and its environs in my time for all the discussions, hacking sessions, and fika we've had together: Dan Licata, Favonia, Ed Morehouse, Daniel Gratzer, Anders Mörtberg. The Principles of Programming group, and especially the Concert Reading Group, supplied me with my earliest understandings of programming languages and type theory; I am grateful to the generation of students before me for explaining it all to me and to the generation after for making me figure out how to explain it myself. Steve Awodey and his gang of philosophers taught me everything I know about category theory over years and years of seminars; I am particularly indebted to Steve, Ulrik Buchholtz, Egbert Rijke, and Mathieu Anel for their insights, but many others have influenced my work. Much of my understanding of internal parametricity was driven by discussions with the directed type theory group at the Snowbird MRC and subsequent conversations with Emily Riehl and Christian Sattler. I thank my committee for their time, and Dan in particular for suggesting a modal parametric type theory.

I thank my many friends at CMU and in Pittsburgh for keeping my spirits bright; special thanks go to the Wednesday cohort, who have kept me from hermitage in the age of distance. Finally, I thank Mom, Dad, and Lindsay, who host me for the moment, for their everlasting support. To everyone else, may we meet again in person someday!

There are places that contain you,
There are corners in your soul,
Plastic laminations in your life.
But when you're on the inside
Of the outside of your thoughts,
Do they restrain or do you stay yourself?

Now the inside of the near place
Is the outside of the far,
But you can only face your space one way.
You're really in the middle
Of the inside of yourself,
And there is only one thing we can say...

You'll never get out, you'll never get out,
You'll never get out of the cube! It's sad!
But you'll never get out, you'll never get out,
You'll never get out 'til you're dead!

— Jim Henson's *The Cube*

Contents

Introduction

This dissertation presents two applications of *interval variables* to the design of type theories: first, to representing *quotients*, and second, to *internalizing parametricity*.

A *(dependent) type theory* is a kind of mathematical framework: a language in which one can describe constructions and establish their properties. Dependent type theories trace their origins to the seminal work of Martin-Löf [Mar75; Mar82]. That work, in turn, grows out of a longer tradition of constructive mathematics, going back to Brouwer's philosophy of intuitionism. One persistent idea in the history of type theories, inherited ultimately from Brouwer, is the identification of *mathematical constructions*—including both definitions and proofs—with *programs*. In this view, a type theory is specifically a language for describing *computational* mathematics. The fundamental objects of type theory, the *types*, are collections of programs; we think of a type as a specification of computational behavior, and its elements as programs that behave in accordance with the specification.

Interval variables are a novel technical device, introduced in the past decade [CCHM15; BCM15; AFH18], for designing type theories where each type is equipped with some kind of predicate or relation on its elements. The most fundamental relation between elements in mathematics is *equality*: so much of mathematics revolves around establishing that two objects are in fact the same (or different). Despite its foundational importance, the representation of equality in type theories has been problematic since the inception of the field. Interval variables present a new approach that resolves many long-standing theoretical and practical issues with existing treatments of equality. Of key importance is the fact that interval-based equality is *contentful*: proofs of equality are programs with non-trivial computational content, in contrast to most earlier approaches. Type theories that use interval variables to represent equalities have been dubbed *cubical type theories*.

The first goal of this dissertation is to develop a general theory of quotient types for cubical type theories, in the guise of *higher inductive types*. A quotient is simply a type defined by taking a pre-existing type and declaring that certain elements should be regarded as equal. The *integers modulo n* are one classical example: an integer modulo n is simply an integer, but we regard two such integers m_0 and m_1 as equal whenever their difference is a multiple of n. As with many constructions that involve equality in an

essential way, quotients in type theory have long been ill-behaved, lacking in particular the crucial property of *effectivity*. A quotient is effective when it is possible to extract a proof of relatedness from an equality in the quotient; in the case of integers modulo n, this would mean that a proof of an equality $m_0 = m_1$ between integers modulo n can be used to compute an integer p such that $m_0 - m_1 = p \cdot n$. This essential property fails to hold in general in traditional type theories precisely because their equality proofs are contentless; with contentful interval-based equality, we are able to achieve effectivity without issue.

In a cubical type theory, it is natural to regard quotients as analogous to *inductive types*. An inductive type is one whose elements are those built from some collection of constructors. The natural numbers are the prototypical example: a natural number is a term built from the two constructors zero and $\mathrm{suc}(-)$, with $\mathrm{suc}(\mathrm{zero})$ representing 1, $\mathrm{suc}(\mathrm{suc}(\mathrm{zero}))$ representing 2, and so on. We can, in the same way, think of the equalities introduced by a quotient type as being built by constructors. *Higher inductive types* provide a mutual generalization of inductive types and quotients, allowing for both term and equality constructors. We argue that higher inductive types are a more natural abstraction than quotients in cubical type theory; they directly provide common constructions (such as *truncations*) that can only be indirectly represented using ordinary quotients. The first part of this dissertation lays out an schema for specifying higher inductive types and defines a computational realization of each such specification, providing a general framework that unifies existing examples of higher inductive types [Uni13, Chapter 6; CCHM15; AFH18; CHM18].

The second goal of this dissertation is to extend cubical type theory with *internal parametricity*. *Parametricity* is a technique used in computer science to analyze polymorphic programs, programs parameterized by type variables. Developed by Reynolds [Rey83], parametricity rests on the fact that the polymorphic programs definable in sufficiently restrictive type theories are guaranteed to *act on relations*. Parametricity theorems are normally external—they are theorems *about* type theories, not theorems proven *inside* type theories—but Bernardy and Moulin have recently shown that parametricity can be given a computational interpretation and made available *within* a type theory [BM12; BM13]. Like cubical type theory, the mechanisms of internal parametricity also rest on interval variables. Where cubical type theory uses interval variables to equip each type with an *equality* relation, internal parametricity uses intervals to equip each type with an *arbitrary* relation. In the second part of this dissertation, we show that internal parametricity can be combined with cubical type theory. By taking advantage of the good properties of cubical equality, we are able to simplify aspects of existing internally parametric type theories. Moreover, we are able to use parametricity to prove theorems involving higher inductive constructions that are prohibitively difficult to verify in plain cubical type theory.

Contributions

This dissertation is divided into four parts. Each part begins with an introduction; the introductions are meant to be more accessible than the thesis as a whole, and can be read in sequence for a more extended overview of its objectives and contributions.

Part I is a review first of dependent type theory and then cubical type theory, roughly as presented by Angiuli [Ang19]. These are the prerequisites on which the rest of the dissertation depends; each subsequent part presents an extension to the cubical type theory framework.

Part II presents our schema for higher inductive types as an extension to cubical type theory. We develop a language for specifying such types and show that each specification can be realized in type theory with a computational interpretation.

Part III extends cubical type theory with internal parametricity, which endows every construction in the theory with an action on relations. We examine the consequences of such an action, and apply it in particular to mechanically check theorems which are prohibitively difficult to prove in ordinary cubical type theory. Our motivating example uses higher inductive types, but we do not depend on the entirety of Part II; the introduction of that part is sufficient background for an intuitive understanding. We also present a formalism for the type theory and a presheaf model of that formalism.

Part IV builds on Part III, extending parametric cubical type theory with a system of cohesive modalities that allow the interaction of parametric and non-parametric constructions. This is essential for the results we prove in the previous part to be used in ordinary cubical type theory.

Publications The results of Part II and Part III have been published in the following papers.

- Evan Cavallo and Robert Harper. "Higher inductive types in cubical computational type theory". In: *PACMPL* 3.POPL (2019), 1:1–1:27. DOI: 10.1145/3290314
- Evan Cavallo and Robert Harper. "Internal Parametricity for Cubical Type Theory". In: *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain.* 2020, 13:1–13:17. DOI: 10.4230/LIPIcs.CSL.2020.13

The contents of Part II have been generalized from their form in the first paper to admit dependency and path types in recursive arguments.

Part I

Cubical type theory

Chapter 1

Introduction

1.1 Equality in type theory

A *(dependent) type theory* is a kind of framework for mathematics organized around the idea of a *type*. "Type theory" is not a term with a precise, universal definition; rather, it is a term with many definitions, some formal and mathematical, others philosophical. In practice, it refers to a vaguely-delimited constellation of systems surrounded by a common literature: a type theory is that which a type theorist studies. Nevertheless, there are some largely-unifying principles that guide the design of type theories. One is, of course, the concept of a type. Another is the idea that type theories are *constructive* or *computational*: that proofs conducted in a type theory have some kind of computational content, or that they are proofs about computational objects. Our own perspective on type theory, which derives from Martin-Löf's *Constructive Mathematics and Computer Programming* [Mar82] and Constable's subsequent program of *computational type theory* [Con09], is that type theory is a language for *classifying programs*, that is, reasoning about their computational behavior.

A type, then, is a classifier of programs, which is to say that it is a collection of programs possessing some property. A definition of a type theory consists of a specification of its types and the programs each classifies. For example, a theory might contain a type Int classifying programs that compute integers. Then "$2+2$" would be one such program (it computes the integer 4); we say $2+2$ *is an element of* Int (or *is a term of type* Int, or simply *is in* Int) and write $2+2 \in \mathsf{Int}$. The statements we make in a type theory, dubbed *judgments* by Martin-Löf, assert typehood and elementhood (Figure 1.1).

Type formers enable us to build new types from old ones: perhaps $\mathsf{Int}\times\mathsf{Int}$ is the type of programs that compute pairs of integers, while $\mathsf{Int} \to \mathsf{Int}$ is the type of programs that take an integer as input and output a new integer. In a type theory with a sufficiently expressive collection of type formers, we can formulate complex mathematical results as instances

Judgment	Reading
A type	A is a type
$A = A'$ type	A and A' are equal types
$M \in A$	M is an element of type A
$M = M' \in A$	M and M' are equal elements of type A

Figure 1.1: Judgments of a type theory (simplified)

of the typing judgment $M \in A$. For a trivial example, a program $F \in \mathsf{Int} \to \mathsf{Int}$ validates the (boring) theorem "for every integer, there exists an integer". To express something more involved, like the existence of additive inverses, we need more sophisticated types; with the machinery we will develop in Chapter 2, the type of additive inverse functions can be written as follows.

$$(n : \mathsf{Int}) \to (m : \mathsf{Int}) \times \mathsf{Id}(\mathsf{Int}, m + n, 0)$$

Glossed, this is the type of functions that take an input integer $n \in \mathsf{Int}$ and output a pair of results: another integer $m \in \mathsf{Int}$, but also a certificate that $m+n$ is equal to 0. This "type of certificates" $\mathsf{Id}(\mathsf{Int}, m + n, 0)$ is called an *identity type*: its elements are proofs that $m+n$ and 0 are *identical* (or *identified*) as elements of Int. To set up a type theory including such types, we must answer a tricky question: what kind of program constitutes a proof that two integers are the same? More broadly, how do we understand proofs of equality from a computational perspective? These questions are at the root of this thesis; as we will see, they are not at all straightforward to answer.

The history of identity types is a complex one, entangled intimately with a distinction between "extensional" and "intensional" type theories. But until relatively recently, all computational explanations of identity types have shared a common feature: the programs classified by an identity type $\mathsf{Id}(A, M, N)$ are computationally trivial. That is, the *output* or *computational behavior* of a program $P \in \mathsf{Id}(A, M, N)$ ("M and N are identified in A") is uninteresting; the only interesting question is whether such a program exists. This seems natural from a classical mathematical perspective: once you have proven two objects are equal, you need never again to think about *why* or *how* they are equal. You merely cite the theorem when you need it. However, committing to this apparently innocuous conception of equality in a computational setting actually has disastrous consequences for mathematical reasoning. For the purposes of this thesis, the most notable casualty is the *quotient type*.

Effective quotients Given a type A, a *quotient* of A is, roughly speaking, a type that has the same elements of A but where some previously distinct elements are now regarded as equal. As a simple example, we might define the *integers modulo n* (Int_n) for any natural

number $n \in \mathsf{Nat}$ as a quotient of Int. Elements of Int_n are integers, but we say that $m_0, m_1 \in \mathsf{Int}_n$ are equal as soon as they differ by some integer multiple of n. (Thus Int_3, for example, has three distinct elements: every element is equal to one of 0, 1, or 2.) In syntax, we intend the equality relation for Int_n to be given by the following type.

$$m_0 \approx m_1 \coloneqq (p : \mathsf{Int}) \times \mathsf{Id}(\mathsf{Int}, m_1 - m_0, p \cdot n)$$

That is, m_0 and m_1 are equal whenever there is some $p \in \mathsf{Int}$ equipped with a proof that $m_1 - m_0$ is equal to $p \cdot n$.

In a traditional computation-based type theory, we could have something like the following rule for deducing equalities in Int_n. A *rule* is simply a principle for deducing true judgments; we write the premises above a horizontal line and the conclusion below.

$$\frac{P \in m_0 \approx m_1}{\star \in \mathsf{Id}(\mathsf{Int}_n, m_0, m_1)}$$

That is, if we have some element P of the type $m_0 \approx m_1$, we can conclude that m_0 and m_1 are equal in Int_n. Because equality has no computational content, the program serving as evidence for this equality is simply a placeholder symbol $\star$.

Using this rule, we can check that the program that takes any element $P \in m_0 \approx m_1$ as input and returns $\star$—written $(\lambda P.\, \star)$—has the type $(m_0 \approx m_1) \to \mathsf{Id}(\mathsf{Int}_n, m_0, m_1)$. But what about the other direction? An element of $\mathsf{Id}(\mathsf{Int}_n, m_0, m_1)$ carries no information, so is no little help in constructing an element $P \in m_0 \approx m_1$. In the particular case of Int_n, we can get by with the other information we have on hand: we can compute the quotient $Q \coloneqq (m_1 - m_0)/n$ and know by the fact that $\mathsf{Id}(\mathsf{Int}_n, m_0, m_1)$ is inhabited that we will have $\langle Q, \star \rangle \in m_0 \approx m_1$. This route is not available in general, however. Consider quotienting $\mathsf{Int} \to \mathsf{Int}$, the type of functions from integers to integers, by the following relation, which identifies functions that agree on all arguments m above some number n.

$$f_0 \bowtie f_1 \coloneqq (p : \mathsf{Int}) \times ((m : \mathsf{Int}) \to (m > n) \to \mathsf{Id}(\mathsf{Int}, f_0\, m, f_1\, m))$$

Just knowing that there is *some* number p with this property will not suffice to reconstruct such a number; writing T for the quotient type, we can construct no program of type $\mathsf{Id}(T, f_0, f_1) \to (f_0 \bowtie f_1)$. There is thus a general mismatch between relations and the induced equalities in their quotient types, a failure of *effectivity of quotients*. Lack of effectivity is a serious problem: it prevents us from relating properties of T with properties of $\mathsf{Int} \to \mathsf{Int}$.

In short, quotients are constructions where data and equality collide: we frequently want to quotient by a relation whose proofs carry data (like the $p \in \mathsf{Int}$ in these examples), what we will call a *contentful relation*. We simply cannot do so in a satisfactory way when we forbid proofs of equality from carrying data. (For a more formal analysis of this incompatibility, see [Mai98].)

We are naturally led, then, to search for a computational interpretation that *does* allow equality proofs to carry data, what we will call *contentful equality*. (Effectivity of quotients is not the sole reason to do so, not by a long shot, but is a convenient potted motivation for the purposes of this thesis.) We take inspiration and intuition from two phenomena: the informal treatment of isomorphisms as equalities in everyday mathematics, and the notion of *path* in topology and homotopy theory.

Isomorphism as equality We look to *isomorphism* as an example of a contentful relation that is often treated like equality on an informal level. For an illustrative example, we draw from group theory, the study of sets equipped with binary operations and satisfying certain axioms we will not enumerate. One example of a group is the set of real numbers with the operation of addition: $(\mathbb{R}, +)$. Another is the set of positive real numbers with the operation of multiplication: $(\mathbb{R}_+, \cdot)$. These two groups are not *equal* in the standard sense, but they *are* isomorphic. That is, there are functions $\exp \in \mathbb{R} \to \mathbb{R}_+$ and $\ln \in \mathbb{R}_+ \to \mathbb{R}$ converting between the two sets that (1) are mutually inverse, meaning that $\ln(\exp(a)) = a$ and $\exp(\ln(b)) = b$, and (2) preserve the operations, meaning that $\exp(a + b) = \exp(a) \cdot \exp(b)$ and $\ln(a \cdot b) = \ln(a) + \ln(b)$. The existence of this isomorphism means that $(\mathbb{R}, +)$ and $(\mathbb{R}_+, \cdot)$ are *practically identical* from the perspective of group theory. Any "group-theoretic property" that holds of one will hold of the other. Unlike an actual equality, however, we cannot only remember that $(\mathbb{R}, +)$ and $(\mathbb{R}_+, \cdot)$ *are* isomorphic: we need to remember *how* they are isomorphic. As an example, consider the following true statement.

> For every $a \in \mathbb{R}$, we have $a + 0 = a$. ✔

If $(\mathbb{R}, +)$ and $(\mathbb{R}_+, \cdot)$ were *equal*, we would be able to replace one with the other and get another true statement. But the following is clearly false!

> For every $a \in \mathbb{R}_+$, we have $a \cdot 0 = a$. ✘

To convert results between the two groups properly, we need to *transport* the constant 0 along the isomorphism $(\exp, \ln)$. In this case, we have $\exp(0) = 1$, so the result is the following true statement.

> For every $a \in \mathbb{R}_+$, we have $a \cdot 1 = a$. ✔

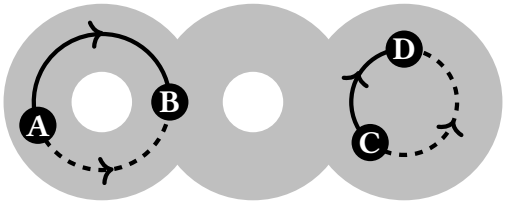
Moreover, there actually *multiple* isomorphisms between $(\mathbb{R}, +)$ and $(\mathbb{R}_+, \cdot)$: a second sends $a \in \mathbb{R}$ to $1/\exp(a) \in \mathbb{R}_+$ and $b \in \mathbb{R}_+$ to $-\ln(b) \in \mathbb{R}$. Thus, $(\mathbb{R}, +)$ and $(\mathbb{R}_+, \cdot)$ are "equal" in (at least) two different ways. When we use the fact that they are "equal" to transport facts between them, we need to be consistent about which "equality" we are

using. Isomorphism is therefore a contentful notion of equality, in that the way we use an equality depends on the contents of its proof (the functions defining the isomorphism).

To be clear, this kind of equality is usually exercised on an *informal* level in everyday mathematics. While it is possible to give precise definitions of "group-theoretic property" so that one can transport such properties between isomorphic groups, this is simply taken for granted in most mathematical writing. Isomorphisms are tacitly treated as if they were true equalities, because experience suggests that this kind of shortcut is harmless and can be eliminated by anyone who cares to be precise.

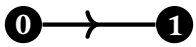
Remarkably, however, this informal use of contentful equality can not only be integrated with type theory in a completely precise way, but actually resolves several long-standing issues with existing treatments of equality, and has moreover been latent in some versions of dependent type theory since the genesis of the field. The key idea is to see equality as something along which we can *transport* results, where the effect of the transport may depend on the proof of the equality. This principle is embodied in Martin-Löf's elimination rule for equality types, the so-called J *rule*.

Equality as path Contentful relations also appear extensively in the fields of (algebraic) topology and homotopy theory in the form of *paths in a space*. These fields study abstract notions of spaces (which we will avoid defining precisely here); a *path* is simply a way to get from one point in a space to another. Let us take the following image as an example of a two-dimensional space X, with marked points **A**,**B**,**C**,**D** and a few examples of paths between them.



A path from one point to another is a contentful relationship between them: we can ask not only *whether* a path exists, but *which* path it is. Indeed, one of the central techniques for classifying spaces in algebraic topology is to count the number of "distinct" paths between given pairs of points. To do so, we also need a criterion for when two paths between a given pair of points are "distinct" or "the same". The standard device is to say that two paths are the same when there is a *homotopy* between them: a way to smoothly deform one path into the other. In our example, the two paths from **C** to **D** are homotopic, but the two paths from **A** to **B** are not, as the hole between them prevents us from deforming one into the other. A homotopy is essentially a *path between paths*; it too is a contentful relationship, one between paths rather than points. We can go further and consider paths between paths between paths and so forth.

We can formulate paths in terms of functions out of an *interval space* "$\mathbb{I}$", a space consisting of two points with a single path between them.



A path from **A** to **B** in our example space X is the same as a continuous function $\mathbb{I} \to X$ that sends **0** to **A** and **1** to **B**; in other words, a path is a picture of $\mathbb{I}$ drawn in X. A homotopy, being a path between paths, is then a function $\mathbb{I} \to (\mathbb{I} \to X)$, or equivalently a function $\mathbb{I} \times \mathbb{I} \to X$ from the product of two intervals into X.

This will be the central organizing concept of the type theory with contentful equality we are about to describe: the representation of equalities in a type as functions from an "interval" into that type.

1.2 Realizing contentful equality

We now apply the two preceding ideas, equality as path and as effecting transport, to the design of type theory. We arrive at *cubical type theory*, which was first developed in two parallel variations by Cohen, Coquand, Huber and Mörtberg [CCHM15] and Angiuli, Favonia, and Harper [AFH18].

Interval terms Cubical type theory enriches Martin-Löf's type theory with a new *interval object* $\mathbb{I}$, which behaves much like a type and is used to represent equalities. (The interval is not *actually* a type, for technical reasons that we sweep under the rug here.) Following the topological definition, we take *paths* in a type A to be functions from the interval into A.

$$P \in \mathbb{I} \to A$$

Among the elements of the interval are two distinguished "endpoint" constants, $0 \in \mathbb{I}$ and $1 \in \mathbb{I}$. Any path $P \in \mathbb{I} \to A$ is thus more specifically a path *from* $P\,0$ *to* $P\,1$. As we are usually interested in paths between a particular pair of elements, we introduce a type of *paths in* A *with fixed endpoints* $M_0 \in A$ *and* $M_1 \in A$.

$$P \in \mathsf{Path}(A, M_0, M_1)$$

The elements of this type are functions $P \in \mathbb{I} \to A$ such that $P\,0 = M_0 \in A$ and $P\,1 = M_1 \in A$. Here we come to one of the subtler aspects of type theory, cubical or otherwise: the "=" here is not the contentful equality we are in the process of fleshing out, but a separate, contentless equality we call *exact equality*. It is necessary to have such a notion of "strict" equality—to formulate the conditions on elements of path types, for one. We merely want to separate it from the paths we use as "mathematical" equalities to avoid the pitfalls of contentless equality. Exact equality differs from path equality on two axes.

First, an exact equality $M = N \in A$ is not a type but a judgment, a statement of the same kind as the elementhood judgment $M \in A$. Judgments are not themselves types, so we cannot speak about the exact equality judgment inside our type theory. That is, the following is not a well-formed type.

$$(n : \mathsf{Int}) \to ((m : \mathsf{Int}) \times (m + n = 0 \in \mathsf{Int})) \quad \text{✘}$$

In contrast, the path type *is* of course a type. We say that exact equality is an *external* equality, while the path type is an *internal* equality.

Second, $M = N \in A$ is a *substitutional* equality. This means that, given any judgment depending on an element of A, we can silently replace M with N anywhere we like without affecting the validity of the judgment. For example, if $P \in \mathsf{Path}(A, O, M)$, then it is also the case that $P \in \mathsf{Path}(A, O, N)$. In contrast, paths are merely *transportational*: if we have $Q \in \mathsf{Path}(A, M, N)$ and $P \in \mathsf{Path}(A, O, M)$, then it is not necessarily the case that $P \in \mathsf{Path}(A, O, N)$. Instead, there is an *operation*, "transport", which we can apply with Q to obtain a new term $P' \in \mathsf{Path}(A, O, N)$. In particular, the result P' can vary depending on the form of Q.

The substitutional/transportational distinction is correlated with, if not identical to, our previous contentless/contentful distinction: we think of the former as a description of the available logical principles, while the latter is a description of a computational interpretation. The external/internal and substitutional/transportational axes, on the other hand, are independent. For example, the identity type in Martin-Löf's extensional type theory is internal and substitutional.

Conversely, cubical type theory's path type is merely the internalization of an external transportational notion of path. To understand this, we need to delve a bit deeper into the details of type theory by introducing the idea of a *hypothetical judgment*. A hypothetical judgment is one that depends on some collection of typed variables (the *hypotheses*). For example, the judgment $a : A \gg M \in B$ asserts that the term M has type B under the assumption that the variable a has type A. Both M and B may make use of the variable a. As a concrete example, the judgment $m : \mathsf{Int}, n : \mathsf{Int} \gg m + n \in \mathsf{Int}$ asserts that $m + n$ is an integer whenever m and n are integers.

Using the hypothetical judgment, we can state the following rule for constructing elements of the function type $(a : A) \to B$.

$$\frac{a : A \gg N \in B}{\lambda a.\, N \in (a : A) \to B}$$

In words, if N is an element of B under the assumption that a has type A, then the function that takes in an element a of A and returns N—here written $\lambda a.\, N$—is a function of type $(a : A) \to B$. (The prefix λ for the function constructor is traditional, dating back to

Church's λ-*calculus*; we uniformly write variable bindings either in the form "$a.$" or with a type annotation as "$a : A$".)

Conversely, if we have an element of a function type $F \in (a : A) \to B$, we can *apply* it to any element $M \in A$ to obtain an element of $B[M/a]$ (the result of substituting the term M for the variable a in B).

$$\frac{F \in (a : A) \to B \qquad M \in A}{F\,M \in B[M/a]}$$

We may therefore say that the function type $(a : A) \to B$ is an *internalization* of the external concept of hypothetical judgment. Indeed, it is a unifying design principle of type theories that each type former internalizes some judgmental concept. In the case of paths, the path type $\mathsf{Path}(A, M_0, M_1)$ serves to internalize the hypothetical judgment $x : \mathbb{I} \gg - \in A$ (together with conditions on the endpoints).

The name *cubical type theory* comes from the intuitive reading of judgments such as $x_1 : \mathbb{I}, \ldots, x_n : \mathbb{I} \gg M \in A$ that depend on multiple interval variables. Where the term M in $x : \mathbb{I} \gg M \in A$ is a path or *line* in the type A, a term $x_1 : \mathbb{I}, \ldots, x_n : \mathbb{I} \gg M \in A$ is an n-dimensional *(hyper)cube* in A, filled in as each of the variables ranges between 0 and 1.

Coercion The utility of paths—the ability to transport results across them—is delivered by an operation called *coercion*. The effect of coercion is expressed by the following rule.

$$\frac{x : \mathbb{I} \gg A\ \mathsf{type} \qquad r \in \mathbb{I} \qquad s \in \mathbb{I} \qquad M \in A[r/x]}{\mathrm{coe}^{r \to s}_{x.A}(M) \in A[s/x]}$$

In words, if we have a *line of types* $x : \mathbb{I} \gg A\ \mathsf{type}$ and an inhabitant $M \in A[r/x]$ of some type along that line, then we may *coerce* it to obtain an element of any other type $A[s/x]$ along the line.

Transport along paths *within* types arises as a corollary of coercion. Suppose we have a family of types $a : A \gg B\ \mathsf{type}$ depending on a variable of type A, a path $x : \mathbb{I} \gg P \in A$ in the indexing type, and an inhabitant $N \in B[P[0/x]/a]$, which we can read as a proof that the property B holds of the term $P[0/x]$. Then we can obtain a term of type $B[P[1/x]/a]$ using coercion as follows.

$$\mathsf{transport}^{0 \to 1}_{a.B}(x.P, N) \coloneqq \mathrm{coe}^{0 \to 1}_{x.B[P/a]}(N) \in B[P[1/x]/a]$$

Thus, any "property" B that is satisfied by a term $M \in A$ is also satisfied by any term connected to M by a path.

Specifying the computational behavior of $\mathrm{coe}^{r \to s}_{x.A}(M)$ for each possible type line $x.A$ is the main technical challenge of designing a cubical type theory. (To do so requires an additional concept, path *composition*, that we will introduce later on.) Reflecting the contentful nature of path equality, this behavior does depend in general on the entirety of the line $x.A$, not only on the source and destination points $A[r/x]$ and $A[s/x]$.

Univalence With some clever programming, one may show that the coercion function $\lambda a.\, \mathrm{coe}^{0\to 1}_{x.A}(a) \in A[0/x] \to A[1/x]$ induced by a type path $x.A$ is in fact an isomorphism, with inverse given by the reverse coercion $\lambda a.\, \mathrm{coe}^{1\to 0}_{x.A}(a)$. That is, every $x : \mathbb{I} \gg A$ type induces an isomorphism between its endpoints, which we call $\mathrm{coe}^{0\simeq 1}_{x.A} \in A[0/x] \simeq A[1/x]$. In keeping with our conception of isomorphism as a kind of contentful equality, we might hope for the reverse: that every $A[0/x] \simeq A[1/x]$ induces a path from $A[0/x]$ to $A[1/x]$, with the property that coercing along said path applies the underlying function of the isomorphism. To have this correspondence would be a great boon: it would allow us to automatically transport theorems between isomorphic types, justifying formally that common informal mathematical practice.

Such a principle was first proposed by Voevodsky [Voe14] in the form of the *univalence axiom* for Martin-Löf's intensional type theory. To state the univalence axiom, we need to introduce two preliminaries: first, a more careful definition of isomorphism[1], and second, the concept of a *universe*.

Definition 1.2.1 (Isomorphism). Given a function $f \in A \to B$, a *left inverse* for f is an element of the type $\mathsf{Linv}(A, B, f)$ defined as follows.

$$\mathsf{Linv}(A, B, f) \coloneqq (g : B \to A) \times ((a : A) \to \mathsf{Path}(A, g\,(f\,a), a))$$

That is, a left inverse is a function $g \in B \to A$ such that $g\,(f\,a)$ is equal to a for all $a \in A$. A *right inverse* for f is an element of $\mathsf{Rinv}(A, B, f)$.

$$\mathsf{Rinv}(A, B, f) \coloneqq (h : B \to A) \times ((b : B) \to \mathsf{Path}(B, f\,(h\,b), b))$$

A function is an isomorphism when it has both a left and right inverse.

$$\mathsf{IsIso}(A, B, f) \coloneqq \mathsf{Linv}(A, B, f) \times \mathsf{Rinv}(A, B, f)$$

The type of isomorphisms between A and B, written $A \simeq B$, is then defined as follows.

$$(A \simeq B) \coloneqq (f : A \to B) \times \mathsf{IsIso}(A, B, f)$$

When f is an isomorphism, we can prove its left and right inverse functions $g, h \in B \to A$ are equal up to a path. Nevertheless, requiring that they be the same *a priori* leads to an ill-behaved definition of isomorphism, interprovable with but not isomorphic to the one we present here. We will not get into the reasons here, but the reader can

[1] I use *isomorphism* for what is more commonly called an *equivalence* in the homotopy type theory and cubical type theory community. I feel that isomorphism is the more suggestive term for a computer scientist's ears: "equivalence" suggests a contentless relation such as contextual equivalence or logical equivalence. Mathieu Anel has also suggested that isomorphism is a more appropriate term from an ∞-categorical perspective.

find a detailed discussion in [Uni13, Chapter 4], where several isomorphic definitions of isomorphism are presented. Voevodsky's original definition is there called a *contractible map*, while the definition we use here was suggested by Joyal and is there called a *bi-invertible map*.

A *universe* is a type whose elements are themselves types. While there cannot be a type of *all* types—this leads to paradoxes [Gir72]—we can consistently introduce a type of *some* types. It is common for type theories to contain an infinite hierarchy of universes, each contained in the next ($\mathsf{U}_0 \in \mathsf{U}_1 \in \mathsf{U}_2 \in \cdots$) and closed under operations like the product and function type formers (if $A, B \in \mathsf{U}_n$ then $A \times B \in \mathsf{U}_n$, and so on), such that every type belongs to *some* universe. Universes make it possible to express internally statements that quantify over types or talk about paths between types; Voevodsky's univalence principle is one such statement.

Definition 1.2.2 (Univalence). A universe U is *univalent* when the canonical map from paths in U to isomorphisms is itself an isomorphism.

$$\mathsf{IsUnivalent}[\mathsf{U}] :\equiv (A, B : \mathsf{U}) \to \mathsf{IsIso}(\mathsf{Path}(\mathsf{U}, A, B), (A \simeq B), \lambda p.\, \mathrm{coe}^{0 \simeq 1}_{x.\, p\, x})$$

Voevodsky's original definition of univalence was stated in terms of Martin-Löf's identity types and their elimination principle rather than paths and coercion, but the spirit is the same. Voevodsky presented the univalence axiom as an extension to Martin-Löf's *intensional type theory* (**ITT**) formalism [Mar75]. By *formalism*, we mean a collection of rules for deducing true judgments. The **ITT** formalism is validated by Martin-Löf's computational reading of the judgments: if the premises of one of the rules of **ITT** are true in that interpretation, so is the conclusion. We say therefore say that Martin-Löf's type theory is a *model* of the formalism. Voevodsky showed that **ITT** is consistent with the univalence axiom, meaning that there exists *some* model of the combined theory, by defining a classical (that is, non-computational and non-constructive) interpretation in *simplicial sets* [KL12a]. It is, however, incompatible with Martin-Löf's computational interpretation of **ITT**, which uses a contentless interpretation of equality.

The first cubical type theories were conceived in the search for a computational interpretation for **ITT** with the univalence axiom. Although the implementation of the univalence isomorphism in cubical type theory is intimidatingly technical, the bedrock that makes it all work is the contentful equality provided by paths.

In this dissertation, we are primarily interested in cubical type theories and their relatives in their own right, divorced from the originally motivating formalism of **ITT** with univalence. Paths are not only better-behaved but also in many ways more convenient for reasoning than the equality interface provided by **ITT**. One simple example is the proof of *function extensionality*, the principle that two functions are equal whenever they return equal results on all arguments. This principle is not even provable in pure **ITT**; Voevodsky observed that it is a consequence of the univalence axiom [Voe15, §11], but the proof is

non-trivial [Uni13, §4.9]. In a cubical type theory, on the other hand, its proof is incredibly simple: if two functions $f, g : A \to B$ come with a proof $p : (a : A) \to \mathsf{Path}(B, f\ a, g\ a)$, then we have $\lambda^{\mathbb{I}} x.\ \lambda a.\ p\ a\ x \in \mathsf{Path}(A \to B, f, g)$. A second concept streamlined by the use of paths for equality—and the subject of the first contribution of this thesis—is that of the higher inductive type.

Outline In this part, we review first Martin-Löf's type theory (Chapter 2) and then cubical type theory (Chapter 3). None of this material is novel. Our presentation of both theories hews fairly closely to Angiuli's dissertation [Ang19], itself a modernization and extension to the cubical setting of Allen's computational interpretation of Martin-Löf's type theory [All87]. In our description of cubical type theory, we omit some of the mechanically involved aspects, in particular the implementations of coercion in "V types" and composition in the universe. For these details, we refer to Angiuli.

In each chapter, we focus primarily on the understanding of type theories as systems for reasoning about programs. However, we also include discussion of related formalisms and their non-computational models, having developments to present on these fronts in Part III.

Chapter 2

Martin-Löf's type theory

Martin-Löf's vision of type theory is the shared core of the systems we will develop and apply in this thesis. Although we will need to modify our conception of computation to develop *cubical* type theory—and so Parts II to IV do not depend directly on the technical content of this chapter—a tour through "ordinary" type theory will be useful to establish the basic vocabulary and organizational principles on which will rely going forwards.

Following Martin-Löf's *Constructive Mathematics and Computer Programming* [Mar82] and Constable's program of *computational type theory* [Con09], we take the computational aspect of type theories as primary: a type theory is a system for reasoning about computational constructions. That is, the central judgments of a type theory—A type and $M \in A$—make assertions about the behavior of programs, here A and M. In Section 2.1, we give a computational definition of type theory in the mode of Allen [All87]. Our presentation closely follows the modernized account given by Angiuli [Ang19, Chapter 2]. Once we have defined what it means to be a type theory, we give a (fairly minimal) example: a type theory with products, natural numbers, identity types, and one universe of types.

In Section 2.2, we also present a formalism for Martin-Löf type theories, a curated collection of rules for establishing the validity of judgments (A type, $M \in A$, and so on). A formalism will not capture completely the truth defined by a computational type theory, but this is not the objective: the goal is to specify a useful *interface*, which can serve as an abstract window on type theories computational and otherwise. A formalism must balance expressivity and applicability, be abstract enough to serve as an interface to a variety of type theories but concrete enough to prove the kinds of theorems its user wants to prove. The study of formalisms is in particular essential for the implementation of proof assistants, computer systems that check the validity of mathematical arguments within some formalism and assist users in constructing them. An informed choice of formalism can make a great difference in the usability of a system, particularly by affecting the degree to which bureaucratic arguments can be automated.

After designing a formalism with an eye towards a computational interpretation, we

Judgment	Reading
Γ ctx	Γ is a context
$\Gamma \gg A$ type	A is a type in context Γ
$\Gamma \gg A = A'$ type	A and A' are equal types in context Γ
$\Gamma \gg M \in A$	M is an element of type A in context Γ
$\Gamma \gg M = M' \in A$	M and M' are equal elements of A in context Γ

Figure 2.1: Judgments of Martin-Löf type theories

can moreover consider alternative interpretations, that is, readings of the judgments that validate the rules of the formalism. When we prove a result in a formalism, we obtain a result in each of its interpretations; thus breadth of interpretability is another goal in formalism design. In Section 2.2.2 we briefly describe one alternative interpretation for our formalism, interpreting types as classical sets.

2.1 A logic of programs

A type theory is a system for *behavioral* classification of programs: a judgment in type theory asserts something about how a program behaves. More specifically, it tells us something about the *value* that a program returns; for example, the judgment $N \in \mathsf{Int}$ asserts that N evaluates to a value V that is an integer.

A type theory is therefore defined by two components. First, we need a definition of *program*: what does it mean to evaluate the term N? This component is the *operational semantics*. Second, we need a system for classifying the values that are returned by programs: what does it mean for the value V returned by N to belong to the type Int? This component is the *value type system*. From the two components, we derive an interpretation for the judgments of Martin-Löf type theory, which are shown in Figure 2.1.

2.1.1 Operational semantics

We specify our language of untyped programs by a *structural operational semantics* [Plo04]. A structural operational semantics describes the evaluation of programs in terms of two judgments, M val ("M is a value") and $M \longmapsto N$ ("M steps to N"). These judgments operate on closed programs, *i.e.*, programs not containing any free variables. A particular operational semantics is specified by defining these two judgments, typically as generated by a collection of reduction rules.

Definition 2.1.1. An *operational semantics* is a definition of two judgments M val and $M \longmapsto N$ on closed terms, satisfying the following *determinism* properties.

- If $M \longmapsto N$ and $M \longmapsto N'$, then $N = N'$.
- It is never the case that both M val and $M \longmapsto N$ for some N.

Given an operational semantics, which specifies the one-step behavior of a program—either it is an inert value or reduces—we derive judgments $M \longmapsto^* N$ ("M reduces to N") and $M \Downarrow V$ ("M evaluates to "V") as generated by the following rules.

$$\frac{}{M \longmapsto^* M} \qquad \frac{M \longmapsto^* N \qquad N \longmapsto P}{M \longmapsto^* P} \qquad \frac{M \longmapsto^* V \qquad V \text{ val}}{M \Downarrow V}$$

That is, $M \longmapsto^* N$ holds when M becomes N after zero or more steps, while $M \Downarrow V$ holds when M becomes the value V after zero or more steps. We write *Val* for the collection of values.

Remark 2.1.2. It would suffice to require not determinism but merely *confluence*: that if $M \longmapsto N$ and $M \longmapsto N'$, then there is some P such that $N \longmapsto P$ and $N' \longmapsto P$. What we really need is that evaluation $M \Downarrow V$ produces unique results.

2.1.2 Value type system

A value type system defines two notions: the values that are names for types (such as Int) and the values that are elements of those types (such as 0,1,2,…). Each of these is specified in a binary way by a *partial equivalence relation*, which simultaneously specifies which terms *are* types/elements and when they are *equal* as types/elements.

Notation 2.1.3. Given a binary relation R on terms—that is, a set of pairs of terms—and two terms M, M', we write $M \approx M' \in R$ as syntactic sugar for $(M, M') \in R$, and $M \in R$ for $(M, M) \in R$.

Definition 2.1.4. A *partial equivalence relation (PER)* on terms is a binary relation satisfying the following properties.

- *Symmetry:* If $M \approx N \in R$ then $N \approx M \in R$.
- *Transitivity:* If $M \approx N \in R$ and $N \approx P \in R$, then $M \approx P \in R$.

The *field* of a partial equivalence relation R is the collection of terms M such that $M \in R$; restricted to its field, R becomes an equivalence relation. Thus, a PER concisely specifies a subset of the collection of all terms and an equivalence relation on that set.

Definition 2.1.5. A *candidate type system* is a ternary relation $\tau \subseteq \mathit{Val} \times \mathit{Val} \times \mathit{PER}(\mathit{Val})$ that relates values V, V', and partial equivalence relations R on values.

Notation 2.1.6. Given a candidate type system τ, we write $\tau \vDash V \approx V' \downarrow R$ as syntactic sugar for $(V, V', R) \in \tau$, and $\tau \vDash V \downarrow R$ for $(V, V, R) \in \tau$. We write $\tau[R]$ for the binary relation $\tau \vDash (-) \approx (-) \downarrow R$, which relates types when they are equated by τ with interpretation R.

We read an instance $\tau \vDash V \approx V' \downarrow R$ of the relation as asserting that V and V' are equal type names in τ and that their elements are defined by the PER R: $W \approx W' \in R$ means that W and W' are equal elements of the type named by V (or V').

Definition 2.1.7. A candidate type system is a *type system* when it satisfies the following additional axioms.

- *PER:* For any fixed PER R, the relation $\tau[R]$ is a partial equivalence relation.
- *Unicity:* If $\tau \vDash V \approx V' \downarrow R$ and $\tau \vDash V \approx V' \downarrow R'$, then $R = R'$.

The former ensures that value type equality is a partial equivalence relation; the latter ensures that each type name has at most one interpretation as a relation.

2.1.3 Typing judgments

Given an operational semantics and candidate type system τ, we derive an interpretation of the typing judgments in two stages: first, we extend the type system to closed terms that may not be values, then to open terms. The status of non-value closed terms is determined by evaluating them: if terms evaluate to equal values, then they are equal terms.

Definition 2.1.8. Let R be a relation. We define a relation $\Downarrow R$ as follows: $M \approx M' \in \Downarrow R$ holds when there exist values V, V' such that $M \Downarrow V$, $M' \Downarrow V'$, and $V \approx V' \in R$.

Definition 2.1.9 (Closed judgments).

- *Closed types:* $\Vdash A = A'$ type is defined to hold when $A \approx A' \in \Downarrow\tau[R]$ for some R.
- *Closed terms:* $\Vdash M = M' \in A$ is defined to hold when $A \in \Downarrow\tau[R]$ for some R such that $M \approx M' \in \Downarrow R$.

The unary judgment $\Vdash A$ type is shorthand for $\Vdash A = A$ type. Likewise, $\Vdash M \in A$ is shorthand for $\Vdash M = M \in A$.

Both types and terms are programs: a type is simply a program that computes the name of a value type (as specified by the type system). Note that if $\Vdash M \in A$ and $M \Downarrow V$, we always have $\Vdash M = V \in A$.

The open judgments are defined by *functionality*: an open type is well-formed when it takes equal instantiations of its variables to equal closed types, and likewise for elements. The two open judgments are defined together with the *context judgment* Γ ctx and the *closing substitution* judgment $\Vdash \gamma \in \Gamma$. A context is a collection of typed variables $(a_1 : A_1, \ldots, a_n : A_n)$, while a closing substitution into a context Γ is a list of instantiations $(M_1/a_1, \ldots, M_n/a_n)$ for the variables listed in Γ. Given a term M and a substitution γ, we write $M\gamma$ for the result of applying the substitutions in γ to M.

Definition 2.1.10 (Contexts, closing substitutions, and open judgments).

- *Closing substitutions:* $\Vdash \gamma = \gamma' \in \Gamma$ is the least judgment closed under the following rules.

$$\frac{}{\Vdash \cdot = \cdot \in \cdot} \qquad \frac{\Vdash \gamma = \gamma' \in \Gamma \qquad \Vdash M = M' \in A\gamma}{\Vdash (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma, a : A)}$$

- *Open types:* $\Gamma \gg A = A'$ type is defined to hold when $\Vdash A\gamma = A'\gamma'$ type holds for all $\Vdash \gamma = \gamma' \in \Gamma$.
- *Open terms:* $\Gamma \gg M = M' \in A$ is defined to hold when $\Vdash M\gamma = M\gamma' \in A\gamma$ holds for all $\Vdash \gamma = \gamma' \in \Gamma$.
- *Contexts:* $\Gamma = \Gamma'$ ctx is the least judgment closed under the following rules.

$$\frac{}{\cdot = \cdot \text{ ctx}} \qquad \frac{\Gamma = \Gamma' \text{ ctx} \qquad \Gamma \gg A = A' \text{ type}}{(\Gamma, a : A) = (\Gamma', a : A') \text{ ctx}}$$

The unary judgments $\Vdash \gamma \in \Gamma$, $\Gamma \gg A$ type, $\Gamma \gg M \in A$, and Γ ctx are shorthand for $\Vdash \gamma = \gamma \in \Gamma$, $\Gamma \gg A = A$ type, $\Gamma \gg M = M \in A$, and $\Gamma = \Gamma$ ctx respectively.

Notation 2.1.11. When we want to emphasize the dependence of the judgments on the background type system, we add the prefix $\tau \vDash \cdots$, as in $\tau \vDash \Gamma \gg A = A'$ type.

Notation 2.1.12. In a value type system τ, given A type, we write $[\![A]\!]^\tau$ for the necessarily unique value relation such that $\tau \vDash A \downarrow [\![A]\!]^\tau$. We omit the annotation τ when it is clear from context.

Having defined the type-theoretic judgments, we can begin checking that they satisfy the kind of properties we would expect, assembling a collection of rules that can be used to build up larger results without explicitly working with the definitions of the judgments. From this point forward, we assume that our candidate type system τ is a genuine type system. As all of these rules are standard and fairly intuitive, we will not provide proofs except to give a feel for the general shape of the arguments; for a more thorough tour, we refer as always to [Ang19].

Symmetry and transitivity We warm up by proving symmetry and transitivity of the binary judgments, which follow more or less immediately from the corresponding properties of the value type system. It is convenient to first prove the rules for the closed judgments, then extend them uniformly to the open judgments.

Rules 2.1.13 (Symmetry and transitivity for closed judgments).

$$\frac{\Vdash A = B\ \mathsf{type}}{\Vdash B = A\ \mathsf{type}} \qquad \frac{\Vdash A = B\ \mathsf{type} \qquad \Vdash B = C\ \mathsf{type}}{\Vdash A = C\ \mathsf{type}}$$

$$\frac{\Vdash M = N \in A}{\Vdash N = M \in A} \qquad \frac{\Vdash M = N \in A \qquad \Vdash N = P \in A}{\Vdash M = P \in A}$$

Proof. Consider first the symmetry of the typing judgment. By definition of $\Vdash A = B\ \mathsf{type}$, our assumption is that $A \Downarrow A_0$ and $B \Downarrow B_0$ for some values A_0, B_0 and $\tau \vDash A_0 \approx B_0 \downarrow R$ for some R. By symmetry of the value type system, it follows that $\tau \vDash B_0 \approx A_0 \downarrow R$ and thus that $\Vdash B = A\ \mathsf{type}$.

For transitivity, $\Vdash A = B\ \mathsf{type}$ tells us that $A \Downarrow A_0$ and $B \Downarrow B_0$ with $\tau \vDash A_0 \approx B_0 \downarrow R$, while $\Vdash B = C\ \mathsf{type}$ tells us that $B \Downarrow B_0'$ and $C \Downarrow C_0$ with $\tau \vDash B_0' \approx C_0 \downarrow R'$. By determinism of the type system, we know that $B_0 = B_0'$. Applying symmetry and transitivity of the value type system, we can conclude that $\tau \vDash B_0 \downarrow R$ and $\tau \vDash B_0 \downarrow R'$; thus $R = R'$ by unicity. Finally, transitivity of the value type system now applied with $\tau \vDash A_0 \approx B_0 \downarrow R$ and $\tau \vDash B_0 \approx C_0 \downarrow R$ gives the result.

Symmetry and transitivity for terms follow by similar arguments, this time using the fact that the relations returned by a value type system are always PERs. □

Rules 2.1.14 (Symmetry and transitivity for closing substitutions).

$$\frac{\Gamma\ \mathsf{ctx} \qquad \Vdash \gamma = \gamma' \in \Gamma}{\Vdash \gamma' = \gamma \in \Gamma} \qquad \frac{\Gamma\ \mathsf{ctx} \qquad \Vdash \gamma = \gamma' \in \Gamma \qquad \Vdash \gamma' = \gamma'' \in \Gamma}{\Vdash \gamma = \gamma'' \in \Gamma}$$

Proof. By induction on the defining rules for closing substitutions and Rules 2.1.13. □

Rules 2.1.15 (Symmetry and transitivity for open judgments).

$$\frac{\Gamma\ \mathsf{ctx} \qquad \Gamma \gg A = B\ \mathsf{type}}{\Gamma \gg B = A\ \mathsf{type}} \qquad \frac{\Gamma\ \mathsf{ctx} \qquad \Gamma \gg A = B\ \mathsf{type} \qquad \Gamma \gg B = C\ \mathsf{type}}{\Gamma \gg A = C\ \mathsf{type}}$$

$$\frac{\Gamma\ \mathsf{ctx} \qquad \Gamma \gg M = N \in A}{\Gamma \gg N = M \in A} \qquad \frac{\Gamma\ \mathsf{ctx} \qquad \Gamma \gg M = N \in A \qquad \Gamma \gg N = P \in A}{\Gamma \gg M = P \in A}$$

Proof. We give the proof for type symmetry; the others follow the same pattern. Suppose that $\Gamma \gg A = B$ type. To show $\Gamma \gg B = A$ type, let an arbitrary $\Vdash \gamma = \gamma' \in \Gamma$; we must show that $\Vdash B\gamma = A\gamma'$ type. By symmetry of the closing substitution judgment, we have $\Vdash \gamma' = \gamma \in \Gamma$. Applying $\Gamma \gg A = B$ type with this substitution, we get $\Vdash A\gamma' = B\gamma$ type. By symmetry of the closed typing judgment, we thus conclude $\Vdash B\gamma = A\gamma'$ type. □

Rules for open judgments follow in general from the closed case in this fashion: each instance of the hypotheses implies the corresponding instance of the conclusion. For this reason, we will typically only give proofs for closed versions of rules.

Exact coercion Unicity of the value type system implies the important *exact coercion* rule, which allows us to transfer terms between equal types.

Rule 2.1.16 (Exact coercion).

$$\frac{\Gamma \gg M = M' \in A \qquad \Gamma \gg A = B \text{ type}}{\Gamma \gg M = M' \in B}$$

Structural rules The *structural rules* describe the behavior of variables in the context. *Weakening* states that any true judgment is still true in the presence of additional hypotheses; *cut* allows us to substitute terms for variables. For types, for example, we have the following; similar principles apply to their elements.

Rules 2.1.17 (Structural rules for types).

Weakening
$$\frac{\Gamma \gg A = A' \text{ type} \qquad \Gamma \gg B \text{ type}}{\Gamma, b : B \gg A = A' \text{ type}}$$

Cut
$$\frac{\Gamma, b : B \gg A = A' \text{ type} \qquad \Gamma \gg N = N' \in B}{\Gamma \gg A[N/b] = A'[N'/b] \text{ type}}$$

Open substitutions Although not strictly necessary to fill out the picture in Figure 2.1, it is useful to introduce a notion of *open* substitutions, substitutions from one context to another. We can define these in the same way we defined closed substitutions, just with all judgments parameterized by another context.

Definition 2.1.18 (Open substitutions). We define $\Gamma' \gg \gamma = \gamma' \in \Gamma$ to be the least judgment closed under the following principles.

$$\frac{}{\Gamma' \gg \cdot = \cdot \in \cdot} \qquad \frac{\Gamma' \gg \gamma = \gamma' \in \Gamma \qquad \Gamma' \gg M = M' \in A\gamma}{\Gamma' \gg (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma, a : A)}$$

An essential property of the open judgments is their *stability under substitution*: if some open judgment $\Gamma \gg \mathcal{J}$ holds and we have a substitution $\Gamma' \gg \gamma \in \Gamma$, then $\Gamma' \gg \mathcal{J}\gamma$ also holds.

Rules 2.1.19 (Stability under substitution).

$$\frac{\Gamma' \gg \gamma = \gamma' \in \Gamma \qquad \Gamma \gg A = A' \text{ type}}{\Gamma' \gg A\gamma = A'\gamma' \text{ type}} \qquad \frac{\Gamma' \gg \gamma = \gamma' \in \Gamma \qquad \Gamma \gg M = M' \in A}{\Gamma' \gg M\gamma = M'\gamma' \in A\gamma}$$

$$\frac{\Gamma' \gg \gamma = \gamma' \in \Gamma \qquad \Gamma \gg \delta = \delta' \in \Delta}{\Gamma' \gg \delta\gamma = \delta'\gamma' \in \Delta}$$

In particular, we can see the structural rules as arising from this principle. For any Γ ctx, we have a trivial *identity substitution* $\Gamma \gg \mathsf{id}_\Gamma \in \Gamma$ that replaces each variable with itself. Weakening for types and terms then follows from the fact that we also have $\Gamma, b : B \gg \mathsf{id}_\Gamma \in \Gamma$. Similarly, cut follows from the existence of the extended substitutions $\Gamma \gg (\mathsf{id}_\Gamma, N/b) = (\mathsf{id}_\Gamma, N'/b) \in (\Gamma, b : B)$ for any $\Gamma \gg N = N' \in B$. Finally, note that the stability of substitutions themselves under substitution gives us *composition* of substitutions.

2.1.4 Constructing a type system

Now that we have seen how to obtain a type theory from an operational semantics and type system, let us instantiate the framework with an example or two. We list the defining operational semantics rules for a bare-bones language in Figure 2.2, the terms of which are drawn from the following grammar.

$$\begin{aligned} A, B, M, N, P \quad ::= \quad & a \mid (a : A) \to B \mid \lambda a.\, N \mid N\, M \\ & \mid (a : A) \times B \mid \langle M, N \rangle \mid \mathsf{fst}(P) \mid \mathsf{snd}(P) \\ & \mid \mathsf{Nat} \mid \mathsf{zero} \mid \mathsf{suc}(M) \mid \mathsf{elim}_{\mathsf{Nat}}(n.B; M; N, n.b.P) \\ & \mid \mathsf{Id}(A, M, N) \mid \mathsf{refl}(M) \mid \mathsf{elim}_{\mathsf{Id}}(a_0.a_1.p.B, P, a.N) \\ & \mid \mathsf{Unit} \mid \star \\ & \mid \mathsf{Void} \mid \mathsf{abort} \\ & \mid \mathsf{U} \end{aligned}$$

To define a value type system closed under the various type formers, we use the *Knaster-Tarski fixed-point theorem* [Tar55; DP02, §8.20], which states that any monotone operator on a complete lattice has a least fixed-point. (Here, we only need the theorem for lattices of subsets.)

Functions

$$\frac{}{(a : A) \to B \text{ val}} \qquad \frac{}{\lambda a.\, N \text{ val}} \qquad \frac{F \longmapsto F'}{F\, M \longmapsto F'\, M} \qquad \frac{}{(\lambda a.\, N)\, M \longmapsto N[M/a]}$$

Products

$$\frac{}{(a : A) \times B \text{ val}} \qquad \frac{}{\langle M, N \rangle \text{ val}} \qquad \frac{P \longmapsto P'}{\mathsf{fst}(P) \longmapsto \mathsf{fst}(P')} \qquad \frac{P \longmapsto P'}{\mathsf{snd}(P) \longmapsto \mathsf{snd}(P')}$$

$$\frac{}{\mathsf{fst}(\langle M, N \rangle) \longmapsto M} \qquad \frac{}{\mathsf{snd}(\langle M, N \rangle) \longmapsto N}$$

Natural numbers

$$\frac{}{\mathsf{Nat} \text{ val}} \qquad \frac{}{\mathsf{zero} \text{ val}} \qquad \frac{}{\mathsf{suc}(M) \text{ val}}$$

$$\frac{N \longmapsto N'}{\mathsf{elim}_{\mathsf{Nat}}(n.B; N; Z, n.b.S) \longmapsto \mathsf{elim}_{\mathsf{Nat}}(n.B; N'; Z, n.b.S)}$$

$$\frac{}{\mathsf{elim}_{\mathsf{Nat}}(n.B; \mathsf{zero}; Z, n.b.S) \longmapsto Z}$$

$$\frac{}{\mathsf{elim}_{\mathsf{Nat}}(n.B; \mathsf{suc}(N); Z, n.b.S) \longmapsto S[N/n, \mathsf{elim}_{\mathsf{Nat}}(n.B; N; Z, n.b.S)/b]}$$

Identity types

$$\frac{}{\mathsf{Id}(A, M_0, M_1) \text{ val}} \qquad \frac{}{\mathsf{refl}(M) \text{ val}}$$

$$\frac{P \longmapsto P'}{\mathsf{elim}_{\mathsf{Id}}(a_0.a_1.p.B, P, a.N) \longmapsto \mathsf{elim}_{\mathsf{Id}}(a_0.a_1.p.B, P', a.N)}$$

$$\frac{}{\mathsf{elim}_{\mathsf{Id}}(a_0.a_1.p.B, \mathsf{refl}(M), a.N) \longmapsto N[M/a]}$$

Unit *Void* *Universe*

$$\frac{}{\mathsf{Unit} \text{ val}} \qquad \frac{}{\star \text{ val}} \qquad \frac{}{\mathsf{Void} \text{ val}} \qquad \frac{}{\mathsf{U} \text{ val}}$$

Figure 2.2: Operational semantics for a bare-bones type theory

Theorem 2.1.20 (Knaster-Tarski for power sets). Let S be a set and let F be an operator on subsets of S. Suppose that F is monotone: if $T \subseteq T' \subseteq S$, then $F(T) \subseteq F(T')$. Then there is a subset $\mu F \subseteq S$ satisfying the following properties.

- μF is a *fixed-point* of F: we have $F(\mu F) = \mu F$.
- μF is the *least pre-fixed-point* of F: if any subset $T \subseteq S$ satisfies $F(T) \subseteq T$, then $\mu F \subseteq T$.

In particular, μF is also the *least fixed-point* of F.

We construct value type systems by applying this theorem to the set $Val \times Val \times PER(Val)$, the subsets of which are the candidate value type systems. We also apply the theorem with $Val \times Val$ in order to construct the relations interpreting individual types with recursive structure, such as the natural numbers and inductive types more generally. For these purposes, it is necessary that we obtain not only relations but PERs; in the case of a candidate type system, we also need to check the uniqueness condition. To do so, we introduce the following definitions.

Definition 2.1.21. Let R be a binary relation. We define relations $Sym^+(R)$ and $Trans^+(R)$ as follows.

- $M \approx M' \in Sym^+(R)$ holds when $M \approx M' \in R$ and $M' \approx M \in R$.
- $M \approx M' \in Trans^+(R)$ when $M \approx M' \in R$ and for any term N, we have:
  - if $N \approx M \in R$, then $N \approx M' \in R$
  - if $M' \approx N \in R$, then $M \approx N' \in R$.

Proposition 2.1.22. Given any binary relation R, we have $Sym^+(\Downarrow R) = \Downarrow Sym^+(R)$ and $Trans^+(\Downarrow R) = \Downarrow Trans^+(R)$.

The following lemma, a trivial consequence of the universal property of the least pre-fixed-point, provides a convenient set of conditions we can check to verify PER-hood.

Lemma 2.1.23. Let F be a monotone operator on binary relations. If $F(Sym^+(\mu F)) \subseteq Sym^+(\mu F)$, then μF is symmetric; if $F(Trans^+(\mu F)) \subseteq Trans^+(\mu F)$, then μF is transitive. In particular, if both hold, then μF is a PER.

Proof. The hypotheses say exactly that $Sym^+(\mu F)$ and $Trans^+(\mu F)$ are pre-fixed-points of F, thus that $\mu F \subseteq Sym^+(\mu F)$ and $\mu F \subseteq Trans^+(\mu F)$. By inspection of the definitions of $Sym^+(\mu R)$ and $Trans^+(\mu F)$, these two inclusions give that μF is symmetric and transitive respectively. □

For example, we may define a relation for *natural numbers* to be generated by zero and suc ("successor") constructors and check that this is a PER.

Example 2.1.24 (Natural numbers PER). We define a monotone operator F on relations. For any R, $F(R)$ is the union of the following clauses.

- $\mathsf{zero} \approx \mathsf{zero} \in F(R)$.
- $\mathsf{suc}(M) \approx \mathsf{suc}(M) \in F(R)$ whenever $M \approx M' \in \Downarrow R$.

Set $\mathit{Nat} \coloneqq \mu F$. By simple inspection, we have $F(\mathit{Sym}^+(R)) = \mathit{Sym}^+(F(R))$ for any R, so in particular $F(\mathit{Sym}^+(\mu F)) = \mathit{Sym}^+(\mu F)$. For transitivity, suppose we have $V \approx V' \in F(\mathit{Trans}^+(\mu F))$. We want to show $W \approx W' \in \mathit{Trans}^+(\mu F)$, so let W, W' with $W \approx V \in \mu F$ and $V' \approx W' \in \mu F$ be given. By definition of F, we are in one of two cases: either $V = V' = \mathsf{zero}$, or $V = \mathsf{suc}(M)$ and $V' = \mathsf{suc}(M')$ with $M \approx M' \in \Downarrow \mathit{Trans}^+(\mu F)$. In the former case, we must also have $W = W' = \mathsf{zero}$, so $W \approx W' \in \mu F$. In the latter, we can first conclude that $W = \mathsf{suc}(N)$ and $W' = \mathsf{suc}(N')$ with $N \approx M \in \Downarrow \mu F$ and $M' \approx N' \in \Downarrow \mu F$. From $M \approx M' \in \Downarrow \mathit{Trans}^+(\mu F)$, we have $M \approx M' \in \mathit{Trans}^+(\Downarrow \mu F)$ by Proposition 2.1.22. It follows that $N \approx N' \in \Downarrow \mu F$, and thus that $\mathsf{suc}(N) \approx \mathsf{suc}(N') \in \mu F$.

We define analogous operators on candidate type systems.

Definition 2.1.25. Let τ be a candidate value type system. We define candidate value type systems $\mathit{Sym}^+(\tau)$, $\mathit{Trans}^+(\tau)$, and $\mathit{Uni}^+(\tau)$ as follows.

- $\mathit{Sym}^+(\tau) \vDash V \approx V' \downarrow R$ holds when $V \approx V' \in \mathit{Sym}^+(\tau[R])$.
- $\mathit{Trans}^+(\tau) \vDash V \approx V' \downarrow R$ holds when $V \approx V' \in \mathit{Trans}^+(\tau[R])$.
- $\mathit{Uni}^+(\tau) \vDash V \approx V' \downarrow R$ holds when $\tau \vDash V \approx V' \downarrow R$ and for any other PER R', if $\tau \vDash V \approx V' \downarrow R'$ holds then $R = R'$.

Lemma 2.1.26. Let F be a monotone operator on candidate value type systems such that $F(\mathit{Uni}^+(\mu F)) \subseteq \mathit{Uni}^+(\mu F)$, $F(\mathit{Sym}^+(\mu F)) \subseteq \mathit{Sym}^+(\mu F)$, and $F(\mathit{Trans}^+(\mu F)) \subseteq \mathit{Trans}^+(\mu F)$. Then μF is a value type system.

Proof. Following the proof of Lemma 2.1.23. □

Our first example of a type system includes function, product, and identity types, a natural numbers type, and unit and empty types.

Example 2.1.27 (Small type system). We define an operator F on candidate value type systems as follows: given τ, $F(\tau)$ is the union of the following clauses.

- $F(\tau) \vDash (a : A) \to B \approx (a : A') \to B' \downarrow R$ whenever
  - $A \approx A' \in \Downarrow\tau[S]$ for some PER S,
  - we have a family of value PERs $(T_M)_{M \in \Downarrow S}$ such that for every $M \approx M' \in \Downarrow S$, we have $B[M/a] \approx B'[M'/a] \in \Downarrow\tau[T_M]$ and $T_M = T_{M'}$,
  - $V \approx V' \in R$ holds exactly when $V = \lambda a.\, N$ and $V' = \lambda a.\, N'$ for some N, N' with $N[M/a] \approx N'[M'/a] \in \Downarrow T_M$ for every $M \approx M' \in \Downarrow S$.
- $F(\tau) \vDash (a : A) \times B \approx (a : A') \times B' \downarrow R$ whenever
  - we have S and $(T_M)_{M \in \Downarrow S}$ as in the function type clause,
  - $V \approx V' \in R$ holds exactly when $V = \langle M, N \rangle$ and $V' = \langle M', N' \rangle$ for some M, M', N, N' with $M \approx M' \in \Downarrow S$ and $N \approx N' \in \Downarrow T_M$.
- $F(\tau) \vDash \mathsf{Id}(A, M_0, M_1) \approx \mathsf{Id}(A, M_0', M_1') \downarrow R$ whenever
  - $A \approx A' \in \Downarrow\tau[S]$ for some PER S,
  - $M_0 \approx M_0' \in \Downarrow S$,
  - $M_1 \approx M_1' \in \Downarrow S$,
  - $V \approx V' \in R$ holds exactly when $V = \mathsf{refl}(M)$ and $V' = \mathsf{refl}(M')$ with $M \approx M' \in \Downarrow S$, $M \approx M_0 \in \Downarrow S$, and $M \approx M_1 \in \Downarrow S$.
- $F(\tau) \vDash \mathsf{Nat} \approx \mathsf{Nat} \downarrow \mathit{Nat}$, where Nat is as defined in Example 2.1.24.
- $F(\tau) \vDash \mathsf{Unit} \approx \mathsf{Unit} \downarrow R$ when $V \approx V' \in R$ if and only if $V = V' = \star$.
- $F(\tau) \vDash \mathsf{Void} \approx \mathsf{Void} \downarrow R$ when R is the empty relation.

We define the candidate value type system τ_0 to be the least fixed point of F.

Proposition 2.1.28. τ_0 is a value type system.

Proof. By way of Lemma 2.1.26, as in Example 2.1.24; see [Ang19, Lemma 2.6] for an explicit proof. □

We show below that τ_0 validates standard rules for each of the types included. We may use τ_0 as a stepping stone to construct a type system τ_1 with a universe: the elements of U in τ_1 will be the types of τ_0.

Example 2.1.29 (Type system with one universe). We define a monotone operator U on candidate type systems as follows: given τ, $U(\tau) \vDash V \approx V' \downarrow R$ holds when $V = V' = \mathsf{U}$ and R is the relation $W \approx W' \in R \iff \exists S.\ \tau \vDash W \approx W' \downarrow S$. We define a candidate value type system τ_1 to be the least fixed point of the monotone operator $\tau \mapsto F(\tau) \cup U(\tau_0)$.

Proposition 2.1.30. τ_1 is a value type system.

Both τ_1 and its universe U will be closed under functions, products, and identity types and contain natural numbers, unit, and empty types; the universe does not of course contain itself. As demonstrated in [Ang19, §2.2], we could repeat this process to define type systems τ_n with n universes, and ultimately a type system $\tau_\omega \coloneqq \bigcup_n \tau_n$ with an infinite hierarchy of universes $\mathsf{U}_0 \in \mathsf{U}_1 \in \mathsf{U}_2 \in \cdots$. For our purposes, a single universe is sufficient, so we satisfy ourselves with τ_1.

Remark 2.1.31. In the case that a monotone operator F on a complete lattice is *Scott-continuous*—that is, preserves directed sets—its fixed point may be characterized as the union of the sequence $\emptyset \subseteq F(\emptyset) \subseteq F^2(\emptyset) \subseteq \cdots$ [Sco72]. The operator F used in Example 2.1.27 is not continuous, however, as a counterexample observed by Harper demonstrates [Har92, Theorem 7.1]. Consider the following type.

$$A \coloneqq (n : \mathsf{Nat}) \to \mathsf{elim}_{\mathsf{Nat}}(\_.\mathsf{U}; n; \mathsf{Nat}, \_.B.B \times \mathsf{Nat})$$

This is the type of functions that, for every natural number n, returns an element of the $(n+1)$-fold product of Nat. For each $n \in \mathbb{N}$, the candidate type system $F^n(\emptyset)$ for F defined in Example 2.1.27 only contains the k-fold product of Nat for every $k \leq n$, so the union $\bigcup_{n\in\mathbb{N}} F^n(\emptyset)$ does not contain A. The least fixed point of F, on the other hand, *does* contain this type.

2.1.5 Rules for type and term formers

We now take a look at the specific properties of τ_0 and τ_1, namely the types that they support. The rules we check for each type follow a general pattern we will see repeated across every type former we introduce in this thesis: there are *formation*, *introduction*, *elimination*, *reduction*, and (possibly) *uniqueness* rules.

2.1.5.1 Functions

To start with, let us consider function types, which we have included in both τ_0 and τ_1. Our first rule, formation, gives conditions under which a function type is well-formed. For this rule and for all subsequent rules, we give a proof for the case where the conclusion is a closed judgment. The rule for open judgments then follows by applying the closed rule pointwise, as in the derivation of Rules 2.1.15 from Rules 2.1.13 above.

Rule 2.1.32 (Function formation).

$$\frac{\Vdash A = A' \ \mathsf{type} \qquad a : A \gg B = B' \ \mathsf{type}}{\Vdash (a : A) \to B = (a : A') \to B' \ \mathsf{type}}$$

Proof. Function types are always values: we have $(a : A) \to B \Downarrow (a : A) \to B$ and likewise for $(a : A') \to B'$. Thus the conclusion follows from the hypotheses and the definition of the type system, which gives $\tau_i \vDash (a : A) \to B \approx (a : A') \to B' \downarrow R$ (for a certain R). □

Next, we have the introduction rule, which gives conditions under which we may construct (*i.e.*, introduce) an element of a function type. An element of $(a : A) \to B$ is, as one might expect, an element of B that is typed in a context extended with a hypothesis of type A.

Rule 2.1.33 (Function introduction).

$$\frac{\Vdash A \text{ type} \qquad a : A \gg B \text{ type} \qquad a : A \gg N = N' \in B}{\Vdash \lambda a.\, N = \lambda a.\, N' \in (a : A) \to B}$$

Proof. λ-abstractions are always values, so it is enough to check that $\tau_i \vDash (a : A) \to B \downarrow R$ with $\lambda a.\, N \approx \lambda a.\, N' \in R$, which holds by the hypotheses and definition of τ_i. □

So far we have dealt only with values; now we come to some actual computation. Whereas we *introduce* a function by abstracting a variable, we use (or *eliminate*) a function by applying it to some term.

Rule 2.1.34 (Function elimination).

$$\frac{\Vdash F = F' \in (a : A) \to B \qquad \Vdash M = M' \in A}{\Vdash F\, M = F'\, M' \in B[M/a]}$$

Proof. From the assumption $\Vdash F = F' \in (a : A) \to B$ and the definition of the relation for the function type, we have that $F \Downarrow \lambda a.\, N$ and $F \Downarrow \lambda a.\, N'$ for some N, N' such that $a : A \gg N = N' \in B$. We may instantiate the latter judgment with the closing substitution $(M/a) = (M'/a) \in (a : A)$ to obtain $\Vdash N[M/a] = N'[M'/a] \in B[M/a]$. Expanding the definition, we have $B[M/a] \Downarrow V$, $N[M/a] \Downarrow W$, and $N'[M'/a] \Downarrow W'$ with $\tau_i \vDash V \downarrow R$ (for some R) and $W \approx W' \in R$.

Referring to the operational semantics for functions in Figure 2.2, we see that $F\, M \longmapsto^* (\lambda a.\, N)\, M$ and $(\lambda a.\, N)\, M \longmapsto N[M/a]$; thus $F\, M \Downarrow W$. Likewise, we have $F'\, M' \Downarrow W'$. It therefore follows from $W \approx W' \in R$ that $\Vdash F\, M = F'\, M' \in B[M/a]$. □

We can show that the operational semantics rule $(\lambda a.\, N)\, M \longmapsto N[M/a]$ gives rise to an equation in the type theory: if we define a function by abstracting a variable in a term and then apply it to some second term, this is the same as substituting the second term for the variable in the first. Such a rule that governs the reduction of an elimination form applied to an introduction form is often called a *β-rule*.

Rule 2.1.35 (Function reduction).

$$\frac{a : A \gg N \in B \qquad \Vdash M \in A}{\Vdash (\lambda a.\, N)\, M = N[M/a] \in B[M/a]}$$

Proof. By instantiating $a : A \gg N \in B$, we have $\Vdash N[M/a] \in B[M/a]$. Thus $B[M/a] \Downarrow V$ and $N[M/a] \Downarrow W$ with $\tau_i \vDash V \downarrow R$ (for some R) and $W \in R$. As $(\lambda a.\, N)\, M \longmapsto N[M/a]$, we also have $(\lambda a.\, N)\, M \Downarrow V$, and thus $\Vdash (\lambda a.\, N)\, M = N[M/a] \in B[M/a]$. □

The proof of function reduction is an instance of a general principle called *head expansion*: if $M \longmapsto^* N$ and $N = N' \in A$, then $M = N' \in A$.

Finally, we can show that any element of a function type is equal to some λ-abstraction. A rule of this kind, characterizing all elements of a type as equal to some introduction form, is often called an *η-rule*.

Rule 2.1.36 (Function uniqueness).

$$\frac{\Vdash A \text{ type} \qquad a : A \gg B \text{ type} \qquad \Vdash F \in (a : A) \to B}{\Vdash F = \lambda a.\, F\, a \in (a : A) \to B}$$

Proof. By $\Vdash F \in (a : A) \to B$, we have that $F \Downarrow \lambda a.\, N$ with $a : A \gg N \in B$. By head expansion, it follows that $\Vdash F = \lambda a.\, N \in (a : A) \to B$. By weakening and function elimination (for open terms), we thus have $a : A \gg F\, a = (\lambda a.\, N)\, a \in B$. Function reduction then gives $a : A \gg (\lambda a.\, N)\, a = N \in B$, so by transitivity $a : A \gg F\, a = N \in B$. Applying function introduction and symmetry, we get $\Vdash \lambda a.\, N = \lambda a.\, F\, a \in (a : A) \to B$. A second application of head expansion with $F \longmapsto^* \lambda a.\, N$ gives the result. □

2.1.5.2 Products

The elements of the product type $(a : A) \times B$ are pairs $\langle M, N \rangle$ where M is in A and N is in $B[M/a]$; given an element of the product type, we can project its first component with the fst operator or its second component with the snd operator. The proofs of the rules for function types are readily adapted to check the corresponding rules for product types, so we merely list the results here and leave the proofs as an exercise for the reader.

Rules 2.1.37 (Rules for products).

$$\textsc{Formation}\quad \frac{\Vdash A = A' \text{ type} \qquad a : A \gg B = B' \text{ type}}{\Vdash (a : A) \times B = (a : A') \times B' \text{ type}}$$

$$\textsc{Introduction}\quad \frac{\Vdash A \text{ type} \qquad a : A \gg B \text{ type} \qquad \Vdash M = M' \in A \qquad \Vdash N = N' \in B[M/a]}{\Vdash \langle M, N\rangle = \langle M', N'\rangle \in (a : A) \times B}$$

$$\textsc{Elimination-Fst}\quad \frac{\Vdash P = P' \in (a : A) \times B}{\Vdash \mathsf{fst}(P) = \mathsf{fst}(P') \in A} \qquad \textsc{Elimination-Snd}\quad \frac{\Vdash P = P' \in (a : A) \times B}{\Vdash \mathsf{snd}(P) = \mathsf{snd}(P') \in B[\mathsf{fst}(P)/a]}$$

$$\textsc{Reduction-Fst}\quad \frac{\Vdash M \in A}{\Vdash \mathsf{fst}(\langle M, N\rangle) = M \in A}$$

$$\textsc{Reduction-Snd}\quad \frac{\Vdash N \in B[M/a]}{\Vdash \mathsf{snd}(\langle M, N\rangle) = N \in B[M/a]} \qquad \textsc{Uniqueness}\quad \frac{\Vdash P \in (a : A) \times B}{\Vdash P = \langle \mathsf{fst}(P), \mathsf{snd}(P)\rangle \in (a : A) \times B}$$

2.1.5.3 Natural numbers

The natural numbers will be our paradigmatic example of an inductive type, a type whose elements those constructible from a specified set of operators. (Eventually, they will be one particularly trivial instance of the schema for higher inductive types introduced in Part II.) The two introduction rules provide the two ways of constructing a natural number: zero is a natural, and the successor of any natural is again a natural number.

Rules 2.1.38 (Formation and introduction for natural numbers).

$$\textsc{Formation}\quad \frac{}{\Vdash \mathsf{Nat} \text{ type}} \qquad \textsc{Introduction-Zero}\quad \frac{}{\Vdash \mathsf{zero} \in \mathsf{Nat}} \qquad \textsc{Introduction-Suc}\quad \frac{\Vdash N = N' \in \mathsf{Nat}}{\Vdash \mathsf{suc}(N) = \mathsf{suc}(N') \in \mathsf{Nat}}$$

Rather than a projection rule in the vein of function or product types, elimination from the natural numbers is accomplished by an *induction principle*: to construct a dependent function from Nat into some type family $n : \mathsf{Nat} \gg B$ type, we describe what to do in the zero and suc cases. As shown in the first rule of Rules 2.1.39 below, that data comes in the form of a term $Z \in B[\mathsf{zero}/n]$, explaining what to do with zero, and a parameterized term $n : \mathsf{Nat}, b : B \gg S \in B[\mathsf{suc}(n)/n]$, explaining what to do with $\mathsf{suc}(n)$ given a natural n and the result b of recursively applying the eliminator to n.

Rules 2.1.39 (Elimination for natural numbers).

Elimination

$$\frac{\begin{array}{c} n : \mathsf{Nat} \gg B = B' \text{ type} \\ \Vdash N = N' \in \mathsf{Nat} \qquad \Vdash Z = Z' \in B[\mathsf{zero}/n] \qquad n : \mathsf{Nat}, b : B \gg S = S' \in B[\mathsf{suc}(n)/n] \end{array}}{\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; N; Z, n.b.S) = \mathsf{elim}_{\mathsf{Nat}}(n.B'; N'; Z', n.b.S') \in B[N/n]}$$

Reduction-Zero

$$\frac{\Vdash Z \in B[\mathsf{zero}/n]}{\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; \mathsf{zero}; Z, n.b.S) = Z \in B[\mathsf{zero}/n]}$$

Reduction-Suc

$$\frac{\begin{array}{c} n : \mathsf{Nat} \gg B \text{ type} \\ \Vdash N \in \mathsf{Nat} \qquad \Vdash Z \in B[\mathsf{zero}/n] \qquad n : \mathsf{Nat}, b : B \gg S \in B[\mathsf{suc}(n)/n] \end{array}}{\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; \mathsf{suc}(N); Z, n.b.S) = S[N/n, \mathsf{elim}_{\mathsf{Nat}}(n.B; N; Z, n.b.S)/b] \in B[\mathsf{suc}(N)/n]}$$

Because of the recursive character of Nat, we need a bit of setup to prove the well-typedness of the eliminator. We begin by defining a value relation E_{Nat} whose elements are the values on which the Nat eliminator is well-behaved.

Definition 2.1.40. We define $V \approx V' \in E_{\mathsf{Nat}}$ to hold when $V \approx V' \in [\![\mathsf{Nat}]\!]$ and for every $n : \mathsf{Nat} \gg B = B'$ type, $Z = Z' \in B[\mathsf{zero}/n]$, and $n : \mathsf{Nat}, b : B \gg S = S' \in B[\mathsf{suc}(n)/n]$, we have $\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; V; Z, n.b.S) = \mathsf{elim}_{\mathsf{Nat}}(n.B'; V'; Z', n.b.S') \in B[V/n]$.

We will show that all values of Nat are contained in E_{Nat} by exploiting the definition of the value relation for Nat as the least closed under zero and suc.

Lemma 2.1.41. $\mathsf{zero} \in E_{\mathsf{Nat}}$.

Proof. For any B, B', Z, Z', S, S' as in the definition of E_{Nat}, the operational semantics tells us that $\mathsf{elim}_{\mathsf{Nat}}(n.B; \mathsf{zero}; Z, n.b.S) \longmapsto Z$ and $\mathsf{elim}_{\mathsf{Nat}}(n.B'; \mathsf{zero}; Z', n.b.S') \longmapsto Z'$. That $\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; \mathsf{zero}; Z, n.b.S) = \mathsf{elim}_{\mathsf{Nat}}(n.B'; \mathsf{zero}; Z', n.b.S') \in B[\mathsf{zero}/n]$ thus follows from the assumption $\Vdash Z = Z' \in B[\mathsf{zero}/n]$ by head expansion on either side. □

Lemma 2.1.42. If $N \approx N' \in \Downarrow E_{\mathsf{Nat}}$, then $\mathsf{suc}(N) \approx \mathsf{suc}(N') \in E_{\mathsf{Nat}}$.

Proof. For any B, B', Z, Z', S, S' as in the definition of E_{Nat}, the operational semantics gives us the following reductions.

$$\mathsf{elim}_{\mathsf{Nat}}(n.B; \mathsf{suc}(N); Z, n.b.S) \longmapsto S[N/n, \mathsf{elim}_{\mathsf{Nat}}(n.B; N; Z, n.b.S)/b]$$
$$\mathsf{elim}_{\mathsf{Nat}}(n.B'; \mathsf{suc}(N); Z', n.b.S') \longmapsto S'[N'/n, \mathsf{elim}_{\mathsf{Nat}}(n.B'; N'; Z', n.b.S')/b]$$

Expanding $N \approx N' \in \Downarrow E_{\mathsf{Nat}}$, we have $N \Downarrow V$ and $N' \Downarrow V'$ with $\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; V; Z, n.b.S) = \mathsf{elim}_{\mathsf{Nat}}(n.B'; V'; Z', n.b.S') \in B[V/n]$. By head expansion on either side, we see that $\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; N; Z, n.b.S) = \mathsf{elim}_{\mathsf{Nat}}(n.B'; N'; Z', n.b.S') \in B[V/n]$. We also have $\Vdash N = V \in \mathsf{Nat}$ by head expansion, so it follows from exact coercion that these two terms are also equal as elements of $B[N/n]$.

Instantiating the hypothesis $n : \mathsf{Nat}, b : B \gg S = S' \in B[\mathsf{suc}(n)/n]$ with $\Vdash N = N' \in \mathsf{Nat}$ (which follows from $N \approx N' \in \Downarrow E_{\mathsf{Nat}}$) and the above, we see that

$$S[N/n, \mathsf{elim}_{\mathsf{Nat}}(n.B; N; Z, n.b.S)/b] = S'[N'/n, \mathsf{elim}_{\mathsf{Nat}}(n.B'; N'; Z', n.b.S')/b]$$

in $B[\mathsf{suc}(N)/n]$. Now our desired equation follows by head expansion on either side. □

These two lemmas give us what we need to prove Rules 2.1.39.

Proof (of Rules 2.1.39). By Lemmas 2.1.41 and 2.1.42, E_{Nat} is closed under the clauses inductively defining $[\![\mathsf{Nat}]\!]$, so we conclude that $[\![\mathsf{Nat}]\!] \subseteq E_{\mathsf{Nat}}$.

Suppose we have $\Vdash N = N' \in \mathsf{Nat}$ along with the other hypotheses of the elimination rule; then $N \Downarrow V$ and $N' \Downarrow V'$ with $V \approx V' \in [\![\mathsf{Nat}]\!]$, and in particular $V \approx V' \in E_{\mathsf{Nat}}$. By definition of E_{Nat}, this means $\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; V; Z, n.b.S) = \mathsf{elim}_{\mathsf{Nat}}(n.B'; V'; Z', n.b.S') \in B[V/n]$. Applying head expansion on either side and using the equation $\Vdash N = V \in \mathsf{Nat}$ in the type, we obtain the conclusion of the elimination rule: $\Vdash \mathsf{elim}_{\mathsf{Nat}}(n.B; N; Z, n.b.S) = \mathsf{elim}_{\mathsf{Nat}}(n.B'; N'; Z', n.b.S') \in B[N/n]$.

The reduction rules now follow immediately from head expansion. □

2.1.5.4 Identity types

Although part of our goal in this thesis is to *replace* Martin-Löf's identity types with cubical path types, it will nevertheless be useful to have a working understanding of the former. The identity type is an example of an *indexed inductive type*, a type generated by constructors that inhabit different indices of a family of types. It, too, will be an instance of our schema for higher inductive types in Part II.

In particular, the identity type at type A is indexed by two terms $M_0, M_1 \in A$; the elements of the instance $\mathsf{Id}(A, M_0, M_1)$ are proofs that M_0 and M_1 are equal. The sole constructor for the identity type, called refl for "reflexivity", establishes that the reflexive identity types $\mathsf{Id}(A, M, M)$ are inhabited; by exact coercion, this implies that the types $\mathsf{Id}(A, M_0, M_1)$ are moreover inhabited whenever $M_0 = M_1 \in A$.

Rules 2.1.43 (Formation and introduction for identity types).

Formation

$$\frac{\Vdash A = A' \ \mathsf{type} \qquad \Vdash M_0 = M_0' \in A \qquad \Vdash M_1 = M_1' \in A}{\Vdash \mathsf{Id}(A, M_0, M_1) = \mathsf{Id}(A', M_0', M_1') \ \mathsf{type}}$$

Introduction

$$\frac{\Vdash A \ \mathsf{type} \qquad \Vdash M = M' \in A}{\Vdash \mathsf{refl}(M) = \mathsf{refl}(M') \in \mathsf{Id}(A, M, M')}$$

One elimination principle for identity types, historically known as the "J rule" after Martin-Löf, expresses the inductive generation of the family of identity types by the refl constructor. When we have a property $a_0 : A, a_1 : A, p : \mathsf{Id}(A, a_0, a_1) \gg B$ type dependent on pairs of terms and identities between them, it suffices to prove it in the case that the identity is refl.

Rules 2.1.44 (Elimination for identity types).

Elimination

$$\frac{\begin{array}{c} a_0 : A, a_1 : A, p : \mathsf{Id}(A, a_0, a_1) \gg B = B' \ \mathsf{type} \qquad \Vdash M_0 \in A \\ \Vdash M_1 \in A \qquad \Vdash P = P' \in \mathsf{Id}(A, M_0, M_1) \qquad a : A \gg N = N' \in B[a/a_0, a/a_1, \mathsf{refl}(a)/p] \end{array}}{\Vdash \mathsf{elim}_{\mathsf{Id}}(a_0.a_1.p.B, P, a.N) = \mathsf{elim}_{\mathsf{Id}}(a_0.a_1.p.B', P', a.N') \in B[M_0/a_0, M_1/a_1, P/p]}$$

Reduction

$$\frac{\begin{array}{c} a_0 : A, a_1 : A, p : \mathsf{Id}(A, a_0, a_1) \gg B \ \mathsf{type} \\ \Vdash M \in A \qquad a : A \gg N \in B[a/a_0, a/a_1, \mathsf{refl}(a)/p] \end{array}}{\Vdash \mathsf{elim}_{\mathsf{Id}}(a_0.a_1.p.B, \mathsf{refl}(M), a.N) = N[M/a] \in B[M/a_0, M/a_1, \mathsf{refl}(M)/p]}$$

Although the J rule captures the inductive generation of the identity family by the refl constructor, it is actually a fairly weak principle. In particular, it does not suffice to show that proofs of identities are unique: that for any $P, Q \in \mathsf{Id}(A, M_0, M_1)$, there exists some $T \in \mathsf{Id}(\mathsf{Id}(A, M_0, M_1), P, Q)$. This principle is nonetheless true in the computational semantics—indeed, we have $\mathsf{refl}(P) \in \mathsf{Id}(\mathsf{Id}(A, M_0, M_1), P, Q)$. The semantics validates the much stronger *equality reflection* rule, which turns elements of identity types into judgmental equalities.

Rule 2.1.45 (Equality reflection).

$$\frac{\Vdash M_0 \in A \qquad \Vdash M_1 \in A \qquad \Vdash P \in \mathsf{Id}(A, M_0, M_1)}{\Vdash M_0 = M_1 \in A}$$

The equality reflection rule presents difficulties from the perspective of formalism design; a formalism that includes equality reflection (such as Nuprl [Con+86]) necessarily has an undecidable equality judgment, which precludes techniques for automatically checking the truth of judgments. This motivates the restriction to the J rule in formalisms, possibly supported by the uniqueness of identity proofs principle (UIP). As a side effect, the fact that the J rule is compatible with non-unique identity proofs motivated early work in higher-dimensional type theory and models thereof, as we will see below.

2.1.5.5 The unit and empty types

We have included also a type with one element, Unit, and a type with no elements, Void. These are somewhat degenerate: the unit type needs no elimination rule, because its single element carries no interesting information, while the empty type needs no introduction rule, because there is nothing to introduce. The empty type is useful in particular for expressing falsehoods: we can inhabit $A \to \mathsf{Void}$ if A is empty, *i.e.*, if A regarded as a theorem is false. The elimination rule for Void says that we can construct an element of *any* type if we have an element of Void.

Rules 2.1.46 (Unit type).

$$\textsc{Formation}\quad \frac{}{\Vdash \mathsf{Unit}\ \mathsf{type}} \qquad \textsc{Introduction}\quad \frac{}{\Vdash \star \in \mathsf{Unit}}$$

Rules 2.1.47 (Empty type).

$$\textsc{Formation}\quad \frac{}{\Vdash \mathsf{Void}\ \mathsf{type}} \qquad \textsc{Elimination}\quad \frac{\Vdash A\ \mathsf{type} \qquad \Vdash M \in \mathsf{Void}}{\Vdash \mathsf{abort} \in A}$$

2.1.5.6 Universe

Finally, our type theory τ_1 includes a universe of (so-called "small") types. The typical rules for a universe are fairly simple: any element of the universe is a type, and the universe is closed under the same operators as τ_0.

Rules 2.1.48 (Universes).

$$\frac{}{\Vdash \mathsf{U}\ \mathsf{type}} \qquad \frac{\Vdash A = A' \in \mathsf{U}}{\Vdash A = A'\ \mathsf{type}} \qquad \frac{\Vdash A = A' \in \mathsf{U} \qquad a : A \gg B = B' \in \mathsf{U}}{\Vdash (a : A) \to B = (a : A') \to B' \in \mathsf{U}} \qquad \dots$$

The benefit of having a universe is that we can write definitions and prove theorems that quantify over types within the theory. For example, we can define composition of functions between arbitrary types:

$$\lambda A, B, C, g, f, a.\, g\,(f\,a) \in (A, B, C : \mathsf{U}) \to (B \to C) \to (A \to B) \to (A \to C)$$

(For concision, we group iterated abstractions, such as $(A : \mathsf{U}) \to (B : \mathsf{U}) \to \cdots$ and $\lambda A.\, \lambda B. \cdots$ in the above, as comma-separated lists.)

2.2 Formalisms

If a type theory is a definition of truth, a formalism is a *window onto truth*; in computer science terms, an interface. The concerns of formalism design are much the same as those of interface design. On the one hand, there is *usability*: is the formalism expressive enough to prove the kind of results we want to prove, and is it structured to make them easy to prove? On the other hand, there is *range of applicability*: is the interface generic enough to be used as a window on a wide variety of notions of truth (*i.e.*, implementations)? Formalisms are particulary relevant to type theories because they can form the basis of proof assistants, programs that help a user develop proofs and check their correctness.

Algebraic theories Our type theories are built on top of an untyped programming language, the terms of which are the subjects of the typing judgments. This is sensible if we want to ground our truth in computation, but for an interface we would like to abstract away those details; this way we can realize the interface with implementations that compute differently, do not compute, or are not syntactic at all (such as set-theoretic interpretations).

Various techniques therefore exist to ensure that a formalism does not make reference to so-called "raw terms". For example, the substitution operation $-[M/a]$ we have used in our computational type theories is an operator on raw terms; in our formalisms, we avoid it by using *explicit substitutions* [ACCL91], term formers that are internal to the type theory (of the same status as, say, $\mathsf{suc}(-)$) rather than external operations.

These techniques have the additional useful effect of making our formalisms instances of the class of *generalized algebraic theories* (GATs) [Car86]. All GATs satisfy certain generic results; for example, the collection of interpretations of a given GAT can be organized into a category (the *category of models*) with an initial object given by the so-called *syntactic model*. These results are tremendously useful for establishing key properties of the formalism such as normalization, as is well-demonstrated by a recent explosion of research (*e.g.*, [Shu15; Coq19; CHS19; KHS19; SAG19; SA21]). Proving any such properties is beyond the scope of this thesis, but we aim with our novel formalism in Part III to create a setting amenable to such approaches.

We make one concession to readability by not completely annotating terms. For our formalism to be a GAT, the terms should be annotated with enough information to recover the derivation of their well-formedness; for example, the function application $F\,N$ should be annotated with the domain and codomain types of F. It is mechanical enough for a reader to deduce what annotations should be present in a completely formal presentation, so we will suppress them here.

In the following, we describe the skeleton of a formalism for our small type theory, highlighting the considerations that drive formalism design (as opposed to the design of computational models).

2.2.1 Intensional type theory

Judgment	Presuppositions	Reading
Γ ctx		Γ is a context
$\Gamma' \vdash \gamma : \Gamma$	$(\Gamma, \Gamma'$ ctx$)$	γ is a substitution from Γ to Δ
$\Gamma' \vdash \gamma = \gamma' : \Gamma$	$(\Gamma' \vdash \gamma, \gamma' : \Gamma)$	γ and γ' are equal substitutions
$\Gamma \vdash A$ type	$(\Gamma$ ctx$)$	A is a type in context Γ
$\Gamma \vdash A = A'$ type	$(\Gamma \vdash A, A'$ type$)$	A and A' are equal types in context Γ
$\Gamma \vdash M : A$	$(\Gamma \vdash A$ type$)$	M is a term of type A in context Γ
$\Gamma \vdash M = M' : A$	$(\Gamma \vdash M, M' : A)$	M and M' are equal terms

Figure 2.3: Judgments of the **ITT** formalism

Judgments and presuppositions Like a type theory, a formalism is based on a collection of judgments delineating the well-formed and equal types and elements. For **ITT**, we have the judgments shown in Figure 2.3. We use $\vdash$ and $:$ for entailment and elementhood in formal judgments, reserving $\gg$ and $\in$ for computational judgments. Whereas a computational type theory *defines* the judgments, a formalism merely provides an interface in the form of a collection of rules.

To simplify the presentation of rules, we attach to each formal judgment a collection of *presuppositions*, assumptions under which it makes sense to state a judgment. For example, the judgment $\Gamma \vdash M : A$ presupposes that Γ is a context and A is a type; only under those circumstances does it make sense to ask whether M is an element of A supposing Γ. This allows us to omit hypotheses like Γ ctx and $\Gamma \vdash A$ type from rules when it is clear the rule would not make sense otherwise. Unlike the PER-based computational interpretation, in which we define the unary judgment forms from the binary, here we require both sides of an equation to be well-formed before we can state it.

Explicit substitutions We eliminate the dependence on raw term substitution by making substitution application an operation *inside* the type theory, taking an open substitution $\Gamma' \vdash \gamma : \Gamma$ as an argument.

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

The substitutions form a category: there is always an identity substitution, and we can compose substitutions.

$$\frac{}{\Gamma \vdash \mathsf{id} : \Gamma} \qquad\qquad \frac{\Gamma'' \vdash \gamma' : \Gamma' \qquad \Gamma' \vdash \gamma : \Gamma}{\Gamma'' \vdash \gamma \circ \gamma' : \Gamma}$$

These are subject to the usual equations: $\mathsf{id} \circ \gamma = \gamma$, $\gamma \circ \mathsf{id} = \gamma$, and $\gamma \circ (\gamma' \circ \gamma'') = (\gamma \circ \gamma') \circ \gamma''$. We moreover have equations for computing the action of a substitution.

$$\frac{}{A[\mathsf{id}] = A \ \mathsf{type}} \qquad\qquad \frac{\Gamma'' \vdash \gamma' : \Gamma' \qquad \Gamma' \vdash \gamma : \Gamma \qquad \Gamma \vdash A \ \mathsf{type}}{\Gamma \vdash A[\gamma \circ \gamma'] = A[\gamma][\gamma'] \ \mathsf{type}}$$

When we arrive at the function type below, we will introduce further equations that compute substitutions at each term and type former; for example, function application will come with an equation $(F\,M)[\gamma] = F[\gamma]\,M[\gamma]$.

Hypotheses and variables Again for the purpose of simplifying metatheoretic analysis, we avoid introducing named variables. Thus, a context is not a lookup table associating names with types, but merely a list of types.

$$\frac{}{\cdot \ \mathsf{ctx}} \qquad\qquad \frac{\Gamma \vdash A \ \mathsf{type}}{\Gamma.A \ \mathsf{ctx}}$$

An extended context $\Gamma.A$ comes with a weakening substitution (written p for "projection") that throws away the assumption. On the other hand, we can construct a substitution from some Γ' into an extended context $\Gamma.A$ by taking a substitution $\Gamma' \vdash \gamma : \Gamma$ and attaching an additional term $\Gamma' \vdash M : A[\gamma]$.

$$\frac{\Gamma \ \mathsf{ctx} \qquad \Gamma \vdash A \ \mathsf{type}}{\Gamma.A \vdash \mathsf{p} : \Gamma} \qquad\qquad \frac{\Gamma' \vdash \gamma : \Gamma \qquad \Gamma \vdash A \ \mathsf{type} \qquad \Gamma' \vdash M : A[\gamma]}{\Gamma' \vdash \gamma.M : \Gamma.A}$$

In an extended context $\Gamma.A$, we always have access to at least one variable, namely the one of type A at the top of the the context; we write v for this variable. (Note that we weaken A so that it is well-formed in context $\Gamma.A$.)

$$\frac{\Gamma \vdash A \ \mathsf{type}}{\Gamma.A \vdash \mathsf{v} : A[\mathsf{p}]}$$

In a context like $\Gamma.C.B.A$, variables further back in the context are accessible by way of the projection substitution: we have $\Gamma.C.B.A \vdash \mathsf{v} : A[\mathsf{p}]$, $\Gamma.C.B.A \vdash \mathsf{v}[\mathsf{p}] : B[\mathsf{p} \circ \mathsf{p}]$, and $\Gamma.C.B.A \vdash \mathsf{v}[\mathsf{p} \circ \mathsf{p}] : C[\mathsf{p} \circ \mathsf{p} \circ \mathsf{p}]$. (Henceforth we write p^2, p^3, *etc.* for such iterated projections.) When we apply a substitution $\Gamma' \vdash \gamma.M : \Gamma.A$ into an extended context, the top variable v is instantiated with M by way of the first equation below, while other variables search deeper in the context via the second.

$$\frac{\Gamma' \vdash \gamma : \Gamma \qquad \Gamma \vdash A\ \mathsf{type} \qquad \Gamma' \vdash M : A[\gamma]}{\Gamma' \vdash \mathsf{v}[\gamma.M] = M : A[\gamma]}$$

$$\frac{\Gamma' \vdash \gamma : \Gamma \qquad \Gamma \vdash A\ \mathsf{type} \qquad \Gamma' \vdash M : A[\gamma]}{\Gamma' \vdash \mathsf{p} \circ (\gamma.M) = \gamma : \Gamma}$$

Note that the substitution pairing operator $-.-$ and the two projections p and v behave much like the constructor $\langle -, - \rangle$ and projections fst and snd of the dependent product type; intuitively, the extended context $\Gamma.A$ is the dependent product of Γ and A. The uniqueness rule for products also has a counterpart.

$$\frac{\Gamma' \vdash \gamma : \Gamma.A \qquad \Gamma \vdash A\ \mathsf{type}}{\Gamma' \vdash \gamma = (\mathsf{p} \circ \gamma).\mathsf{v}[\gamma] : \Gamma.A}$$

The mechanical simplicity of nameless variables does unfortunately come at the cost of some readability, so we might be forgiven for using names and leaving the translation to p's and v's to the reader. We nevertheless stick to a nameless presentation, as such a translation becomes less evident in the presence of the new context formers (bridge interval extension and restriction, modalities) we introduce in Parts III and IV.

Function types Now that we have seen how to deal with variables in an algebraic fashion, the rules for function types contain no surprises. (Since there is no need to include a variable binding, we write $A \to B$ here even for dependent function types.)

$$\frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma.A \vdash B\ \mathsf{type}}{\Gamma \vdash A \to B\ \mathsf{type}} \qquad \frac{\Gamma.A \vdash N : B}{\Gamma \vdash \lambda(N) : A \to B}$$

$$\frac{\Gamma.A \vdash B\ \mathsf{type} \qquad \Gamma \vdash F : A \to B \qquad \Gamma \vdash M : A}{\Gamma \vdash F\,M : B[\mathsf{id}.M]} \qquad \frac{\Gamma.A \vdash N : B \qquad \Gamma \vdash M : A}{\Gamma \vdash \lambda(N)\,M = N[\mathsf{id}.M] : B[\mathsf{id}.M]}$$

$$\frac{\Gamma \vdash F : A \to B}{\Gamma \vdash F = \lambda(F[\mathsf{p}]\,\mathsf{v}) : A \to B}$$

We leave off congruence rules—if $A = A'$ type and $B = B'$ type then $A \to B = A' \to B'$ type and so on—as these can be mechanically inferred.

To give equations for the calculation of substitutions in function types, we first observe that context extension has a functorial action on substitutions: given $\Gamma' \vdash \gamma : \Gamma$ and a type $\Gamma \vdash A$ type, we have $\Gamma'.A[\gamma] \vdash \gamma^\times : \Gamma.A$ defined by $\gamma^\times \coloneqq (\gamma \circ \mathsf{p}).\mathsf{v}$. Using this, we can propagate substitutions beneath binders as follows.

$$\begin{aligned}(A \to B)[\gamma] &= A[\gamma] \to B[\gamma^\times] \\ (\lambda(M))[\gamma] &= \lambda(M[\gamma^\times]) \\ (F\,M)[\gamma] &= F[\gamma]\,M[\gamma]\end{aligned}$$

Remember, this is not a definition of substitution by clauses; this is a specification of equations that an interpretation of substitution should satisfy. Of course, if we do not provide sufficient equations, the formalism will be poorly behaved, but this does not mean the formalism is incompletely defined, only unsatisfactory.

We leave the formulation of rules for the other types to the reader; one may simply mimic the suite of rules developed in Section 2.1.

Interpretation in computational type theories Once we have laid out a formalism, we can ask whether a given computational type theory is an instance of the interface it presents.

To start with, we need an interpretation $|-|$ of formal contexts, substitutions, types, and terms as untyped syntax (or operations, in the case of substitutions).[1] Given this, there is a canonical candidate interpretation for a formalism with the judgments given above in the kind of computational type theory we have described: we interpret $\Gamma \vdash A$ type as $|\Gamma| \gg |A|$ type, $\Gamma \vdash M : A$ as $|\Gamma| \gg |M| \in |A|$, and so on. The interpretation is sound if the interpretation of every rule in the formalism is a true principle in the interpretation. We will not go through the work of proving such a theorem here, but the process is fairly straightforward: we have already done the bulk of the work by proving rules for each of the type formers in Section 2.1.

Adequacy One key property we can ask of a formalism for a computational interpretation is *computational adequacy*, the property that the reductions of the interpretation's operational semantics are tracked by the equational theory of the formalism.

Proposition 2.2.1 (Adequacy). If $\cdot \vdash M : A$ and $|M| \longmapsto |N|$, then $\cdot \vdash M = N : A$.

[1] As we are translating from a nameless to named representation of variables, this function should really be parameterized by a variable environment.

This property is indeed satisfied by ITT, reflecting intuitively that it includes the necessary rules for calculating the results of eliminators and substitutions at each type. Adequacy expresses a kind of constructive character of a formalism: for example, if we construct a term of natural number type, it can be "run" to obtain an explicit natural number. Note that a formalism that is constructive in this sense may still have non-constructive models: adequacy only shows that elements *definable in the formalism* can be computed.

2.2.2 A non-computational model

As mentioned above, properties like adequacy of a formalism do not prevent us from interpreting that formalism in non-constructive settings. As an example, we sketch a set-theoretic interpretation of the formalism described above.

We begin by interpreting contexts Γ as sets $[\![\Gamma]\!] \in \mathit{Set}$ and substitutions $\Gamma' \vdash \gamma : \Gamma$ as set-theoretic functions, $[\![\gamma]\!] : [\![\Gamma']\!] \to [\![\Gamma]\!]$. We interpret $\Gamma \vdash A$ type as a family of sets $([\![A]\!]_I)_{I \in [\![\Gamma]\!]}$, and terms $\Gamma \vdash M : A$ as families of elements: $([\![M]\!]_I)_{I \in [\![\Gamma]\!]}$ where $[\![M]\!]_I \in [\![A]\!]_I$ for each $I \in [\![\Gamma]\!]$. The equality judgments are interpreted by set-theoretic equality. Application of substitution is interpreted by reindexing of families: given $\Gamma' \vdash \gamma : \Gamma$ and $\Gamma \vdash A$ type, we define $[\![A[\gamma]]\!]_I \coloneqq [\![A]\!]_{[\![\gamma]\!](I)}$. We interpret the empty context by a one-element set, $[\![\,\cdot\,]\!] \coloneqq \{\star\}$, and context extension by disjoint union (*i.e.*, coproduct) over the elements of the base context: $[\![\Gamma.A]\!] \coloneqq \coprod_{I \in [\![\Gamma]\!]} [\![A]\!]_I$.

Moving on to type formers, we can interpret dependent function types as products, $[\![A \to B]\!]_I \coloneqq \prod_{a \in [\![A]\!]_I} [\![B]\!]_{(I,a)}$, and dependent products as disjoint unions $[\![A \times B]\!]_I \coloneqq \coprod_{a \in [\![A]\!]_I} [\![B]\!]_{(I,a)}$. The identity type $\mathsf{Id}(A, M_0, M_1)$ can be interpreted as a one-element set when M_0 and M_1 are equal and the empty set otherwise.

$$[\![\mathsf{Id}(A, M_0, M_1)]\!]_I \coloneqq \left\{\star \mid [\![M_0]\!]_I = [\![M_1]\!]_I\right\}$$

We may interpret the universe by assuming that our set theory supports a *Grothendieck universe*, essentially a set large enough to be closed under the various type formers.

Chapter 3

Cubical type theory

Cubical type theory enhances Martin-Löf's type theory with a contentful notion of equality, the *path*. It will be the basic substrate with which we work for the remainder of this thesis; Parts II to IV each describe extensions of cubical type theory. We can separate the history of cubical type theory into several distinct if frequently interacting strands: the history of higher-dimensional models of intensional type theory, the history of higher-dimensional formalisms, and the history of constructivity and computation for the two.

Higher-dimensional models The earliest higher-dimensional model of **ITT** is Hofmann and Streicher's *groupoid interpretation* [HS98]. By higher-dimensional model, we mean one that refutes the uniqueness of identity proofs (UIP) principle. Said principle states that any pair of proofs of identity are themselves identical.

$$\frac{A\ \mathsf{type} \qquad M, N \in A \qquad P, Q \in \mathsf{Id}(A, M, N)}{\mathsf{uip} \in \mathsf{Id}(\mathsf{Id}(A, M, N), P, Q)}$$

Hofmann and Streicher's model was indeed designed for the purpose of refuting this principle, showing it independent of **ITT**. They interpret types as *groupoids*, categories in which every morphism is invertible. The identity type between elements a and b of a groupoid G is then the set (*i.e.*, discrete groupoid) of morphisms between them, which may contain multiple elements. The model thus has *one* level of higher structure: there can be distinct proofs of identity between two objects, but any two proofs of identities between identities are necessarily identical.

Awodey and Warren [AW09] picked up this thread by establishing a connection between **ITT**'s identity type and the concept of a *weak factorization system* from homotopy theory. In his dissertation, Warren shows that one can construct n-dimensional models of **ITT** for every n—each refuting an n-dimensional version of UIP—as well as an infinite-dimensional model in *strict ω-groupoids* that refutes all such principles. Van den Berg and

Garner generalized Awodey and Warren's results to construct a larger class of higher-dimensional models of **ITT**, including the infinite-dimensional *topological spaces* and *simplicial sets* [BG12]. Beyond proving independence results for UIP-like principles, these models opened the possibility of using **ITT** as a language for proving theorems about such higher-dimensional settings.

Formalisms Around the same time, Gambino and Garner established a result in the opposite direction: the syntactic category of **ITT**—the category with contexts as objects and substitutions as morphisms—contains a non-trivial weak factorization system [GG08]. That is, this structure can not only appear in models of **ITT**, but is visible in the formalism itself. Van den Berg and Garner further showed that that every syntactic type has the structure of a *weak ω-groupoid*, with operations such as composition and inverse of identities defined using the J rule [BG11].

Separately, Voevodsky became interested in **ITT** as a tool for formalizing results from homotopy theory. He proposed extending **ITT** with an axiom, which he dubbed the *univalence axiom* [Voe14], that identifies identities between types in a universe with isomorphisms. Voevodsky developed a model of **ITT** with the univalence axiom in simplicial sets, which has been since been described by Kapulkin and Lumsdaine [KL12a]. It is perhaps worth noting that Voevodsky was not primarily interested in using higher-dimensional theory to study homotopy theory synthetically in the style of, *e.g.*, [Uni13, §8]. Rather, he was interested in formalizing zero-dimensional ("set-level") mathematics and saw the univalence axiom as indispensable for this purpose. His program, which he referred to as the Univalent Foundations, is embodied in the ongoing **UniMath** project [VAG+20].

The Special Year on Univalent Foundations of Mathematics at the Princeton Institute for Advanced Study culminated in the publication of *Homotopy Type Theory: Univalent Foundations of Mathematics* [Uni13], the "**HoTT** Book", produced collaboratively by the participating researchers. The **HoTT** Book explores the consequences of the univalence axiom in **ITT**, with a particular eye towards synthetic homotopy theory: using the higher-dimensional elements of **ITT** as a language to study higher-dimensional mathematics. For this purpose, the **HoTT** Book also assumed a (not-so-precisely delineated) collection of *higher inductive types*. Conceived at a 2011 workshop in Oberwolfach by Bauer, Lumsdaine, Shulman, and Warren, higher inductive types generalize inductive types to enable the direct construction of higher-dimensional spaces such as n-dimensional spheres. With these in hand, it became possible to replicate and formalize results from classical homotopy, such as the calculation of the fundamental group of the circle [LS13] and the 4th homotopy group of the 3-sphere [Bru16]. We use the name *homotopy type theory* (**HoTT**) for the particular extension of **ITT** with the univalence axiom and higher inductive types defined in the Book, although it is sometimes used as a name for the field more generally.

However, the introduction of all these axioms in **ITT** destroyed the constructive char-

acter of the theory. The axioms provide no way to evaluate a term $\cdot \vdash N : \mathsf{Nat}$ that uses the univalence axiom or higher inductive types, so canonicity fails in **HoTT**. Indeed, it was not even known whether homotopy theory had a constructive *model*, that is, whether it was possible to interpret the formalism without relying on classical principles; Bezem, Coquand, and Parmann showed that Voevodsky's simplicial model relied in an essential way on non-constructivity [BCP15]. Despite this dismal state of affairs, computer scientists were drawn to the promise of homotopy type theory: if it *could* be made computational, it had the potential to resolve several unsatisfactory aspects of **ITT**. The univalence axiom implies *function extensionality*, the principle that functions are identified when they are pointwise identified, which is infuriatingly unprovable using only the J rule. Univalence itself, of course, gives a satisfying characterization of equality in the universe, which **ITT** fails to say anything about on its own. Higher inductive types, meanwhile, would provide effective quotients.

Constructivity and computation In pursuit of a computational interpretation of—or computational replacement for—**HoTT**, Licata and Harper [LH11; LH12] defined a formalism called *two-dimensional type theory* (**2DTT**), which with our numbering conventions would be *one*-dimensional type theory: like the groupoid model, it permits distinct identities between elements but identifies all identities between identities. Their theory contains a single univalent universe. By including rules for calculating with the paths produced by univalence, Licata and Harper were able to prove canonicity for this theory. The principle underlying the design of **2DTT** is one that is ubiquitous in the study of type theories: *types internalize judgmental structure*. That is, if there is to be a type whose elements are contentful paths, then there should be a judgmental notion capturing the concept of contentful path. In **2DTT**, this judgment takes the form $\Gamma \vdash \alpha : M \simeq_A N$, read "$\alpha$ is a path from M to N in type A".

The next leap in the struggle for computation came from Bezem, Coquand, and Huber [BCH13], who built the first constructively-definable model of **ITT** with univalence. Their model, known as the BCH model, replaces the simplicial sets of Voevodsky's model with *affine cubical sets*. To give a very rough idea, where simplicial sets describe higher-dimensional objects as being built out of n-dimensional triangles (the eponymous simplices), cubical sets describe higher-dimensional objects as assemblies of n-dimensional cubes. For reasons too technical to get into here, this change of setting neatly sidesteps the constructivity issues with the simplicial model. Unfortunately, the BCH model presented problems for interpreting higher inductive types; indeed, it is still unknown whether any but the most trivial of higher inductive types can be interpreted in the BCH model.

The solution, it turned out, was to replace *affine* cubical sets with *structural* cubical sets. This adjustment was pursued in parallel by Cohen, Coquand, Huber, and Mörtberg (CCHM) [CCHM15] and Angiuli, Favonia, and Harper (AFH) [AFH18], using different

variations on structural cubical sets. In both cases, the result was not merely a constructive model but a *cubical type theory*. In the CCHM case, this came in the form of a formalism with an interpretation in cubical sets and a canonicity result due to Huber [Hub19]; in the AFH case, in the form of a type theory in our computational sense. Each presented a theory with a full-fledged, infinite hierarchy of univalent universes, along with examples of higher inductive types. Apart from the difference in settings (formal vs. computational), the CCHM and AFH approaches are broadly similar, but do differ substantially in the finer details. In technical terms, CCHM is based on the *De Morgan cube category*, while AFH is based on the *cartesian cube category*; that basic difference leads to divergences in the ways coercions are calculated at each type. The CCHM model is generalized to give a variety of topos models for De Morgan cubical type theory in [OP18; LOPS18]; the same is done for cartesian type theory in [ABCFHL19]. Cavallo, Mörtberg, and Swan show that the two branches can be viewed as instances of a single construction [CMS20]. As in **2DTT**, the basic move in each of these theories is to treat contentful identities as internalizations of a judgmental phenomenon. in the cubical case, that structure is dependency on an interval variable.

Outline We begin in Section 3.1 with a framework for cartesian cubical type theories in the style of Angiuli, Favonia, and Harper, hewing most closely to the account presented in Angiuli's dissertation [Ang19]. With an instance of the framework in hand, we exercise it a bit in Section 3.2, proving some elementary results in cubical type theory that will serve us later on. As with Martin-Löf type theory, we round out the chapter with a discussion of formalisms and non-computational models in Section 3.3.

3.1 Cubical computational type theory

To start, let us lay out the structure we intend cubical type theories to support. The distinguishing characteristic of all cubical type theories is the ability to assume an interval variable.

$$\frac{\Gamma \ \mathsf{ctx}}{(\Gamma, x : \mathbb{I}) \ \mathsf{ctx}}$$

Although this notationally resembles an ordinary term hypothesis $a : A$, it is really a separate context forming operation; the interval is not a type. Interval *terms*, which can be substituted for such variables, are characterized by separate judgments $\Gamma \gg r \in \mathbb{I}$ and $\Gamma \gg r = s \in \mathbb{I}$, the resemblance to $\Gamma \gg M \in A$ and $\Gamma \gg M = N \in A$ again being merely suggestive. The intuition is that $\mathbb{I}$ is an interval in the sense of topology, a space with two

points (the *endpoints*) and a line between them. We name these two endpoints "0" and "1".

$$\frac{}{0 \in \mathbb{I}} \qquad\qquad \frac{}{1 \in \mathbb{I}}$$

A path is then a type or term that depends on an interval variable: $x : \mathbb{I} \gg A$ type is a path of types, and $x : \mathbb{I} \gg M \in A$ is a path of terms. The endpoints of a path are recovered by substituting the constants 0 and 1: $x : \mathbb{I} \gg A$ type is a path from $A[0/x]$ type to $A[1/x]$ type. (Note that as far as substitution is concerned, interval variables behave exactly like ordinary term variables.)

The judgmental concept of path can then be straightforwardly internalized by *path types*: for each $x : \mathbb{I} \gg A$ type, and pair of endpoint terms $M_0 \in A[0/x]$ and $M_1 \in A[1/x]$, we introduce a type $\mathsf{Path}(x.A, M_0, M_1)$ type whose values are abstracted terms $\lambda^{\mathbb{I}} x.\, M$ such that $x : \mathbb{I} \gg M \in A$, $M[0/x] = M_0 \in A[0/x]$, and $M[1/x] = M_1 \in A[1/x]$. This type will behave much like a function type, albeit one with constraints on its values at 0 and 1. In general, it is like a *dependent* function type: elements of $\mathsf{Path}(x.A, M_0, M_1)$ are paths over the "path of types" $x : \mathbb{I} \gg A$ type.

Notation 3.1.1. When A type does not depend on x, we abbreviate $\mathsf{Path}(x.A, M_0, M_1)$ as $\mathsf{Path}(A, M_0, M_1)$.

In a type theory with interval variables, each type comes equipped with a contentful relation: for any $M_0 \in A$ and $M_1 \in A$, the collection of paths $x : \mathbb{I} \gg M \in A$ such that $M[0/x] = M_0 \in A$ and $M[1/x] = M_1 \in A$ can be thought of as a collection of witnesses that M_0 and M_1 are related. Note that this relation is reflexive by way of constant functions: given any $M \in A$, we have $\lambda\_.\, M \in \mathsf{Path}(A, M, M)$. In order for this contentful relation to be a notion of *equality*, however, more structure is required. For one, nothing here implies that the path relation is symmetric or transitive. More fundamentally, there is no way to *transport* along these paths, to transfer results about a given term to any path-equal term.

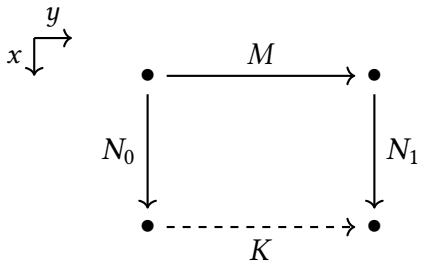
Coercion and composition The second essential component of cubical type theory is thus a pair of operations called the *Kan operations*, so called in reference to the Kan condition in classical homotopy theory [Kan55]. The first of these, coercion, implements the transport of terms along paths of types.

$$\frac{x : \mathbb{I} \gg A \text{ type} \qquad r \in \mathbb{I} \qquad s \in \mathbb{I} \qquad M \in A[r/x]}{\mathsf{coe}^{r \to s}_{x.A}(M) \in A[s/x]}$$

That is, if $x : \mathbb{I} \gg A$ type is a path of types and we have an element of $A[r/x]$, then we can transform it into an element of $A[s/x]$ for any other s.

The second Kan operation, *homogeneous composition* (hcom), serves a more technical purpose. Note that the existence of coercion at all type lines implies that paths are transitive and symmetric. For example, given $P \in \mathsf{Path}(A, M_0, M_1)$, we can compute its inverse as $\mathsf{coe}^{0\to 1}_{x.\mathsf{Path}(A,P\,x,M_0)}(\lambda^{\mathbb{I}} x.\, M_0) \in \mathsf{Path}(A, M_1, M_0)$. Turning this around, however, we will find that we need paths to be symmetric and transitive *in order to implement* coercion at all types, at path types in particular. In other words, we must strengthen our induction hypothesis. Homogeneous composition provides these symmetry and transitivity operations; more precisely, it is a *box-filling* operation that includes the two as special cases.

As an example of box-filling, consider the following situation: we have a path $y : \mathbb{I} \gg M \in A$ together with two additional paths $x : \mathbb{I} \gg N_0 \in A$ and $x : \mathbb{I} \gg N_1 \in A$ that extend from M's endpoints, *i.e.*, satisfy $M[0/y] = N_0[0/x] \in A$ and $M[1/y] = N_1[0/x] \in A$. We can picture these as forming the "open box" shown below, a square with one missing side.



The homogeneous composition of these terms is the dotted line: a path $y : \mathbb{I} \gg K \in A$ such that $K[0/y] = N_0[1/x] \in A$ and $K[1/y] = N_1[1/x] \in A$. In syntax, this path is written as follows.

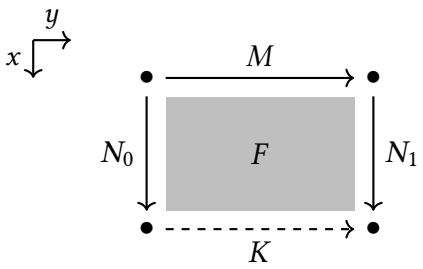
$$y : \mathbb{I} \gg K \coloneqq \mathsf{hcom}^{0\to 1}_{A}(M; y \equiv 0 \hookrightarrow x.N_0, y \equiv 1 \hookrightarrow x.N_1) \in A$$

We can think of K as a composite of three paths: first the inverse of $x.N_0$, then $y.M$, then $x.N_1$. In particular, symmetry and transitivity are special cases. If we instantiate $y.M$ and $x.N_1$ with reflexive paths, then $y.K$ is the inverse of $x.N_0$; if we instantiate $x.N_0$ with a reflexive path, then $y.K$ is the composite of $y.M$ with $x.N_1$.

The general form of homogeneous composition replaces $0 \to 1$ with $r \to s$, allowing us to take a "horizontal" term $y.M$ at any point on the x-axis and move it to any other point. In particular, if we allow the destination point to vary in the example above, we can obtain an *interior* (or *filler*) for the open box.

$$x : \mathbb{I}, y : \mathbb{I} \gg F \coloneqq \mathsf{hcom}^{0\to x}_{A}(M; y \equiv 0 \hookrightarrow x.N_0, y \equiv 1 \hookrightarrow x.N_1) \in A$$

This two-dimensional term will satisfy the equations $x : \mathbb{I} \gg F[0/y] = N_0 \in A$, $x : \mathbb{I} \gg F[1/y] = N_1 \in A$, $y : \mathbb{I} \gg F[0/x] = M \in A$, and $y : \mathbb{I} \gg F[1/x] = K \in A$, thereby filling in the open box as shown below.



The general hcom also replaces the pair $(x \equiv 0 \hookrightarrow y.N_0, x \equiv 1 \hookrightarrow y.N_1)$ with an arbitrary collection of equation-path pairs, which may involve any number of interval variables. These are required to agree on their overlaps; if, for example, they include $y \equiv 0 \hookrightarrow x.N_0$ and $z \equiv 0 \hookrightarrow x.P_0$, then it must be the case that $x : \mathbb{I} \gg N_0[0/z] = P_0[0/y] \in A$. In effect, such a collection forms a frame into which a term may fit, which we call the *tube* of the composite. An hcom takes a term (M in our example) that fits into the tube at one index r, which we call the *cap*, and produces a term that fits into the tube at any other index s. In particular, the picture above generalizes to n-dimensional open box filling: given an n-dimensional cube with one face missing, we can use hcom to produce a term that fits in the missing face.

Computing with cubical terms Of course, everything we have said so far is merely aspirational; we need to set up a computational setting in which these dreams come true. The main obstruction to that dream is the evaluation of coercion, terms of the form $\mathsf{coe}^{r\to s}_{x.A}(M)$. Let us recall some intuition: a line of types $x : \mathbb{I} \gg A$ type is supposed to correspond to an isomorphism, with $\mathsf{coe}^{0\to 1}_{x.A}$ and $\mathsf{coe}^{1\to 0}_{x.A}$ executing the forward and backward functions of the isomorphism respectively. In particular, the evaluation of $\mathsf{coe}^{r\to s}_{x.A}(M)$ *depends on the form of* $x.A$: we have to look at A to determine what isomorphism it represents. If, for example, we have a path $P \in \mathsf{Path}(\mathsf{U}, \mathsf{Bool}, \mathsf{Bool})$, it could represent either the identity isomorphism $\mathsf{Bool} \simeq \mathsf{Bool}$ or the isomorphism that swaps tt and ff. In the former case, we should have $\mathsf{coe}^{0\to 1}_{x.P\,x}(\mathsf{tt}) = \mathsf{tt} \in \mathsf{Bool}$; in the latter, we should have $\mathsf{coe}^{0\to 1}_{x.P\,x}(\mathsf{tt}) = \mathsf{ff} \in \mathsf{Bool}$.

The upshot of this situation is that we must be able to compute in a context with interval variables: we need to evaluate $x.A$ to examine it. This is a stark change of pace from Chapter 2, where our operational semantics only applied to *closed* terms. Fortunately, we do not need to be able to evaluate *all* open terms, just those that depend only on interval terms. For this chapter, those terms will be our "closed" terms.

When we evaluate terms containing interval variables, there is a question of *coherence*: how does substitution for interval variables interact with evaluation? Suppose, for example, that we have some $x : \mathbb{I} \gg M \in A$. Then M should evaluate to some value:

$M \Downarrow V$. On the other hand, we can substitute 0 for x to obtain $M[0/x] \in A[0/x]$ and then evaluate: $M[0/x] \Downarrow V_0$. What should the relationship between $V[0/x]$ and V_0 be?

If we assume that the typing judgments are stable under substitution and that terms are equal to their values—which we would certainly like to be true—then we will have that $x : \mathbb{I} \gg M = V \in A$, thus $M[0/x] = V[0/x] \in A[0/x]$, as well as $M[0/x] = V_0 \in A[0/x]$. It follows that $V[0/x] = V_0 \in A[0/x]$. Flipping our perspective around, if we want stability under substitution and equality to values, we must ensure that our definition of the term judgment guarantees these kind of coherence equations. It is too permissive to say that $x : \mathbb{I} \gg M \in A$ whenever M evaluates to a value in A; we must require that all the substitution instances of M evaluate in a coherent way. The two ideas of *evaluation in an interval context* and *coherent evaluation* form the basis of the computational interpretation of cartesian cubical type theory as presented by Angiuli, Favonia, and Harper, as well as the proof of canonicity for De Morgan cubical type theory due to Huber.

3.1.1 Interval contexts

Now getting into the definition of the framework proper, we first want to distinguish the contexts and substitutions that deal only with interval assumptions; the contexts in which we consider terms to be "closed". We use the letters Ψ and ψ for these contexts and substitutions, distinguishing them from the general Γ and γ. The interval judgments are prior to the definitions that deal with terms; in particular, they do not depend at all on the choice of type system.

Definition 3.1.2 (Contexts). The well-formed interval contexts, Ψ ictx, are inductively defined by the following rules.

$$\frac{}{\cdot \text{ ictx}} \qquad \frac{\Psi \text{ ictx}}{(\Psi, x : \mathbb{I}) \text{ ictx}}$$

Definition 3.1.3 (Interval elements). $\Psi \Vdash r \in \mathbb{I}$ holds when $r = 0$, $r = 1$, or $r = x$ for some $(x : \mathbb{I}) \in \Psi$.

Definition 3.1.4 (Interval substitutions). The well-formed interval substitutions, $\Psi' \Vdash \psi \in \Psi$, are inductively defined by the following rules.

$$\frac{}{\Psi' \Vdash \cdot \in \cdot} \qquad \frac{\Psi' \Vdash \psi \in \Psi \qquad \Psi' \Vdash r \in \mathbb{I}}{\Psi' \Vdash (\psi, r/x) \in (\Psi, x : \mathbb{I})}$$

3.1.2 Operational semantics and type systems

The two defining components of a cubical type theory—its operational semantics and its type system—must take an ambient interval context into account. For the operational semantics, this just means that the judgments operate on terms that may contain interval variables.

Definition 3.1.5. An *operational semantics* is a definition of two judgments M val and $M \longmapsto N$ operating on terms that contain only interval variables, satisfying the following properties.

- *Determinism:* If $M \longmapsto N$ and $M \longmapsto N'$, then $N = N'$. For any M, it is not the case that both M val and $M \longmapsto N$ for some N.
- *Variable preservation:* If $M \longmapsto N$, then the free interval variables in N are a subset of the free variables in M.

Given an operational semantics, we define the induced multi-step judgment $M \longmapsto^* N$ and evaluation judgment $M \Downarrow V$ as in Definition 2.1.1.

Notably, we do *not* require that the operational semantics is stable under interval substitution. That is, we do not ask that M val implies $M\psi$ val or that $M \longmapsto N$ implies $M\psi \longmapsto N\psi$ for every $\Psi' \Vdash \psi \in \Psi$. Indeed, the operational semantics will contain several rules that fail to be stable in this way. This kind of stability *will* be enforced on the level of typed equality judgments, but not at the level of untyped operational semantics.

On the type system side, the data that defines a type A in an interval context Ψ will now consist of a family of relations indexed by substitutions into Ψ, specifying the values of $A\psi$ for each possible $\Psi' \Vdash \psi \in \Psi$.

Definition 3.1.6 (Ψ-relations). Given Ψ ictx, a *Ψ-relation* R is a family of relations $R\langle\psi\rangle$ indexed by substitutions $\Psi' \Vdash \psi \in \Psi$ from arbitrary interval contexts Ψ' into Ψ. A Ψ-relation is a *Ψ-PER* when each $R\langle\psi\rangle$ is a PER. Given a Ψ-relation R and substitution $\Psi' \Vdash \psi \in \Psi$, we define the Ψ'-relation $R\psi$ by $R\psi\langle\psi'\rangle := R\langle\psi\psi'\rangle$.

Notation 3.1.7. When R is a Ψ-relation and M and M' are terms in context Ψ, we will write $M \approx M' \in R$ as syntactic sugar for $M \approx M' \in R\langle \mathrm{id}_\Psi \rangle$. This permits us to write $M \approx M' \in R\psi$ in place of $M \approx M' \in R\langle\psi\rangle$.

Note that we do not require Ψ-relations to be stable under substitution in general: we do not ask that $M \approx M' \in R$ implies $M\psi \approx M'\psi \in R\psi$. Indeed, we are primarily interested in Ψ-relations on *values*, and it may not even be the case that $V\psi$ is a value for V val.

With each value type assigned a Ψ-relation of values, we extend the value relation to terms by *coherent extension*. As described above, we want to require that the interval substitution instances of a term in a type evaluate in a coherent way.

Definition 3.1.8 (Incoherent evaluation). We generalize the evaluation operator $\Downarrow -$ from relations (Definition 2.1.8) to Ψ-relations pointwise: $M \approx M' \in (\Downarrow R)\langle\psi\rangle$ holds when $M \Downarrow V$ and $M' \Downarrow V'$ with $V \approx V' \in R\langle\psi\rangle$.

Definition 3.1.9 (Coherent evaluation). Given a Ψ-relation R, we define its *coherent extension to terms* $\Downarrow\!\!\Downarrow R$ as follows. Let $\Psi' \Vdash \psi \in \Psi$ be given. Then $M \approx M' \in (\Downarrow\!\!\Downarrow R)\langle\psi\rangle$ holds when for every subsequent pair of substitutions, $\Psi_1 \Vdash \psi_1 \in \Psi'$ and $\Psi_2 \Vdash \psi_2 \in \Psi_1$, the following conditions are satisfied.

- $M\psi_1 \Downarrow V$ and $M'\psi_1 \Downarrow V'$ for some values V and V'.
- $N \approx N' \in \Downarrow\!\!\Downarrow R\psi\psi_1\psi_2$ for $N \in \{M\psi_1\psi_2, V\psi_2\}$ and $N' \in \{M'\psi_1\psi_2, V'\psi_2\}$.

Proposition 3.1.10. If R is a Ψ-PER, then $\Downarrow\!\!\Downarrow R$ is a Ψ-PER.

We will ultimately use $\Downarrow\!\!\Downarrow -$ to define the typing judgments induced by a type system. The third condition imposes the "coherence"; it requires that the following square of substitutions and evaluations commutes up to the equality defined by $\Downarrow\!\!\Downarrow R$.

$$\begin{array}{ccc} M\psi_1 & \Longrightarrow & V \\ {\scriptstyle -\psi_2}\downarrow & & \downarrow{\scriptstyle -\psi_2} \\ M\psi_1\psi_2 & \Longrightarrow & \bullet \end{array}$$

The outer quantification over ψ_1 ensures that $\Downarrow\!\!\Downarrow R$ is always stable under interval substitution; if $M \approx M' \in \Downarrow\!\!\Downarrow R$, then $M\psi \approx M'\psi \in \Downarrow\!\!\Downarrow R\psi$ for any ψ. In order for a value Ψ-relation R to be suitable as the interpretation of a type, it must satisfy an additional well-formedness condition called *value-coherence*, which asks that all values related by R are in fact coherently related by R.

Definition 3.1.11. A Ψ-relation R on values is *value-coherent* when $R \subseteq \Downarrow\!\!\Downarrow R$.

As with ordinary relations, the collection of Ψ-relations on a given field is a lattice, and so we can obtain fixed-points of monotone operators on Ψ-relations. To check that these fixed-points are PERs, we again use operators Sym^+ and $Trans^+$ defined in Definition 2.1.21, which we extend pointwise to Ψ-relations.

Lemma 3.1.12. Let F be a monotone operator on Ψ-relations. If we have $F(Sym^+(\mu F)) \subseteq Sym^+(\mu F)$, then μF is symmetric. If $F(Trans^+(\mu F)) \subseteq Trans^+(\mu F)$, then μF is transitive.

Proposition 3.1.13. We have $\Downarrow\!\!\Downarrow Sym^+(R) \subseteq Sym^+(\Downarrow\!\!\Downarrow R)$ and $\Downarrow\!\!\Downarrow Trans^+(R) \subseteq Trans^+(\Downarrow\!\!\Downarrow R)$.

We likewise parameterize type systems by an interval context, separately specifying the value types available at each Ψ.

Definition 3.1.14. A *candidate type system* is a four-place relation τ relating interval contexts Ψ, values V and V' with free variables contained in Ψ, and value-coherent Ψ-PERs R.

Notation 3.1.15. Given a candidate type system τ, we write $\tau \vDash \Psi \Vdash V \approx V' \downarrow R$ as syntactic sugar for $(\Psi, V, V', R) \in \tau$, and $\tau \vDash \Psi \Vdash V \downarrow R$ for $(\Psi, V, V, R) \in \tau$. Given a Ψ-PER R, we write $\tau[R]$ for the Ψ-relation $V \approx V' \in \tau[R]\langle\psi\rangle :\Longleftrightarrow \tau \vDash \Psi' \Vdash V \approx V' \downarrow R\psi$.

Definition 3.1.16. A candidate type system is a *type system* when it satisfies the following additional axioms.

- *PER:* For any fixed Ψ-PER R, $\tau[R]$ is a Ψ-PER.
- *Unicity:* If $\tau \vDash \Psi \Vdash V \approx V' \downarrow R$ and $\tau \vDash \Psi \Vdash V \approx V' \downarrow R'$, then $R = R'$.
- *Value-coherence:* For any fixed R, $\tau[R]$ is value-coherent.

We have analogues of the operators $Uni^+(-)$, $Sym^+(-)$, and $Trans^+(-)$ from Definition 2.1.25 defined pointwise in the context Ψ. Defining a similar operator for value-coherence, we can derive a condition analogous to Lemma 2.1.26 for checking that a candidate is a type system.

Definition 3.1.17. For any candidate τ, we define a candidate type system $Coh^+(\tau)$ as follows: $Coh^+(\tau) \vDash \Psi \Vdash V \approx V' \downarrow R$ holds when $V \approx V' \in \Downarrow\tau[R]$ holds.

Proposition 3.1.18. Let F be a monotone operator on candidate type systems such that $F(Sym^+(\mu F)) \subseteq Sym^+(\mu F)$, $F(Trans^+(\mu F)) \subseteq Trans^+(\mu F)$, $F(Uni^+(\mu F)) \subseteq Uni^+(\mu F)$, and $F(Coh^+(\mu F)) \subseteq Coh^+(\mu F)$. Then μF is a type system.

3.1.3 Pretypes

Going forward, we assume a candidate type system τ and begin defining the judgments of type theory. We are not quite ready to define types; we still need to cut down to relations that support the coercion and composition operations. As it will usually take some work to prove that each type supports those operations, however, it is useful to introduce some intermediate notation. We therefore introduce a preliminary judgment, the *pretype* judgment, that does not require Kan operations. We can also define the element judgment at this stage; we do not need to know that a pretype supports the Kan operations to define what its elements are.

As in MLTT, the first step is to define the closed judgments, where "closed" now means dependent only on an interval context, extending the type system from values to all closed terms. As suggested above, we replace the use of evaluation in MLTT with coherent evaluation.

Definition 3.1.19 (Closed pretype judgments).

- *Pretypes:* $\Psi \Vdash A = A'$ pretype holds when $A \approx A' \in \Downarrow\tau[R]$ for some Ψ-PER R.
- *Terms:* $\Psi \Vdash M = M' \in A$ holds when $A \in \Downarrow\tau[R]$ for some Ψ-PER R such that $M \approx M' \in \Downarrow R$.

As always, $\Psi \Vdash A$ pretype and $\Psi \Vdash M \in A$ are shorthand for $\Psi \Vdash A = A$ pretype and $\Psi \Vdash M = M \in A$ respectively.

Next, we come to open judgments. At this point, we also want to introduce a notion of *constraints*, equations $\xi = (r \equiv s)$ on interval terms. These will come in handy when we specify homogeneous composition, which involves terms that are only well-typed when some such equation is assumed. To express well-typedness under the assumption of constraints, we allow constraints to appear in contexts: if Γ is a context and ξ is a constraint well-formed over Γ, then Γ, ξ is also a context. Before we define the open type and term judgments, we first define the open interval and constraint judgments.

Definition 3.1.20 (Open interval judgments). The judgment $\Gamma \gg r \in \mathbb{I}$ is defined to hold when $r = 0$, $r = 1$, or $r = x$ for some $(x : \mathbb{I})$ occurring in Γ. The equality judgment $\Gamma \gg r = s \in \mathbb{I}$ is defined to hold when $\Gamma \gg r, s \in \mathbb{I}$ and r and s are in the equivalence relation generated by the constraints appearing in Γ.

Definition 3.1.21 (Open constraint judgments). The judgments $\Gamma \gg \xi = \xi' \in \mathbb{F}$ and $\Gamma \gg \xi$ satisfied are inductively defined by the following rules.

$$\frac{\Gamma \gg r = r' \in \mathbb{I} \qquad \Gamma \gg s = s' \in \mathbb{I}}{\Gamma \gg (r \equiv s) = (r' \equiv s') \in \mathbb{F}} \qquad \frac{\Gamma \gg r = s \in \mathbb{I}}{\Gamma \gg (r \equiv s) \text{ satisfied}}$$

Note that we define the interval and constraint judgments independently of the term hypotheses in the context. In particular, an inconsistent term hypothesis does not cause interval terms to be equated: we will not have $a : \mathsf{Void} \gg 0 = 1 \in \mathbb{I}$.

To define the open typing judgments, we again introduce an auxiliary notion of *closing substitution*. Like pretypes and terms, these substitutions now take place relative to an interval context. Aside from that wrinkle, however, the definitions are precisely the same.

Definition 3.1.22 (Closing substitutions). The judgment $\Psi \Vdash \gamma = \gamma' \in \Gamma$ is generated by the following rules.

$$\frac{}{\Psi \Vdash \cdot = \cdot \in \cdot} \qquad \frac{\Psi \Vdash \gamma = \gamma' \in \Gamma \qquad \Psi \Vdash r \in \mathbb{I}}{\Psi \Vdash (\gamma, r/x) = (\gamma', r/x) \in (\Gamma, x : \mathbb{I})}$$

$$\frac{\Psi \Vdash \gamma = \gamma' \in \Gamma \qquad \Psi \Vdash \xi\gamma \text{ satisfied}}{\Psi \Vdash \gamma = \gamma' \in (\Gamma, \xi)} \qquad \frac{\Psi \Vdash \gamma = \gamma' \in \Gamma \qquad \Psi \Vdash M = M' \in A\gamma}{\Psi \Vdash (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma, a : A)}$$

Definition 3.1.23 (Open pretype and element judgments).

- *Open pretypes:* $\Gamma \gg A = A'$ pretype is defined to hold when $\Psi \Vdash A\gamma = A'\gamma'$ pretype holds for all $\Psi \Vdash \gamma = \gamma' \in \Gamma$.
- *Open terms:* $\Gamma \gg M = M' \in A$ is defined to hold when $\Psi \Vdash M\gamma = M\gamma' \in A\gamma$ holds for all $\Psi \Vdash \gamma = \gamma' \in \Gamma$.

Definition 3.1.24 (Contexts). The judgment $\Gamma = \Gamma'$ ctx is generated by the following rules.

$$\frac{}{\cdot = \cdot \text{ ctx}} \qquad \frac{\Gamma = \Gamma' \text{ ctx}}{(\Gamma, x : \mathbb{I}) = (\Gamma', x : \mathbb{I}) \text{ ctx}} \qquad \frac{\Gamma = \Gamma' \text{ ctx} \qquad \Gamma \gg \xi = \xi' \in \mathbb{F}}{(\Gamma, \xi) = (\Gamma', \xi') \text{ ctx}}$$

$$\frac{\Gamma = \Gamma' \text{ ctx} \qquad \Gamma \gg A = A' \text{ pretype}}{(\Gamma, a : A) = (\Gamma', a : A') \text{ ctx}}$$

We derive the unary versions of these judgments from their binary versions in the usual way. Although the notation $\Psi' \Vdash \psi \in \Psi$ is now *a priori* ambiguous—it could refer to an interval substitution or closing substitution—the two conflated judgments coincide. Moreover, because the closed judgments are stable under interval substitution, we have $\Psi \gg A$ pretype if and only if $\Psi \Vdash A$ pretype and so on.

It will be useful in the future to extend the notion of substitution-indexed relations from substitutions into interval contexts to closing substitutions into arbitrary contexts.

Definition 3.1.25 (Open relations). Given a Ψ-PER R on lists of terms, an *R-relation* S is a family of relations $S\langle\gamma\rangle$ indexed by lists $\gamma \in R\langle\psi\rangle$ with the property that $S\langle\gamma\rangle = S\langle\gamma'\rangle$ whenever $\gamma \approx \gamma' \in R\langle\psi\rangle$. We extend $\Downarrow -$ to R-relations in the obvious way, replacing the initial interval substitution $\Psi' \Vdash \psi \in \Psi$ in Definition 3.1.9 with an instantiation $\gamma \in R\langle\psi\rangle$.

In a type system τ with $\tau \vDash \Gamma$ ctx, a *Γ-relation* is a $[\![\Gamma]\!]^\tau$-relation, where $[\![\Gamma]\!]^\tau$ is the $\cdot$-PER defined by $\gamma \approx \gamma' \in [\![\Gamma]\!]^\tau\langle\Psi\rangle$ if and only if $\Psi \Vdash \gamma = \gamma' \in \Gamma$. Given a Γ-relation R and $\Gamma' \gg \gamma \in \Gamma$, we write $R\gamma$ for the Γ'-relation defined by $R\gamma\langle\gamma'\rangle := R\langle\gamma\gamma'\rangle$. Given a

Γ-relation R and terms M, M', we write $\Gamma \gg M \approx M' \in R$ to mean that $M\gamma \approx M'\gamma' \in R\langle\gamma\rangle$ holds for all $\Psi \Vdash \gamma \in \Gamma$.

Note that while the definitions of Ψ- and R-relations are dependent only on the theory of interval substitutions, the definition of Γ-relation is of course relative to a type system.

3.1.4 Kan operations and types

Finally, we define the conditions under which a pretype becomes a type: when it supports the coercion and homogeneous composition operations.

Definition 3.1.26 (Coercion). We say that a Ψ-relation R *supports coercion at* A, A' when it validates the following rules for every $\Psi', x : \mathbb{I} \Vdash \psi \in \Psi$ and $\Psi' \Vdash r, s \in \mathbb{I}$.

$$\frac{M \approx M' \in \Downarrow R[\psi, r/x]}{\mathrm{coe}^{r\to s}_{x.A\psi}(M) \approx \mathrm{coe}^{r\to s}_{x.A'\psi}(M') \in \Downarrow R[\psi, s/x]} \qquad \frac{M \in \Downarrow R[\psi, r/x]}{\mathrm{coe}^{r\to r}_{x.A\psi}(M) \approx M \in \Downarrow R[\psi, r/x]}$$

That is, a relation R supports coercion at A, A' when we can coerce along any substitution instance $R\psi$ that forms a line of types in some direction x; moreover, we require that the trivial coercion $r \to r$ is equal to the identity function. We say that $\Psi \Vdash A = A'$ pretype support coercion when $[\![A]\!]$ (equivalently, $[\![A']\!]$) supports coercion at A, A'.

Definition 3.1.27 (Homogeneous composition). We say that a Ψ-relation R *supports homogeneous composition at* A, A' when it validates the following rules for every $\Psi' \Vdash \psi \in \Psi$, interval terms $\Psi' \Vdash r, s \in \mathbb{I}$, and list of constraints $\Psi' \Vdash \xi_i \in \mathbb{F}$ for $0 \le i < n$.

$$\frac{\begin{array}{c} M \approx M' \in R\psi \\ (\forall i, j)\ \Psi', \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N'_j \in \Downarrow R\psi \qquad (\forall i)\ \Psi', \xi_i \gg M \approx N_i[r/x] \in \Downarrow R\psi \end{array}}{\mathsf{hcom}^{r\to s}_{A\psi}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx \mathsf{hcom}^{r\to s}_{A'\psi}(M'; \overrightarrow{\xi_i \hookrightarrow x.N'_i}) \in \Downarrow R\psi}$$

$$\frac{\begin{array}{c} M \in \Downarrow R\psi \\ (\forall i, j)\ \Psi', \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N_j \in \Downarrow R\psi \qquad (\forall i)\ \Psi', \xi_i \gg M \approx N_i[r/x] \in \Downarrow R\psi \end{array}}{\mathsf{hcom}^{r\to r}_{A\psi}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx M \in \Downarrow R\psi}$$

$$\frac{\begin{array}{c} \Psi' \Vdash \xi_k \text{ satisfied} \qquad M \in \Downarrow R\psi \\ (\forall i, j)\ \Psi', \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N_j \in \Downarrow R\psi \qquad (\forall i)\ \Psi', \xi_i \gg M \approx N_i[r/x] \in \Downarrow R\psi \end{array}}{\mathsf{hcom}^{r\to s}_{A\psi}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx N_k[s/x] \in \Downarrow R\psi}$$

We use the notation $\overrightarrow{\cdots}$ to denote lists (here of entries $\xi_i \hookrightarrow x.N_i$), leaving quantification over the indexing variable (here i) implicit for sake of concision. In short, R has homogeneous composition when we can compose in any instance $R\psi$; the result must fit into the tube $\overrightarrow{\xi_i \hookrightarrow x.N_i}$ instantiated at s, and the trivial composition $r \to r$ is required to be the identity. In order for a tube to be well-formed, its entries must agree where their equations overlap; this is effected by the requirement $\Psi', \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N_j \in R\psi$.

We say that $\Psi \Vdash A = A'$ pretype support composition when $[\![A]\!]$ supports composition at A, A'.

Remark 3.1.28. As above, we use the word *trivial* to describe coercions and composites $r \to r$; *degenerate*, one the other hand, refers to paths of the form $\lambda^{\mathbb{I}}\_.\, M$.

Definition 3.1.29 (Kan types).

- *Closed types:* $\Psi \Vdash A = A'$ type is defined to hold when $\Psi \Vdash A = A'$ pretype support coercion and homogeneous composition.
- *Open types:* $\Gamma \gg A = A'$ type is defined to hold when $\Psi \Vdash A\gamma = A'\gamma'$ type holds for all $\Psi \Vdash \gamma = \gamma' \in \Gamma$.

Like the pretype judgment, the closed type judgment is stable under interval substitution by construction: if $\Psi' \Vdash \psi \in \Psi$ and $\Psi \Vdash A = A'$ type, then $\Psi' \Vdash A\psi = A'\psi$ type.

This completes the derivation of the type-theoretic judgments from an operational semantics and type system. We leave the definition of $\Gamma' \gg \gamma = \gamma' \in \Gamma$ to the reader, this being simple to extrapolate from Definition 2.1.18 and the definition of closing substitutions above.

3.1.5 Constructing a cubical type theory

We now upgrade our examples of type systems to include cubical elements. To the syntax described in Section 2.1, we add the following terms for path types, V *types* (to be introduced below), and the Kan operations.

$$\begin{aligned} A, B, M, N, I \quad ::= \quad & \cdots \\ & \mid \mathsf{Path}(x.A, M, N) \mid \lambda^{\mathbb{I}} x.\, M \mid P\, r \\ & \mid \mathsf{V}_r(A, B, I) \mid \mathsf{v}_r(M, N) \mid \mathsf{vproj}_r(M, I) \\ & \mid \mathsf{coe}^{r \to s}_{x.A}(M) \mid \mathsf{hcom}^{r \to s}_{A}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \end{aligned}$$

We give the operational semantics for formation, introduction, and elimination of path and V types in Figure 3.1. In addition to these, we must also describe the evaluation

Paths

$$\frac{}{\mathsf{Path}(x.A, M, N)\ \mathsf{val}} \qquad \frac{}{\lambda^{\mathbb{I}} x.\, M\ \mathsf{val}} \qquad \frac{P \longmapsto P'}{P\, r \longmapsto P'\, r} \qquad \frac{}{(\lambda^{\mathbb{I}} x.\, M)\, r \longmapsto P[r/x]}$$

$\vee$ ***types***

$$\frac{}{\mathsf{V}_x(A_0, A_1, I)\ \mathsf{val}} \qquad \frac{}{\mathsf{V}_\varepsilon(A_0, A_1, I) \longmapsto A_\varepsilon} \qquad \frac{}{\mathsf{v}_x(M_0, M_1)\ \mathsf{val}} \qquad \frac{}{\mathsf{v}_\varepsilon(M_0, M_1) \longmapsto M_\varepsilon}$$

$$\frac{M \longmapsto M'}{\mathsf{vproj}_x(M, I) \longmapsto \mathsf{vproj}_x(M', I)} \qquad \frac{}{\mathsf{vproj}_x(\mathsf{v}_x(M, N), I) \longmapsto N}$$

$$\frac{}{\mathsf{vproj}_0(M, I) \longmapsto \mathsf{fst}(I)\, M} \qquad \frac{}{\mathsf{vproj}_1(N, I) \longmapsto N}$$

Figure 3.1: Additional operational semantics for cubical type theory

Generic

$$\frac{A \longmapsto A'}{\mathsf{coe}^{r \to s}_{x.A}(M) \longmapsto \mathsf{coe}^{r \to s}_{x.A'}(M)} \qquad \frac{A \longmapsto A'}{\mathsf{hcom}^{r \to s}_{A}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto \mathsf{hcom}^{r \to s}_{A'}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})}$$

Functions

$$\frac{}{\mathsf{coe}^{r \to s}_{x.(a:A) \to B}(F) \longmapsto \lambda a.\, \mathsf{coe}^{r \to s}_{x.B[\mathsf{coe}^{s \to x}_{x.A}(a)/a]}(F\, (\mathsf{coe}^{s \to r}_{x.A}(a)))}$$

$$\frac{}{\mathsf{hcom}^{r \to s}_{(a:A) \to B}(F; \overrightarrow{\xi_i \hookrightarrow x.G_i}) \longmapsto \lambda a.\, \mathsf{hcom}^{r \to s}_{B}(F\, a; \overrightarrow{\xi_i \hookrightarrow x.G_i\, a})}$$

Paths

$$\frac{}{\mathsf{coe}^{r \to s}_{x.\mathsf{Path}(y.A, M_0, M_1)}(P) \longmapsto \lambda^{\mathbb{I}} y.\, \mathsf{com}^{r \to s}_{x.A}(P; y \equiv 0 \hookrightarrow x.M_0, y \equiv 1 \hookrightarrow x.M_1)}$$

$$\frac{}{\begin{array}{c}\mathsf{hcom}^{r \to s}_{\mathsf{Path}(y.A, M_0, M_1)}(P; \overrightarrow{\xi_i \hookrightarrow x.Q_i}) \\ \longmapsto \\ \lambda^{\mathbb{I}} y.\, \mathsf{hcom}^{r \to s}_{A}(P\, y; \overrightarrow{\xi_i \hookrightarrow x.Q_i\, y}, y \equiv 0 \hookrightarrow \_.M_0, y \equiv 1 \hookrightarrow \_.M_1)\end{array}}$$

Figure 3.2: Selected rules for coercion and homogeneous composition

of coe and hcom at each value type. We give two examples—function and path types—in Figure 3.2. In each case, coercion and composition at the compound type reduce to coercion and composition at the component types. To coerce a function along the type line $x.(A \to B)$ from r to s, for example, we precompose with a reversed coercion $\mathsf{coe}^{s \to r}_{x.A}$ in the domain type and then postcompose with a coercion $\mathsf{coe}^{r \to s}_{x.B}$ in the codomain type, thus transforming a function of type $A[r/x] \to B[r/x]$ into one of type $A[s/x] \to B[s/x]$.

The evaluation of coercion for paths, meanwhile, relies on *heterogeneous composition*, a compound operation derived from coercion and homogeneous composition. Heterogeneous composition combines the functionality of the two Kan operations, coercing the base term M along a type line $x.A$ while maintaining a tube of paths $\overrightarrow{\xi_i \hookrightarrow x.N_i}$ lying over said line of types.

Definition 3.1.30. We define the heterogeneous composition operator, com, as follows.

$$\mathsf{com}^{r \to s}_{x.A}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \coloneqq \mathsf{hcom}^{r \to s}_{A[s/x]}(\mathsf{coe}^{r \to s}_{x.A}(M); \overrightarrow{\xi_i \hookrightarrow x.\mathsf{coe}^{x \to s}_{x.A}(N_i)})$$

Rules 3.1.31 (Heterogeneous composition). For the following rules, we assume type lines $\Gamma, x : \mathbb{I} \gg A = A'$ type, interval terms $\Gamma \gg r = r' \in \mathbb{I}$ and $\Gamma \gg s = s' \in \mathbb{I}$, and constraints $\Gamma \gg \xi_i = \xi'_i \in \mathbb{F}$ for some $0 \leq i < n$.

$$\frac{\Gamma \gg M = M' \in A[r/x] \qquad (\forall i, j)\ \Gamma, \xi_i, \xi_j, x : \mathbb{I} \gg N_i = N'_j \in A \qquad (\forall i)\ \Gamma, \xi_i \gg M = N_i[r/x] \in A[r/x]}{\Gamma \gg \mathsf{com}^{r \to s}_{x.A}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = \mathsf{com}^{r' \to s'}_{x.A'}(M'; \overrightarrow{\xi'_i \hookrightarrow x.N'_i}) \in A[s/x]}$$

$$\frac{\Gamma \gg M \in A[r/x] \qquad (\forall i, j)\ \Gamma, \xi_i, \xi_j, x : \mathbb{I} \gg N_i = N_j \in A \qquad (\forall i)\ \Gamma, \xi_i \gg M = N_i[r/x] \in A[r/x]}{\Gamma \gg \mathsf{com}^{r \to r}_{x.A}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = M \in A[s/x]}$$

$$\frac{\Gamma \gg \xi_k \text{ satisfied} \qquad \Gamma \gg M \in A[r/x] \qquad (\forall i, j)\ \Gamma, \xi_i, \xi_j, x : \mathbb{I} \gg N_i = N_j \in A \qquad (\forall i)\ \Gamma, \xi_i \gg M = N_i[r/x] \in A[r/x]}{\Gamma \gg \mathsf{com}^{r \to s}_{x.A}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = N_k[s/x] \in A[s/x]}$$

Proof. Straightforward consequences of the defining rules for coercion and composition in types (given in Section 3.1.4). □

The definition of coercion at path types demonstrates the necessity of the composition operator. To ensure that the result of coercing has the necessary endpoints, we need an operation that maintains them. The definition of homogeneous composition at the path

type, meanwhile, motivates the general form of hcom, as we must to add new entries to the tube for the endpoints whenever we compose in a path type.

We have deliberately omitted the most complex pieces of cubical type theory: the definitions of coercion and composition in the V types and the definition of composition in the universe. While these are of course crucial to cubical type theory, they will not play an explicit role in this thesis. We therefore refer to [AFH18, Figure 4.2, Section 4.4.9, Section 4.4.11] for details.

We now define our first candidate type system. To the constructs of our Martin-Löf type theory, we add path types, V types, and composite types (which implement composition in the universe). We encourage the unfamiliar reader to ignore the specification of V types for now; we will explain them in Section 3.1.6. As we will never need the definition of composite types, we gloss over these entirely. Note that we do not include identity types in our cubical type system; as they are in Chapter 2, these will fail to support the Kan operations. Of course, it has been our intention to replace identity types with path types from the beginning. (We will, however, return to identity types in Part II.)

Example 3.1.32 (Small type system). We define an operator F on candidate type systems as follows: given τ, $F(\tau)$ is the union of the following clauses.

- $F(\tau) \vDash \Psi \Vdash (a : A) \to B \approx (a : A') \to B' \downarrow R$ whenever
  - $A \approx A' \in \Downarrow\tau[S]$ for some Ψ-PER S,
  - $B\psi[M/a] \approx B'\psi[M'/a] \in \Downarrow\tau[T_M]$ for all $M \approx M' \in R\psi$, for some S-PER T,
  - $V \approx V' \in R\langle\psi\rangle$ holds for $\Psi' \Vdash \psi \in \Psi$ exactly when $V = \lambda a.\, N$ and $V' = \lambda a.\, N'$ for some N, N' with $N[M/a] \approx N'[M'/a] \in \Downarrow T_M\psi$ for all ψ and $M \approx M' \in \Downarrow S\psi$,
- $F(\tau) \vDash \Psi \Vdash \mathsf{Path}(x.A, M_0, M_1) \approx \mathsf{Path}(x.A, M_0', M_1') \downarrow R$ whenever
  - $A \approx A' \in \Downarrow\tau[S]$ for some $(\Psi, x : \mathbb{I})$-PER S,
  - $M_\varepsilon \approx M_\varepsilon' \in \Downarrow S[\varepsilon/x]$ for $\varepsilon \in \{0, 1\}$,
  - $V \approx V' \in R\langle\psi\rangle$ holds for $\Psi' \Vdash \psi \in \Psi$ exactly when $V = \lambda^{\mathbb{I}} x.\, M$ and $V' = \lambda^{\mathbb{I}} x.\, M'$ for some M, M' with $M \approx M' \in \Downarrow S\psi$ and $M[\varepsilon/x] \approx M_\varepsilon\psi \in \Downarrow S\psi[\varepsilon/x]$ for $\varepsilon \in \{0, 1\}$.
- $F(\tau) \vDash \Psi \Vdash \mathsf{V}_r(A, B, I) \approx \mathsf{V}_r(A', B', I') \downarrow R$ whenever
  - $\Psi \Vdash r \in \mathbb{I}$,
  - $A \approx A' \in \Downarrow\tau[S]$ for some $(\Psi, r \equiv 0)$-PER S,
  - $B \approx B' \in \Downarrow\tau[T]$ for some Ψ-PER T,
  - $I \approx I' \in (S \simeq T)$, where $S \simeq T$ is the $(\Psi, r \equiv 0)$-PER that relates equal isomorphisms (Definition 1.2.1) between the elements of S and T.

- $V \approx V' \in R\langle\psi\rangle$ holds for $\Psi' \Vdash \psi \in \Psi$ exactly when one of the following holds:
  - $r\psi = 0$, and $V \approx V' \in S\psi$,
  - $r\psi = 1$, and $V \approx V' \in T\psi$,
  - $r\psi = x$, and $V = \mathsf{v}_x(M, P)$ and $V' = \mathsf{v}_x(M', P')$ with $M \approx M' \in \Downarrow S\psi$, $P \approx P' \in \Downarrow T\psi$, and $(\mathsf{fst}(I\psi))\, M \approx P \in \Downarrow T\psi$, where in the final equation we regard $T\psi$ as a $(\Psi', x \equiv 0)$-PER by weakening.

- Clauses for dependent products and composites of types. The first two are pointwise extensions of the clauses in Example 2.1.27, as with function types above. For composites of types, see [Ang19, Figure 4.3, Section 4.4.11].

We define the candidate type system τ_0 to be the least fixed point of F.

In order for F to be a genuine operator on candidate type systems, it must be the case that the Ψ-relations it assigns to each value type are actually value-coherent Ψ-PERs. This is most easily seen to be true as a corollary of the introduction rules for each type's relation, so we defer the proof for the moment. Similarly, the condition on $Coh^+(-)$ required by Proposition 3.1.18, which we need to see that the fixed-point is a type system, will be a consequence of the formation rules for each type in $F_0(\tau)$. The unicity and PER conditions, on the other hand, are as straightforward as before.

Example 3.1.33 (Type system with one universe). We can define a type system with a universe by following the recipe of Example 2.1.29. For a candidate τ, we define a candidate $U(\tau)$ by declaring that that $U(\tau) \vDash \Psi \Vdash V \approx V' \downarrow R$ holds when $V = V' = \mathsf{U}$ and R is the Ψ-relation $W \approx W' \in R\langle\psi\rangle \iff \exists S.\ \tau \vDash \Psi' \Vdash W \approx W' \downarrow S$ for $\Psi' \Vdash \psi \in \Psi$. Our candidate type system with a universe, τ_1, is then the fixed point of $\tau \mapsto F(\tau) \cup U(\tau_0)$.

3.1.6 Rules for cubical type theories

Taking τ_0 and τ_1 as our prototypical (candidate) type systems, we now build up an edifice of rules associated to each type. This is a more difficult task than for pure Martin-Löf type theory because of the demands of coherent evaluation: to show that some term belongs to a type, we have to analyze its behavior under iterated applications of interval substitution and evaluation. Fortunately, we can at least factor the results through a collection of more intuitive lemmas, so that we need not interact with the definition of $\Downarrow -$ directly.

First, the following lemma can be used to show that a pair of values in some R belongs also to $\Downarrow R$: it suffices to show that every substitution instance belongs either to R or to $\Downarrow R$. Often, while the terms themselves are values, but some substitutions cause them to become non-values already known to belong to $\Downarrow R$.

Lemma 3.1.34 (Coherent value introduction). Let R be a value Ψ-relation, and let M and M' be terms in context Ψ. If for all $\Psi' \Vdash \psi \in \Psi$, either $M\psi \approx M'\psi \in R\psi$ or $M\psi \approx M'\psi \in \Downarrow R\psi$, then $M \approx M' \in \Downarrow R$.

Proof. Let $\Psi_1 \Vdash \psi_1 \in \Psi$ and $\Psi_2 \Vdash \psi_2 \in \Psi_1$ be given. We are in one of two cases.

- $M\psi_1 \approx M'\psi_1 \in \Downarrow R\psi_1$.

  Instantiating this relation with the substitutions $\Psi_1 \Vdash \mathsf{id}_{\Psi_1} \in \Psi_1$ and $\Psi_2 \Vdash \psi_2 \in \Psi_1$, we get that $M\psi_1 \Downarrow V$ and $M'\psi_1 \Downarrow V'$ with $N \approx N' \in \Downarrow R\psi\psi_1\psi_2$ for $N \in \{M\psi_1\psi_2, V\psi_2\}$ and $N' \in \{M'\psi_1\psi_2, V'\psi_2\}$, as needed.

- $M\psi_1 \approx M'\psi_1 \in R\psi_1$.

  Then $M\psi_1$ and $M'\psi_1$ are values, so the requirement reduces to showing that $M\psi_1\psi_2 \approx M'\psi_1\psi_2 \in \Downarrow R\psi\psi_1\psi_2$. This is the case both if $M\psi_1\psi_2 \approx M'\psi_1\psi_2 \in R\psi_1\psi_2$ and if $M\psi_1\psi_2 \approx M'\psi_1\psi_2 \in \Downarrow R\psi_1\psi_2$. □

Second, we have an analogue of head expansion. Given a term M' in R, it is not necessarily the case that any M such that $M \longmapsto M'$ is equal to M' in R. If, however, every instance $M\psi$ steps to a term equal to $M'\psi$, then we can deduce an equality.

Lemma 3.1.35 (Coherent head expansion). Let R be a value Ψ-PER, and let M, M' be terms in context Ψ. If for every $\Psi' \Vdash \psi \in \Psi$, we have $M\psi \longmapsto^* M_\psi$ for some M_ψ with $M_\psi \approx M'\psi \in \Downarrow R\psi$, then $M \approx M' \in \Downarrow R$.

Proof. Let $\Psi_1 \Vdash \psi_1 \in \Psi$ and $\Psi_2 \Vdash \psi_2 \in \Psi_1$ be given. First, we have some M_1 with $M\psi_1 \longmapsto^* M_1$ and $M_{\psi_1} \approx M'\psi \in \Downarrow R\psi_1$. By instantiating the latter fact at the substitutions $\Psi_1 \Vdash \mathsf{id}_{\Psi_1} \in \Psi_1$ and $\Psi_2 \Vdash \psi_2 \in \Psi_1$, we have some V and V' such that $M\psi_1 \longmapsto^* M_1 \Downarrow V$, $M\psi_2 \Downarrow V'$, and $N \approx N' \in \Downarrow R\psi\psi_1\psi_2$ for $N \in \{M_1\psi_2, V\psi_2\}$ and $N' \in \{M'\psi_1\psi_2, V'\psi_2\}$.

Second, we have some M_2 with $M\psi_1\psi_2 \longmapsto^* M_2$ and $M_2 \approx M'\psi_1\psi_2 \in \Downarrow R\psi_1\psi_2$. This implies in particular that $M\psi_1\psi_2 \approx M'\psi_1\psi_2 \in \Downarrow R\psi_1\psi_2$. Finally, by the assumption that R is a PER, we can deduce the final necessary relation, $M\psi_1\psi_2 \approx V'\psi_2 \in \Downarrow R\psi_1\psi_2$, from the other three. □

Once we can establish that Ψ-PER R is value-coherent—something we usually use the above lemmas to prove—we can deduce that terms in $\Downarrow R$ are related to their values.

Lemma 3.1.36 (Evaluation). Let R be a value-coherent value Ψ-PER and let $M \in \Downarrow R$. Then $M \Downarrow V$ with $M \approx V \in \Downarrow R$.

Proof. Instantiating $M \in \Downarrow R$ with identity substitutions, we know that $M \Downarrow V$. Moreover, for any $\Psi' \Vdash \psi \in \Psi$, instantiating $M \in \Downarrow R$ with id_Ψ and ψ tells us in particular that

$M\psi \approx V \in \Downarrow R\psi$. Expanding $\Downarrow R\psi$, we have that $M\psi \Downarrow V_\psi$ for some V_ψ with $V_\psi \approx V \in R\psi$. By value-coherence, we then have $V_\psi \approx V \in \Downarrow R\psi$. It follows by coherent head expansion that $M \approx V \in \Downarrow R$. □

Finally, we have a lemma allowing us to analyze the behavior of "eliminator-like" terms—that evaluate some argument and then do something with its value—in terms of their behavior of values.

Definition 3.1.37 (Eager terms). We say that a term $a.N$ depending on one term variable and interval variables in Ψ is *eager* when for every $\Psi' \Vdash \psi \in \Psi$ and term M in Ψ', we have $N\psi[M/a] \Downarrow W$ if and only if there exists some V such that $M \Downarrow V$ and $N\psi[V/a] \Downarrow W$.

Lemma 3.1.38 (Elimination). Fix an ambient candidate type system satisfying the unicity and PER conditions. Let $a.N, a.N'$ be eager terms. Suppose we have $\Psi \Vdash A$ pretype and a $(\Psi, a{:}A)$-PER S. Given any sub-relation $R \subseteq [\![A]\!]$ with the property that $N\psi[V/a] \approx N'\psi[V'/a] \in \Downarrow S\langle \psi, V/a\rangle$ for all $\Psi' \Vdash \psi \in \Psi$ and $V \approx V' \in R\psi$, we have $N\psi[M/a] \approx N'[M'/a] \in \Downarrow S\langle \mathrm{id}_\Psi, M/a\rangle$ for all $M \approx M' \in \Downarrow R$.

Proof. Suppose $M \approx M' \in \Downarrow R$, and let $\Psi_1 \Vdash \psi_1 \in \Psi$ and $\Psi_2 \Vdash \psi_2 \in \Psi_1$ be given. Then we have $M\psi_1 \Downarrow V$, $M'\psi_1 \Downarrow V'$, $M\psi_1\psi_2 \Downarrow V_{12}$, $M'\psi_1\psi_2 \Downarrow V'_{12}$, $V\psi_2 \Downarrow V_2$, and $V'\psi_2 \Downarrow V'_2$, for some values such that V_{12}, V_2 are pairwise related to V'_{12}, V'_2 by $R\psi_1\psi_2$.

By assumption, we know that $N\psi_1[V/a] \approx N'\psi_1[V'/a] \in \Downarrow S\langle \psi_1, V/a\rangle$. Note that as $[\![A]\!]$ is value-coherent, we have $\Psi_1 \Vdash M\psi_1 = V \in A\psi_1$ by Lemma 3.1.36, thus that $\Downarrow S\langle \psi_1, V/a\rangle = \Downarrow S\langle \psi_1, M\psi_1/a\rangle$. By instantiating with id_{Ψ_1} and ψ_2, we can conclude that $N\psi_1[V/a] \Downarrow W$ and $N'\psi_1[V'/a] \Downarrow W'$ with $P \approx P' \in \Downarrow S\langle \psi_1\psi_2, V\psi_1\psi_2/a\rangle$ for $P \in \{N\psi_1\psi_2[V\psi_2/a], W\psi_2\}$ and $P' \in \{N'\psi_1\psi_2[V'\psi_2/a], W'\psi_2\}$.

Also by assumption, we know that $N\psi_1\psi_2[X/a] \approx N'\psi_1\psi_2[X'/a] \in \Downarrow S\langle \psi_1\psi_2, X/a\rangle$ for $X \in \{V_{12}, V_2\}$ and $X' \in \{V'_{12}, V'_2\}$; again, we have $\Downarrow S\langle \psi_1\psi_2, X/a\rangle = \Downarrow S\langle \psi_1\psi_2, M\psi_1\psi_2/a\rangle$ for such X. Using the inclusion of $\Downarrow$ in $\Downarrow$, we have in particular that $N\psi_1\psi_2[X/a] \approx N'\psi_1\psi_2[X'/a] \in \Downarrow S\langle \psi_1\psi_2, M\psi_1\psi_2/a\rangle$ for such X, X'. Because $a.N, a.N'$ are eager terms, we know that $N\psi_1\psi_2[V_{12}/a]$ has the same value as $N\psi_1\psi_2[M\psi_1\psi_2/a]$ and $N\psi_1\psi_2[V_2/a]$ has the same value as $N\psi_1\psi_2[V\psi_2/a]$; likewise for their primed equivalents. Thus $N\psi_1\psi_2[Q/a] \approx N'\psi_1\psi_2[Q'/a] \in \Downarrow S\langle \psi_1\psi_2, M\psi_1\psi_2/a\rangle$ for $Q \in \{M\psi_1\psi_2, V\psi_2\}$ and $Q' \in \{M'\psi_1\psi_2, V'\psi_2\}$.

Using that S is a PER, we may combine the above to conclude that we have $P \approx P' \in \Downarrow S\langle \psi_1\psi_2, V\psi_1\psi_2/a\rangle$ for $P \in \{N\psi_1\psi_2[M\psi_1\psi_2/a], W\psi_2\}$ and $P' \in \{N\psi_1\psi_2[M'\psi_1\psi_2/a], W'\psi_2\}$, as required. □

3.1.6.1 Path types

The first type we consider is the path type, which provides a gentle introduction to reasoning with Ψ-relations. None of the operational semantics rules for path types depends on

the status of an interval term, with the effect that the proofs are more or less the same as they would be in ordinary Martin-Löf type theory. By the same token, the rules for the existing types of Martin-Löf type theory are easy to reprove in the cubical setting. Of course, we must now also check that each of these types supports coercion and composition.

Rule 3.1.39 (Path pretype formation).

$$\frac{\Psi, x : \mathbb{I} \Vdash A = A' \text{ type} \qquad \Psi \Vdash M_0 = M_0' \in A[0/x] \qquad \Psi \Vdash M_1 = M_1' \in A[1/x]}{\Psi \Vdash \mathsf{Path}(x.A, M_0, M_1) = \mathsf{Path}(x.A', M_0', M_1') \text{ pretype}}$$

Proof. We aim to apply coherent value introduction, Lemma 3.1.34. For every $\Psi' \Vdash \psi \in \Psi$, we see that $\mathsf{Path}(x.A, M_0, M_1)\psi$ and $\mathsf{Path}(x.A', M_0', M_1')\psi$ are values. Moreover, we have $\tau_i \vDash \Psi' \Vdash \mathsf{Path}(x.A, M_0, M_1)\psi \approx \mathsf{Path}(x.A', M_0', M_1') \downarrow R\psi$ where the Ψ-relation R is defined like so.

$$V \approx V' \in R\langle\psi\rangle \iff \begin{cases} V = \lambda^{\mathbb{I}} x.\, M \text{ and } V' = \lambda^{\mathbb{I}} x.\, M' \text{ for some } M, M' \\ \text{with } \tau \vDash \Psi', x : \mathbb{I} \Vdash M = M' \in A\psi \text{ and} \\ \tau \vDash \Psi' \Vdash M[\varepsilon/x] = M_\varepsilon \psi \in A\psi[\varepsilon/x] \text{ for each } \varepsilon \in \{0, 1\} \end{cases}$$

This relies on the stability of the judgments under substitution: for all $\Psi' \Vdash \psi \in \Psi$, we have $\Psi', x : \mathbb{I} \Vdash A\psi = A'\psi$ pretype, $\Psi' \Vdash M_0\psi = M_0'\psi \in A\psi[0/x]$, and $\Psi' \Vdash M_1\psi = M_1'\psi \in A\psi[1/x]$.

In other words, we have $\mathsf{Path}(x.A, M_0, M_1)\psi \approx \mathsf{Path}(x.A', M_0', M_1')\psi \in \tau_i[R]\psi$ for every $\Psi' \Vdash \psi \in \Psi$. It follows by Lemma 3.1.34 that $\mathsf{Path}(x.A, M_0, M_1) \approx \mathsf{Path}(x.A', M_0', M_1') \in \Downarrow \tau_i[R]$, which is to say that $\Psi \Vdash \mathsf{Path}(x.A, M_0, M_1) = \mathsf{Path}(x.A', M_0', M_1')$ pretype. □

The above provides one case of type value-coherence, necessary to show τ_i is a type system: it implies that whenever $\tau_i \vDash \Psi \Vdash \mathsf{Path}(x.A, M_0, M_1)\psi \approx \mathsf{Path}(x.A', M_0', M_1')\psi \downarrow R$, we actually have $\Psi \Vdash \mathsf{Path}(x.A, M_0, M_1)\psi = \mathsf{Path}(x.A', M_0', M_1')\psi$ pretype.

The introduction rule follows by a similar argument.

Rule 3.1.40 (Path introduction).

$$\frac{\Psi, x : \mathbb{I} \Vdash A \text{ type} \qquad \Psi, x : \mathbb{I} \Vdash M \in A}{\Psi \Vdash \lambda^{\mathbb{I}} x.\, M = \lambda^{\mathbb{I}} x.\, M' \in \mathsf{Path}(x.A, M[0/x], M[1/x])}$$

Proof. Once again, we go by Lemma 3.1.34. For every $\Psi' \Vdash \psi \in \Psi$, we have $(\lambda^{\mathbb{I}} x.\, M)\psi \approx (\lambda^{\mathbb{I}} x.\, M')\psi \in R\psi$, where R is as defined in the proof of Rule 3.1.39, using the stability of our hypotheses under substitution. It therefore follows that $\lambda^{\mathbb{I}} x.\, M \approx \lambda x.\, M' \in \Downarrow R$. □

The introduction rule shows that the relation named by the path type is itself value-coherent. Formally, this result is a prerequisite to defining the candidate type system, but there is no real circularity here, only a perversion of the conceptual order for presentation's sake.

For elimination, it is convenient to prove the reduction rule *before* the binary elimination rule itself. At this point we switch from blindly applying value introduction to blindly applying coherent head expansion (Lemma 3.1.35).

Rule 3.1.41 (Path reduction).

$$\frac{\Psi, x : \mathbb{I} \Vdash A\ \mathsf{type} \qquad \Psi, x : \mathbb{I} \Vdash M \in A \qquad \Psi \Vdash r \in \mathbb{I}}{\Psi \Vdash (\lambda^{\mathbb{I}} x.\, M)\, r = M[r/x] \in A[r/x]}$$

Proof. By substitution, we know that $\Psi \Vdash M[r/x] \in A[r/x]$. For all $\Psi' \Vdash \psi \in \Psi$, we have $((\lambda^{\mathbb{I}} x.\, M)\, r)\psi \longmapsto M[r/x]\psi$, so $\Psi \Vdash (\lambda^{\mathbb{I}} x.\, M)\, r = M[r/x] \in A[r/x]$ by coherent expansion. □

As path application evaluates its principal argument, we use the elimination lemma (Lemma 3.1.38) to prove its well-typedness.

Rule 3.1.42 (Path elimination).

$$\frac{\begin{array}{c}\Psi, x : \mathbb{I} \Vdash A\ \mathsf{type} \qquad (\forall \varepsilon)\ \Psi \Vdash M_\varepsilon \in A[\varepsilon/x] \\ \Psi \Vdash P = P' \in \mathsf{Path}(x.A, M_0, M_1) \qquad \Psi \Vdash r \in \mathbb{I}\end{array}}{\Psi \Vdash P\, r = P'\, r \in A[r/x]}$$

Proof. By applying Lemma 3.1.38 with the eager terms $(-)\, r$ and $(-)\, r$, it suffices to prove that for every $\Psi' \Vdash \psi \in \Psi$ and $\Psi', x : \mathbb{I} \Vdash M = M' \in A\psi$, we have $\Psi' \Vdash (\lambda^{\mathbb{I}} x.\, M)\, (r\psi) = (\lambda^{\mathbb{I}} x.\, M')\, (r\psi) \in A[r/x]\psi$. By substitution, we have $\Psi' \Vdash M[r\psi/x] = M'[r\psi/x] \in A[r/x]\psi$, from which the necessary equation follows by applying path reduction on either side. □

In addition to the usual suite of rules, we also want to know that the endpoints of a path element are equal to those prescribed by its type. For this we use the evaluation lemma (Lemma 3.1.36) to reduce to the case where the path is a value. Here we need that the relation named by the path type is value-coherent, which we established with the introduction rule.

Rule 3.1.43 (Path boundary).

$$\frac{\begin{array}{c}\Psi, x : \mathbb{I} \Vdash A\ \mathsf{type} \qquad (\forall \varepsilon)\ \Psi \Vdash M_\varepsilon \in A[\varepsilon/x] \\ \Psi \Vdash P \in \mathsf{Path}(x.A, M_0, M_1) \qquad \varepsilon \in \{0, 1\}\end{array}}{\Psi \Vdash P\, \varepsilon = M_\varepsilon \in A[\varepsilon/x]}$$

Proof. By Lemma 3.1.36, we have that $P \Downarrow V$ with $\Psi \Vdash P = V \in \mathsf{Path}(x.A, M_0, M_1)$. By the elimination rule already proven, we know $\Psi \Vdash P\,\varepsilon = V\,\varepsilon \in A[\varepsilon/x]$. Moreover, V is of the form $\lambda^{\mathbb{I}} x.\, M$ with $\Psi, x : \mathbb{I} \Vdash M \in A$ and $\Psi \Vdash M[\varepsilon/x] = M_\varepsilon \in A[\varepsilon/x]$. By path reduction, we then have $\Psi \Vdash V\,\varepsilon = M[\varepsilon/x] \in A[\varepsilon/x]$. We obtain the result by concatenating $P\,\varepsilon = V\,\varepsilon$, $V\,\varepsilon = M[\varepsilon/x]$, and $M[\varepsilon/x] = M_\varepsilon$. □

Finally, the uniqueness rule follows in much the same way.

Rule 3.1.44 (Path uniqueness).

$$\frac{\Psi, x : \mathbb{I} \Vdash A\ \mathsf{type} \qquad (\forall \varepsilon)\ \Psi \Vdash M_\varepsilon \in A[\varepsilon/x] \qquad \Psi \Vdash P \in \mathsf{Path}(x.A, M_0, M_1)}{\Psi \Vdash P = \lambda^{\mathbb{I}} x.\, P\,x \in \mathsf{Path}(x.A, M_0, M_1)}$$

Proof. By Lemma 3.1.36, we have that $P \Downarrow \lambda^{\mathbb{I}} x.\, M$ with with $\Psi, x : \mathbb{I} \Vdash M \in A$ and $\Psi \Vdash M[\varepsilon/x] = M_\varepsilon \in A[\varepsilon/x]$ for $\varepsilon \in \{0, 1\}$; by weakening and path elimination, we know $\Psi, x : \mathbb{I} \Vdash P\,x = (\lambda^{\mathbb{I}} x.\, M)\,x \in A$. Path reduction then gives $x : \mathbb{I} \Vdash (\lambda^{\mathbb{I}} x.\, M)\,x = M \in A$, so by transitivity $x : \mathbb{I} \Vdash P\,x = M \in A$. Applying path introduction, we get $\Psi \Vdash \lambda^{\mathbb{I}} x.\, P\,x = \lambda^{\mathbb{I}} x.\, M \in \mathsf{Path}(x.A, M_0, M_1)$, which combined with $\Psi \Vdash P = \lambda^{\mathbb{I}} x.\, M \in \mathsf{Path}(x.A, M_0, M_1)$ gives the desired equation. □

To prove that the function type supports the Kan operations is equally straightforward. Like the reduction rules for function application, the reduction rules for coe and hcom in the function type (Figure 3.2) are stable under interval substitution. Therefore, it is only necessary to check that the reduced terms are well-typed and satisfy the necessary equations (*e.g.*, that coercion and composition $r \longrightarrow r$ are identity functions); the results for the unreduced terms then follow by coherent head expansion. We leave these verifications as an exercise for the reader.

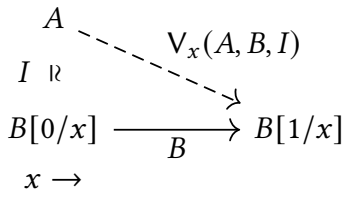
3.1.6.2 V types

The univalence axiom is realized in cubical type theory by V *types*, which create lines of types from isomorphisms. Although we will not have much need to work with V types directly—we mostly use the univalence theorem they imply—they do provide a more thorough exercise of the lemmas defined above: unlike function types, the operational semantics rules of V types are not stable under interval substitution.

Given $\Psi \Vdash A, B$ type and an isomorphism $\Psi \Vdash I \in A \simeq B$ between them, the V type $\Psi, x : \mathbb{I} \Vdash \mathsf{V}_x(A, B, I)$ type is a path that connects the types A and B: we will have $\Psi \Vdash \mathsf{V}_0(A, B, I) = A$ type and $\Psi \Vdash \mathsf{V}_1(A, B, I) = B$ type, as the reduction rules in Figure 3.1 suggest. While we will not go through this in any detail here, coercion along a V type applies the isomorphism: $\lambda a.\, \mathrm{coe}^{0 \to 1}_{x.\mathsf{V}_x(A,B,I)}(a)$ is equal, up to a path, to the underlying

function $\mathrm{fst}(I)$ of the isomorphism. Thus V types provide an inverse to the function that converts paths of types to isomorphisms using coe.

The precise formulation of V is slightly more subtle: rather than merely creating a path from an isomorphism, V actually *composes* an isomorphism onto an existing path to produce a new path. That is, given $\Psi \Vdash A$ type, a path $\Psi, x : \mathbb{I} \Vdash B$ type, and an isomorphism $\Psi \Vdash I \in A \simeq B[0/x]$ between A and the zero endpoint of B, the type $\Psi, x : \mathbb{I} \Vdash \mathsf{V}_x(A, B, I)$ type is a path between A and the one endpoint of B. The V type therefore connects the two ends of a "V" shape formed by I and B.



When B happens to be a *degenerate* path, we recover the picture of a path directly constructed from an isomorphism.

Put in terms of an arbitrary interval term r, as opposed to a variable x, we arrive at the following formation and boundary rules, which require the arguments A and I to exist only under the constraint $r \equiv 0$.

Rules 3.1.45 (V type formation).

$$\frac{\Psi \Vdash r \in \mathbb{I} \qquad \Psi, r \equiv 0 \gg A = A' \text{ type} \qquad \Psi \Vdash B = B' \text{ type} \qquad \Psi, r \equiv 0 \gg I = I' \in A \simeq B}{\Psi \Vdash \mathsf{V}_r(A, B, I) = \mathsf{V}_r(A', B', I') \text{ pretype}}$$

$$\frac{\Psi \Vdash A \text{ type}}{\Psi \Vdash \mathsf{V}_0(A, B, I) = A \text{ pretype}} \qquad \frac{\Psi \Vdash B \text{ type}}{\Psi \Vdash \mathsf{V}_1(A, B, I) = B \text{ pretype}}$$

Proof. The reduction rules follow immediately from coherent head expansion, as we have $\mathsf{V}_0(A, B, I)\psi \longmapsto A\psi$ and $\mathsf{V}_1(A, B, I)\psi \longmapsto B\psi$ for all interval substitutions ψ. For the formation rule, we go by the coherent value lemma applied to $\tau_i[R]$, where R is the Ψ-PER assigned to the V type in Example 3.1.32. For a given $\Psi' \Vdash \psi \in \Psi$, we are in one of the following cases.

- Case: $r\psi = 0$. Then by the reduction rule already proven, we have $\Psi' \Vdash \mathsf{V}_{r\psi}(A, B, I) = A$ pretype and $\Psi' \Vdash \mathsf{V}_{r\psi}(A', B', I') = A'$ pretype. By transitivity, it follows that $\Psi' \Vdash \mathsf{V}_r(A, B, I)\psi = \mathsf{V}_r(A', B', I')\psi$ pretype.
- Case: $r\psi = 1$. Symmetric to the previous case.

- Case: $r\psi = x$. Then $\mathsf{V}_r(A, B, I)\psi$ and $\mathsf{V}_r(A', B', I')\psi$ are values, and we have $\tau_i \vDash \Psi' \Vdash \mathsf{V}_r(A, B, I)\psi \approx \mathsf{V}_r(A', B', I')\psi \downarrow R_\psi$ by definition of the type system. □

The above proof is representative of the shape of formation and introduction proofs for types with unstable operational semantics. Typically, the boundary of a term reduces in some way, in which case we apply some reduction rules to simplify the goal to an equation we already know to hold. When we are not on the boundary, on the other hand, the terms in question are typically values.

A value of the type $\Psi, x : \mathbb{I} \Vdash \mathsf{V}_x(A, B, I)$ type is a term $\mathsf{v}_x(M, P)$, which collects a line $\Psi, x : \mathbb{I} \Vdash P \in B$ in direction x with a term $\Psi \vdash M : A$, living at $x \equiv 0$, which is mapped by $\mathrm{fst}(I)$ to $P[0/x]$. This data parallels the shape of the V type itself.

We see intuitively that the dependent paths over the V type, *i.e.*, the elements of some type $\mathsf{Path}(x.\mathsf{V}_x(A, B, I), M, N)$, correspond to paths in B establishing that M and N correspond to each other across the isomorphism I. That is, the prototypical element of this path type is of the form $\lambda x.\, \mathsf{v}_x(M, P)$ where $\Psi \Vdash P[0/x] = (\mathrm{fst}(I))\, M \in B$ and $\Psi \Vdash P[1/x] = N \in B$.

Rules 3.1.46 (V type introduction).

$$\frac{\Psi \Vdash r \in \mathbb{I} \qquad \Psi, r \equiv 0 \gg I \in A \simeq B \qquad \Psi, r \equiv 0 \gg M = M' \in A \qquad \Psi \Vdash N = N' \in B \qquad \Psi, r \equiv 0 \gg (\mathrm{fst}(I))\, M = N \in B}{\Psi \Vdash \mathsf{v}_r(M, N) = \mathsf{v}_r(M', N') \in \mathsf{V}_r(A, B, I)}$$

$$\frac{\Psi \Vdash M \in A}{\Psi \Vdash \mathsf{v}_0(M, N) = M \in A} \qquad \frac{\Psi \Vdash N \in B}{\Psi \Vdash \mathsf{v}_1(M, N) = N \in B}$$

We leave the proofs of these rules as an exercise to the reader; they follow the same pattern as the proof of the formation rules.

The elimination operator for V types extracts an element of B. With the reduction rules for V types, we have examples of coherent head expansion where the reduction rule is not stable under substitution.

Rules 3.1.47 (V type reduction).

$$\frac{\Psi \Vdash r \in \mathbb{I} \qquad \Psi, r \equiv 0 \gg I \in A \simeq B \qquad \Psi, r \equiv 0 \gg M \in A \qquad \Psi \Vdash N \in B \qquad \Psi, r \equiv 0 \gg (\mathsf{fst}(I))\, M = N \in B}{\Psi \Vdash \mathsf{vproj}_r(\mathsf{v}_r(M,N), I) = N \in B}$$

$$\frac{\Psi \Vdash M \in A \qquad \Psi \Vdash I \in A \simeq B}{\Psi \Vdash \mathsf{vproj}_0(M, I) = (\mathsf{fst}(I))\, M \in A} \qquad \frac{\Psi \Vdash N \in B}{\Psi \Vdash \mathsf{vproj}_1(M, N) = N \in B}$$

Proof. Again, the reduction rules in the 0 and 1 cases follow immediately by coherent head expansion. We also apply coherent head expansion for the first rule, but now we have to do some case analysis. Let $\Psi' \Vdash \psi \in \Psi$ be given. Then we are in one of three cases.

- Case: $r\psi = 0$. Then $\mathsf{vproj}_r(\mathsf{v}_r(M,N), I)\psi \longmapsto \mathsf{vproj}_r(M, I)\psi$. By the reduction rule for 0 just proven and the assumed equation $\Psi, r \equiv 0 \gg (\mathsf{fst}(I))\, M = N \in B$, the latter is equal to $N\psi$ in $B\psi$.
- Case: $r\psi = 1$. Then $\mathsf{vproj}_r(\mathsf{v}_r(M,N), I)\psi \longmapsto \mathsf{vproj}_r(N, I)\psi$, and the latter is equal to $N\psi$ in $B\psi$ by the reduction rule for 1 just proven.
- Case: $r\psi = x$. Then $\mathsf{vproj}_r(\mathsf{v}_r(M,N)\psi, I) \longmapsto N\psi$, and the latter is well-typed by hypothesis. □

With these rules, we have seen enough to get a sense of how proofs of rules proceed in cubical computational type theory. Although the definition of $\Vdash -$ is hairy, the process is fairly intuitive filtered through the lens of our battery of lemmas: when we check that a term is well-typed, we need to make sure that its substitution instances behave in a way that is coherent up to the equality of the type.

3.2 Programming in a cubical type theory

We now give a few basic definitions and constructions *within* a cubical type theory. These are largely chosen for their relevance to more novel results in *parametric* cubical type theory that we construct in Part III, Chapter 10 and Part IV, Chapter 15, but we hope to also give a taste of cubical argumentation. The reader interested in developing further intuition can find further examples of cubical programming and theorem-proving in [VMA19, §2; Ben19; MP20; ACMZ21]; we also suggest experimenting with the **redtt** proof assistant [redtt] and the **Agda** proof assistant's cubical mode and library [Agda; CubAg].

We give our proofs in a style reminiscent of the **HoTT** Book [Uni13]: a combination of textual argument and explicit syntax. Textual statements should be understood as syntactic sugar for type expressions: when we say "for all $a : A$, there exists $b : B$ such that…", we mean that the type $(a : A) \to (b : B) \times \cdots$ is inhabited, not some metatheoretic property. Also, we will use the notation $M_0 \rightsquigarrow M_1$ an informal shorthand for the path type $\mathsf{Path}(A, M_0, M_1)$.

3.2.1 Path induction

We have seen that, via coercion, we can transport properties between path-equal terms. We now show that transport further induces an *a priori* stronger principle: a version of the J eliminator that defines identity types (Section 2.1.5.4). We obtain this result as a corollary of *singleton contractibility*.

Definition 3.2.1. A type is a *proposition* if there is a path between any pair of elements in A, that is, if the following type is inhabited.

$$\mathsf{IsProp}(A) \coloneqq (a_0, a_1 : A) \to \mathsf{Path}(A, a_0, a_1)$$

We define the *universe of propositions* as $\mathsf{U} \coloneqq (A : \mathsf{U}) \times \mathsf{IsProp}(A)$. A is *contractible* when it is an inhabited proposition.

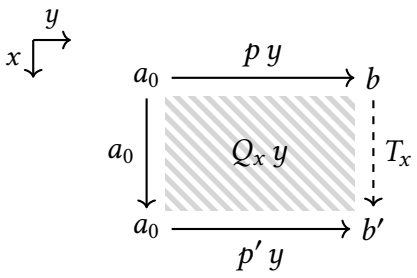
$$\mathsf{IsContr}(A) \coloneqq A \times \mathsf{IsProp}(A)$$

Equivalently, a type is contractible when it contains an element to which all its other elements are equal up to a path. Singleton contractibility says that *singleton type* $(a : A) \times \mathsf{Path}(A, a_0, a)$ of terms path-equal to some fixed point $a_0 : A$ is always contractible, the canonical inhabitant being $\langle a_0, \lambda^{\mathbb{I}}\_.\, a_0 \rangle$. Given any other element $\langle b, p \rangle$, we have evidence p that the first component b is path-equal to a_0, and we can moreover show that $\lambda^{\mathbb{I}}\_.\, a_0$ and p correspond over this path.

Lemma 3.2.2 (Singleton contractibility). For any A type and $a_0 : A$, the singleton type $(a : A) \times \mathsf{Path}(A, a_0, a)$ is contractible.

Proof. The singleton type is inhabited, as a_0 is equal to itself by the reflexive path: we have $\langle a_0, \lambda^{\mathbb{I}}\_.\, a_0 \rangle : (a : A) \times \mathsf{Path}(A, a_0, a)$. To see that the singleton type is a proposition, suppose we are given $\langle b, p \rangle, \langle b, p' \rangle : (a : A) \times \mathsf{Path}(A, a_0, a)$. To construct a path between these in the type $(a : A) \times \mathsf{Path}(A, a_0, a)$, we need a pair of terms $x : \mathbb{I} \gg T_x \in A$ and $x : \mathbb{I} \gg Q_x \in \mathsf{Path}(A, a_0, T_x)$ that reduce to $\langle b, p \rangle$ when $x = 0$ and $\langle b', p' \rangle$ when $x = 1$. We might look to define $\lambda^{\mathbb{I}} x.\, T_x$ as the concatenation of $p^{-1} : b \rightsquigarrow a_0$ with $p : a_0 \rightsquigarrow b'$, but we will actually

have an easier time if we take a look at Q_x first. For this term, what we need is a two-dimensional term fitting in the following square boundary.



This is a perfect candidate for the application of homogeneous composition: we have a box with three fixed sides and one undetermined side (T_x, which is up to us to define as we like) that we must fill with a square term. We may therefore define Q_x as follows.

$$Q_x \coloneqq \lambda^{\mathbb{I}} y.\, \mathsf{hcom}_A^{0 \to y}(a_0; x = 0 \hookrightarrow z.p\, z, x = 1 \hookrightarrow y.p'\, z)$$

From the tube constraints of this composite, we have that $Q_0 = p \in \mathsf{Path}(A, a_0, b)$ and $Q_1 = p' \in \mathsf{Path}(A, a_0, b')$. If we define $T_x \coloneqq Q_x\, 1$, we moreover see that $x : \mathbb{I} \gg Q_x \in \mathsf{Path}(A, a_0, T_x)$, and so

$$\lambda^{\mathbb{I}} x.\, \langle T_x, Q_x \rangle \in \mathsf{Path}((a : A) \times \mathsf{Path}(A, a_0, a), \langle b, p \rangle, \langle b', p' \rangle)$$

as desired. □

Lemma 3.2.3 (J for paths). Let a $a_0 : A, a_1 : A, p : \mathsf{Path}(A, a_0, a_1) \gg B$ type be given with some $d : (a : A) \to B[a/a_0, a/a_1, \lambda^{\mathbb{I}}\_.\, a/p]$. Then B is inhabited for any $a_0 : A$, $a_1 : A$, and $p : \mathsf{Path}(A, a_0, a_1)$.

Proof. Given $a_0 : A$, $a_1 : A$, and $p : \mathsf{Path}(A, a_0, a_1)$, we have two elements of the singleton type $(a : A) \times \mathsf{Path}(A, a_0, a)$: the canonical $\langle a_0, \lambda^{\mathbb{I}}\_.\, a_0 \rangle$ as well as $\langle a_1, p \rangle$. By Lemma 3.2.2, there is a path between these, some Q of type $\langle a_0, \lambda^{\mathbb{I}}\_.\, a_0 \rangle \rightsquigarrow \langle a_1, p \rangle$. Fixing a_0, we can recast B as a type family indexed by singletons like so.

$$c : (a : A) \times \mathsf{Path}(A, a_0, a) \gg B' \coloneqq B[\mathsf{fst}(c)/a_1, \mathsf{snd}(c)/p]\ \mathsf{type}$$

Then we have $d\, a_0 \in B'[\langle a_0, \lambda^{\mathbb{I}}\_.\, a_0 \rangle / c]$. We obtain our desired result by coercing $d\, a_0$ along our path between the singletons: $\mathsf{coe}^{0 \to 1}_{x.B'[Q\, x/c]}(d\, a_0) \in B'[\langle b_1, q \rangle / c]$. □

3.2.2 Paths in compound types

One theoretical and practical benefit of path equality is the ease with which we can characterize the path types of compound types. We have already seen this implicitly in the previous section, where we built a path in a product type (the singleton type) from a pair of paths $\lambda^{\mathbb{I}} x.\, T_x$ and $\lambda^{\mathbb{I}} x.\, Q_x$ in the component types. That particular case is an instance of the following general principle: we have an isomorphism between paths in products and products of paths.

Lemma 3.2.4 (Paths in products). Let $x : \mathbb{I} \gg A$ type and $x : \mathbb{I}, a : A \gg B$ type be given together with $t_0 : ((a : A) \times B)[0/x]$ and $t_1 : ((a : A) \times B)[1/x]$. Then we have an isomorphism of the following type.

$$\begin{gathered}\mathsf{Path}(x.(a : A) \times B, t_0, t_1) \\ \simeq \\ (p : \mathsf{Path}(x.A, \mathsf{fst}(t_0), \mathsf{fst}(t_1))) \times \mathsf{Path}(x.B[p\,x/a], \mathsf{snd}(t_0), \mathsf{snd}(t_1))\end{gathered}$$

That is, a path in a product type is a product of paths.

Proof. In the forward direction, given t:$\mathsf{Path}(x.(a : A) \times B, t_0, t_1)$, we have the pair of paths $\langle \lambda^{\mathbb{I}} x.\, \mathsf{fst}(t\,x), \lambda^{\mathbb{I}} x.\, \mathsf{snd}(t\,x) \rangle$. In the reverse, given a pair of paths across the two types, $p : \mathsf{Path}(x.A, \mathsf{fst}(t_0), \mathsf{fst}(t_1))$ and $q : \mathsf{Path}(x.B[p\,x/a], \mathsf{snd}(t_0), \mathsf{snd}(t_1))$, we have a path in the product type $\lambda^{\mathbb{I}} x.\, \langle p\,x, q\,x \rangle$. It is straightforward to check that these two constructions are inverse up to exact equality. □

The following characterization of paths in function types—a path between functions is a proof they are path-equal on all arguments—is similarly immediate.

Lemma 3.2.5 (Function extensionality). Let A type and $x : \mathbb{I}, a : A \gg B$ type be given together with $f_0 : (a : A) \to B[0/x]$ and $f_1 : (a : A) \to B[1/x]$. Then we have an isomorphism of the following type.

$$\mathsf{Path}(x.(a : A) \to B, f_0, f_1) \simeq (a : A) \to \mathsf{Path}(x.B, f_0\,a, f_1\,a)$$

That is, a path in a function type is a family of paths when the domain is homogeneous.

Proof. Given p in the former type, we have $\lambda a.\, \lambda^{\mathbb{I}} x.\, p\,x\,a$ in the latter; given h in the latter, we have $\lambda^{\mathbb{I}} x.\, \lambda a.\, h\,a\,x$ in the former. These two constructions are clearly inverse up to exact equality. □

The above is, however, limited to dependent paths $\mathsf{Path}(x.(a : A) \to B, F_0, F_1)$ where the domain type A is independent of x. We can give a more general principle in the case

that A depends on x: a path between functions is a function from paths in the domain to paths in the codomain. The proof of this result is rather more involved and in particular makes use of coercion.

Lemma 3.2.6 (Paths in function types). Let $x : \mathbb{I} \gg A$ type and $x : \mathbb{I}, a : A \gg B$ type be given together with $f_0 : ((a : A) \to B)[0/x]$ and $f_1 : ((a : A) \to B)[1/x]$. Then we have an isomorphism of the following type.

$$\mathsf{Path}(x.(a : A) \to B, f_0, f_1)$$
$$\simeq$$
$$(a_0 : A[0/x])\ (a_1 : A[1/x])\ (p : \mathsf{Path}(x.A, a_0, a_1)) \to \mathsf{Path}(x.B[p\,x/a], f_0\,a_0, f_1\,a_1)$$

That is, a path in a function type is a function from paths in the domain to paths in the codomain.

Proof. Given q in the former type, we have $\lambda a_0.\, \lambda a_1.\, \lambda p.\, \lambda^{\mathbb{I}} x.\, (q\,x)\,(p\,x)$ in the latter.

Conversely, suppose we are given h in the latter. Supposing $x : \mathbb{I}$ and $a : A$, we must construct an element of B that becomes $f_0\,a$ when $x = 0$ and $f_1\,a$ when $x = 1$. Employing coercion, we can create a path P_x along A from the single element a.

$$P_x \coloneqq \lambda^{\mathbb{I}} y.\, \mathrm{coe}^{x \to y}_{x.A}(a) \in \mathsf{Path}(y.A[y/x], \mathrm{coe}^{x \to 0}_{x.A}(a), \mathrm{coe}^{x \to 1}_{x.A}(a))$$

Note that we have $P_x\,x = a \in A$. By applying h to this path, we obtain a corresponding path along B.

$$h\,(P_x\,0)\,(P_x\,1)\,P_x \in \mathsf{Path}(y.B[y/x, P_x\,y/a], f_0\,(P_x\,0), f_1\,(P_x\,1))$$

Our solution is the evaluation of this path at x, the term $h\,(P_x\,0)\,(P_x\,1)\,P_x\,x \in B$, which has the right type thanks to the equation $P_x\,x = a \in A$. When x is 0, it becomes $f_0\,(P_0\,0)$, which is again $f_0\,a$; when x is 1, it is $f_1\,(P_1\,1) = f_1\,a$.

Now we must check that the two constructions above are mutually inverse. First, given $q : \mathsf{Path}(x.(a : A) \to B, f_0, f_1)$, we need a path of the following type.

$$(\lambda^{\mathbb{I}} x.\, \lambda a.\, q\,x\,((\lambda^{\mathbb{I}} y.\, \mathrm{coe}^{x \to y}_{x.A}(a))\,x)) \rightsquigarrow q$$

In fact, this equation holds up to exact equality, thanks to the reduction equation for trivial coercions.

For the other inverse condition, we see after a bit of computation that we must construct a path of the following type for h in the right hand type.

$$(\lambda a_0.\, \lambda a_1.\, \lambda p.\, \lambda^{\mathbb{I}} x.\, h\,(\mathrm{coe}^{x \to 0}_{x.A}(p\,x))\,(\mathrm{coe}^{x \to 1}_{x.A}(p\,x))\,(\lambda^{\mathbb{I}} y.\, \mathrm{coe}^{x \to y}_{x.A}(p\,x))\,x) \rightsquigarrow h$$

The key step, then, is to construct paths $\mathrm{coe}^{x\to 0}_{x.A}(p\,x) \rightsquigarrow a_0$, $\mathrm{coe}^{x\to 1}_{x.A}(p\,x) \rightsquigarrow a_1$, and $\mathrm{coe}^{x\to y}_{x.A}(p\,x) \rightsquigarrow p\,y$. We can produce the third, which implies the others, as follows.

$$\mathrm{coe}^{x\to y}_{z.\mathsf{Path}(A[z/x],\mathrm{coe}^{x\to z}_{x.A}(p\,x),p\,z)}(\lambda^{\mathbb{I}}\_.\,p\,x) \in \mathsf{Path}(A[y/x], \mathrm{coe}^{x\to y}_{x.A}(p\,x), p\,y)$$

That is, the equation holds by reflexivity when y is x, so we can extend it to all other values of y by coercion. □

Remark 3.2.7. In the case where A is degenerate, Lemma 3.2.6 gives us the following isomorphism.

$$\mathsf{Path}(x.(a:A) \to B, f_0, f_1)$$
$$\simeq$$
$$(a_0 : A)\ (a_1 : A)\ (p : \mathsf{Path}(A, a_0, a_1)) \to \mathsf{Path}(x.B[p\,x/a], f_0\,a_0, f_1\,a_1)$$

We can re-derive the alternative characterization in Lemma 3.2.5 from this principle by singleton contractibility: any pair of arguments a_1, p is equal up to a path to $a_0, \lambda^{\mathbb{I}}\_.\,a_0$.

We round out this section with a couple of results that we will not prove in detail—they are not particularly difficult, but are easiest to prove with a larger toolbox of lemmas than we want to set up here—but which will be useful in the future.

The first of these shows that in order to characterize the path family at some type, we do not need to build an isomorphism explicitly: we only need one of the inverse conditions, the one showing that the characterization is a *retract* of the path family.

Lemma 3.2.8 (Characterization by retract). Let A type and $R{:}A \times A \to \mathsf{U}$ and suppose we have two functions as follows.

$$f : (a_0, a_1 : A) \to R\,\langle a_0, a_1\rangle \to \mathsf{Path}(A, a_0, a_1)$$
$$g : (a_0, a_1 : A) \to \mathsf{Path}(A, a_0, a_1) \to R\,\langle a_0, a_1\rangle$$

If we have paths $g\,a_0\,a_1\,(f\,a_0\,a_1\,q) \rightsquigarrow q$ for all $a_0, a_1 : A$, then $\mathsf{Path}(A, a_0, a_1)$ is isomorphic to $R\,\langle a_0, a_1\rangle$ for all $a_0, a_1 : A$. Moreover, in this case *any* function with the type of f or g is an isomorphism.

Proof (sketch). See [Rij18, Corollary 1.2.6] for a more detailed proof (in **HoTT**).

Such a family of retracts implies that the product type $(a_1 : A) \times R\,\langle a_0, a_1\rangle$ is a retract of $(a_1 : A) \times \mathsf{Path}(A, a_0, a_1)$. The latter is a singleton type, therefore contractible. A retract of a contractible type is also contractible, so $(a_1 : A) \times R\,\langle a_0, a_1\rangle$ is contractible.

Given any family of functions with the same type as f or g, the induced map from $(a_1 : A) \times R\,\langle a_0, a_1\rangle$ to $(a_1 : A) \times \mathsf{Path}(A, a_0, a_1)$ is an isomorphism, because *any* function between contractible types is an isomorphism. That the induced map is an isomorphism in turn implies that the original family of functions is a family of isomorphisms [Rij18, Proposition 1.2.4].

Finally, we have the crucial univalence principle relating paths between types in the universe U to isomorphisms between those types. Note that by the above, we need only prove that the types $A \simeq B$ are retracts of the types $\mathsf{Path}(\mathsf{U}, A, B)$.

Theorem 3.2.9 (Univalence). Let $A, B \in \mathsf{U}$. Then the following function from paths in U to isomorphisms is an isomorphism.

$$\lambda p.\, \mathrm{coe}^{0 \simeq 1}_{x.p\,x} \in \mathsf{Path}(\mathsf{U}, A, B) \to (A \simeq B)$$

Proof. From [Ang19, Theorem 4.105] via Lemma 3.2.8 and [Uni13, Theorem 4.3.2], the last of which shows that isomorphisms are path-equal whenever their underlying forward functions are path-equal. □

3.3 Formalism and models

The cubical type theory τ_1 interprets most of the constructs of the **ITT** formalism sketched in Section 2.2, with the notable exception of identity types (which we have replaced with path types). In Section 5.3 of Part II, we describe one way to recover identity types in a cubical setting. With that lacuna patched, we will be able interpret **ITT** as well as the univalence axiom; using the higher inductive types also constructed in Part II, we can obtain a computational interpretation of **HoTT**.

Alternatively, we may abandon identity types and **HoTT** entirely and instead develop a natively cubical formalism. This approach has some notable benefits. For one, **HoTT** is lacking as a formalism from a computational standpoint, failing to satisfy any kind of adequacy theorem (Proposition 2.2.1) thanks to its lack of rules for reducing applications of univalence and higher inductive type eliminators. This makes it difficult to use **HoTT** to prove calculational results. Indeed, cubical type theory has an advantage in usability more broadly. This observation goes back to Licata and Brunerie, who showed that merely adopting a cubical organization for arguments in **HoTT** could drastically simplify proofs [LB15]. (My own master's thesis [Cav15] owes a tremendous debt to that observation.) The effect is even more pronounced in a natively cubical theory, as many of the rules which hold only up to identity in **HoTT** are exact in cubical type theory. The formulation of path equality as function-like also means that the characterization of paths in compound negative types is very straightforward, as in the proofs of Lemmas 3.2.4 and 3.2.5 above.

We can straightforwardly adjust the formalism for intensional type theory introduced in Section 2.2 to expose cubical elements. We add judgments for two new concepts: the interval and constraints.

Judgment	Presuppositions	Reading
$\Gamma \vdash r : \mathbb{I}$	(Γ ctx)	r is a path interval term in context Γ
$\Gamma \vdash r = r' : \mathbb{I}$	($\Gamma \vdash r, r' : \mathbb{I}$)	r and r' are equal path interval terms
$\Gamma \vdash \xi : \mathbb{F}$	(Γ ctx)	ξ is a constraint in context Γ
$\Gamma \vdash \xi = \xi' : \mathbb{F}$	($\Gamma \vdash \xi, \xi' : \mathbb{F}$)	ξ and ξ' are equal constraints
$\Gamma \vdash \xi$ satisfied	($\Gamma \vdash \xi : \mathbb{F}$)	ξ is a satisfied constraint in context Γ

The interval and constraint judgments are simple to axiomatize, interval and constraint assumptions behaving not dissimilarly to ordinary term variable assumptions. To begin with, we have two new context formers.

$$\frac{\Gamma \text{ ctx}}{\Gamma.\mathbb{I} \text{ ctx}} \qquad \frac{\Gamma \vdash \xi : \mathbb{F}}{\Gamma.\xi \text{ ctx}}$$

The rules for substitutions into contexts with an interval or constraint match their term equivalents.

$$\frac{\Gamma' \vdash \gamma : \Gamma \qquad \Gamma' \vdash r : \mathbb{I}}{\Gamma' \vdash \gamma.r : \Gamma.\mathbb{I}} \qquad \frac{}{\Gamma.\mathbb{I} \vdash \mathsf{p}_{\mathbb{I}} : \Gamma} \qquad \frac{\Gamma' \vdash \gamma : \Gamma \qquad \Gamma' \vdash \xi[\gamma] \text{ satisfied}}{\Gamma' \vdash \gamma.\star : \Gamma.\xi}$$

$$\frac{\Gamma \vdash \xi : \mathbb{F}}{\Gamma.\xi \vdash \mathsf{p}_{\mathbb{F}} : \Gamma} \qquad \Gamma' \vdash \mathsf{p} \circ (\gamma.r) = \gamma : \Gamma \qquad \Gamma' \vdash \gamma = (\mathsf{p}_{\mathbb{I}} \circ \gamma).\mathsf{v}_{\mathbb{I}}[\gamma] : \Gamma.\mathbb{I}$$

$$\Gamma' \vdash \mathsf{p}_{\mathbb{F}} \circ (\gamma.\star) = \gamma : \Gamma \qquad \Gamma' \vdash \gamma = (\mathsf{p}_{\mathbb{F}} \circ \gamma).\star : \Gamma.\xi$$

In addition to variables, the interval is inhabited by the two constants; constraints take the form of equations and are satisfied when those equations hold.

$$\frac{\Gamma \text{ ctx}}{\Gamma.\mathbb{I} \vdash \mathsf{v}_{\mathbb{I}} : \mathbb{I}} \qquad \frac{}{\Gamma \vdash 0 : \mathbb{I}} \qquad \frac{}{\Gamma \vdash 1 : \mathbb{I}} \qquad \frac{\Gamma \vdash \delta : \Delta \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash \mathsf{v}_{\mathbb{I}}[\delta.r] = r : \mathbb{I}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma.\xi \vdash \xi \text{ satisfied}}$$

$$\frac{\Gamma \vdash r : \mathbb{I} \qquad \Gamma \vdash s : \mathbb{I}}{\Gamma \vdash r \equiv s : \mathbb{F}} \qquad \frac{\Gamma \vdash r : \mathbb{I}}{\Gamma \vdash r \equiv r \text{ satisfied}}$$

$$\frac{\Gamma \vdash r : \mathbb{I} \qquad \Gamma \vdash s : \mathbb{I} \qquad \Gamma \vdash r \equiv s \text{ satisfied}}{\Gamma \vdash r = s : \mathbb{I}}$$

With the judgmental apparatus in place, we can specify the type-generic rules for coercion.

$$\frac{\Gamma.\mathbb{I} \vdash A\ \mathsf{type} \qquad \Gamma \vdash r, s : \mathbb{I} \qquad \Gamma \vdash M : A[\mathsf{id}.r]}{\mathrm{coe}_A^{r \to s}(M) : A[\mathsf{id}.s]}$$

$$\frac{\Gamma.\mathbb{I} \vdash A\ \mathsf{type} \qquad \Gamma \vdash r : \mathbb{I} \qquad \Gamma \vdash M : A[\mathsf{id}.r]}{\mathrm{coe}_A^{r \to r}(M) = M : A[\mathsf{id}.r]}$$

Composition is more involved, as we need a way to represent the list of terms constituting a tube. But our goal here is not to give a complete formalism for cubical type theory, only to set up enough structure to make sense of further additions in Part III. We therefore leave the remainder as an exercise to the reader and refer to [ABCFHL19; CCHM15] for more complete examples of cubical formalisms.

3.3.1 Models in cubical sets

The "standard" non-computational models for cubical formalisms interpret contexts as *cubical sets*, presheaves on a given *cube category*. For cartesian cubical type theory, the cube category has a simple description in terms of interval contexts. We assume some basic knowledge of category-theoretic terminology.

Definition 3.3.1. The *cartesian cube category* $\square_c$ is the category whose objects are interval contexts Ψ ictx and whose morphisms $\psi \in \square_c[\Psi', \Psi]$ from Ψ' to Ψ are interval substitutions $\Psi' \Vdash \psi \in \Psi$.

A *presheaf* on a category C is a family of sets indexed by elements of C, with transition functions between those sets indexed by the morphisms of C. More concisely, it is a functor from the opposite category of C into the category of sets.

Definition 3.3.2. A presheaf G on a category C consists of the following data.

- For every $c \in C$, an set $G(c)$.
- For every $f \in C[c', c]$, a function $G(f) : G(c) \to G(c')$.

We require that $G(id_c) = id_{G(c)}$ for every $c \in C$ and $G(f \circ g) = G(g) \circ G(f)$ for every $g \in C[c'', c']$ and $f \in C[c', c]$.

We write $PSh(C)$ for the category of presheaves on C. We define the morphisms $\alpha \in PSh(C)[G, H]$ to be families of functions $\alpha(c) : G(c) \to H(c)$ satisfying a naturality condition.

Definition 3.3.3. A morphism $\alpha \in PSh(C)[G, H]$ is a family $\alpha(c) : G(c) \to H(c)$ of functions such that $H(f) \circ \alpha(c) = \alpha(c') \circ G(f)$ for every $f \in C[c', c]$.

A presheaf $G \in PSh(\square_c)$, then, is a family of sets $G(\Psi)$ indexed by interval contexts (which we think of as the elements in context Ψ) with a function $G(\psi) : G(\Psi) \to G(\Psi')$ for every $\Psi' \Vdash \psi \in \Psi$ (which we think of as the action of interval substitution on those elements). Note the analogy to a context Γ ctx of one of our cubical type theories: for every Ψ, we have the set of closing substitutions $\Psi \Vdash \gamma \in \Gamma$ (modulo equality), and given $\Psi' \Vdash \psi \in \Psi$ and $\Psi \Vdash \gamma \in \Gamma$ we have an induced $\Psi' \Vdash \gamma\psi \in \Gamma$. In accordance with this analogy, cubical sets can serve as an alternative interpretation of the contexts of our cubical formalism, with substitutions between contexts interpreted as morphisms of presheaves.

To interpret the interval judgment, we make use of the *Yoneda embedding* よ, which takes objects of the indexing category to objects of the presheaf category.[1]

Definition 3.3.4. Given $c \in C$, we define $よ(c) \in PSh(C)$ by $よ(c)(d) \coloneqq C[d, c]$ and $よ(c)(f) \coloneqq (-) \circ f$.

We have an interval presheaf $\mathbb{I} \coloneqq よ(x{:}\mathbb{I}) \in PSh(\square_c)$ defined as the Yoneda embedding of the single-interval context. By definition, the elements of $\mathbb{I}(\Psi)$ at a context Ψ are the substitutions $\Psi \Vdash \psi \in (x : \mathbb{I})$, which is to say interval terms $\Psi \Vdash r \in \mathbb{I}$. We then interpret open interval terms $\Gamma \vdash r : \mathbb{I}$ as morphisms $[\![r]\!] \in PSh(\square_c)[[\![\Gamma]\!], \mathbb{I}]$ from the context's interpretation (a cubical set) into this interval presheaf. Context extension by an interval hypothesis is interpreted by (pointwise) product of presheaves: $[\![\Gamma.\mathbb{I}]\!] \coloneqq [\![\Gamma]\!] \times \mathbb{I}$ where $([\![\Gamma]\!] \times \mathbb{I})(\Psi) \coloneqq [\![\Gamma]\!](\Psi) \times \mathbb{I}(\Psi)$.

Types over a context Γ, meanwhile, are interpreted as families indexed by elements of $[\![\Gamma]\!]$ and equipped with interpretations of the Kan operations. First, let us define an intermediate notion of semantic pretype.

Definition 3.3.5. Given a presheaf G, a *semantic pretype over* G is a family T of sets $T(\Psi, g)$ indexed by pairs of $\Psi \in \square_c$ and $g \in G(\Psi)$ and equipped with transition functions $T(\psi, g) : T(\Psi, g) \to T(\Psi', G(\psi)(g))$ for every $\Psi' \Vdash \psi \in \Psi$ such that $T(id_\Psi) = id_{T(\Psi,g)}$ and $T(\psi\psi') = T(\psi') \circ T(\psi)$.

Again, this matches the computational setting, where an open pretype $\Gamma \gg A$ type is defined by the elements of its instances $\Psi \Vdash A\gamma$ type for Ψ ictx and $\Psi \Vdash \gamma \in \Gamma$. Note that given a transformation $\alpha : H \to G$ of syntactic contexts, we can reindex T above to get a semantic pretype α^*T over H: $(\alpha^*T)(\Psi, h) \coloneqq T(\Psi, \alpha(\Psi)(h))$ and $(\alpha^*T)(\psi, h) \coloneqq T(\psi, \alpha(\Psi)(h))$. We thereby interpret substitution on types.

Syntactic elements are interpreted by families of semantic elements.

[1] We employ the character よ ("yo") from the Japanese *hiragana* syllabary to represent the Yoneda embedding. The stylized よ symbol used here was created by Favonia.

Definition 3.3.6. Given a presheaf G and pretype T over G, a *semantic term in* T is a family of elements $t(\Psi, g) \in T(\Psi, g)$ indexed by $\Psi \in \square_c$ and $g \in G(\Psi)$ such that $T(\psi)(t(\Psi, g)) = t(\Psi', G(\psi)(g))$.

Finally, a semantic type is a pretype equipped with operations interpreting the rules for coercion and homogeneous composition.

Definition 3.3.7. Given a presheaf G and semantic pretype T over G, a *coercion operator* c for T is a family of elements as follows: for every $\Psi \in \square_c$, interval terms $r, s \in \mathbb{I}(\Psi)$, context element $g \in G(\Psi, x : \mathbb{I})$, and $t \in T(\Psi, G(\mathrm{id}_\Psi, r/x)(g))$, we require an element $c(\Psi, r, s, g, t) \in T(\Psi, G(\mathrm{id}_\Psi, s/x)(g))$. We ask that these satisfy the following properties.

- $T(\psi)(c(\Psi, r, s, g, t)) = c(\Psi', r\psi, s\psi, G(\psi)(g), T(\psi, g)(t))$ for every $\Psi' \Vdash \psi \in \Psi$.
- $c(\Psi, r, r, g, t) = t$.

We similarly define the concept of homogeneous composition operator for a semantic pretype. A *semantic type* is then a triple (T, c, h) consisting of a semantic pretype with accompanying coercion and homogeneous composition operators.

For interpretations of the individual type formers, we refer to [ABCFHL19], which describes a family of models for a cartesian cubical type theory (broadly similar to ours) in settings such as $PSh(\square_c)$. One may also turn to [BCH13; CCHM15; OP18; LOPS18; CMS20] for presheaf-based or presheaf-like models of other variations of cubical type theory.

Part II

Higher inductive types

Chapter 4

Introduction

We now return to the question that motivated our search for contentful equality: defining (effective) quotients in a computational, type-theoretic setting. In fact, we will obtain our quotients as instances of a broader class of constructions, the *higher inductive types* (or *HITs*). Higher inductive types unify the concepts of inductive type and quotient, recognizing that a quotient is generated by a collection of equalities in the same way that an inductive type is generated by a collection of constructors. When elements and equalities are ultimately the same kind of object, as is the case in cubical type theory, these two varieties of generation can be collapsed into instances of a single phenomenon.

Inductive types Before going "higher", let us first recall the concept of inductive type, one of the fundamental building blocks of traditional type theory. An inductive type is one whose elements are generated by some collection of *constructors*, term formers that introduce elements of the type. The type of natural numbers, Nat, for example, is generated by two constructors: zero and suc (for "successor"), the latter of which takes a natural number as input. By "generated", we mean that every element of Nat is obtained by iterating the constructors: we have zero (*i.e.*, 0), suc(zero) (*i.e.*, 1), suc(suc(zero)) (*i.e.*, 2), and so on, but nothing else. Schematically, we might write the specification of Nat as follows.

$$\begin{array}{l}\textbf{inductive}\ \mathsf{Nat}\ \textbf{where}\\ \mid\ \mathsf{zero} \in \mathsf{Nat}\\ \mid\ \mathsf{suc}(n : \mathsf{Nat}) \in \mathsf{Nat}\end{array}$$

The fact that Nat is generated by these constructors is expressed internally by an *eliminator* term, which we can use to construct functions $(n : \mathsf{Nat}) \to D$ by case analysis. The eliminator for Nat corresponds to the principles of primitive recursion and mathematical induction of classical mathematics. To construct a function out of Nat, it suffices to define

$f(0)$ and to define $f(n+1)$ in terms of $f(n)$; to show that a property P holds of all elements of Nat, it suffices to show that $P(0)$ holds and that $P(n)$ implies $P(n+1)$.

An inductive type might be defined relative to some collection of *parameters*; for example, given parameter types $A, B \in \mathsf{U}$, we can form their *coproduct* (*i.e.*, disjoint union, or sum), whose elements are tagged elements of either A or B.

$$
\begin{array}{l}
A : \mathsf{U}, B : \mathsf{U} \gg \textbf{inductive}\ A + B\ \textbf{where} \\
\mid \mathsf{inl}(a : A) \in A + B \\
\mid \mathsf{inr}(b : B) \in A + B
\end{array}
$$

The eliminator for the coproduct expresses that we can construct a function $(c : A + B) \to D$ given functions $(a : A) \to D[\mathsf{inl}(a)/c]$ and $(b : B) \to D[\mathsf{inr}(b)/c]$; from the logical perspective, we can prove a property of elements of $A + B$ by showing that it holds of all elements of the form $\mathsf{inl}(a)$ and all elements of the form $\mathsf{inr}(b)$.

A bit more generally, an inductive type might also take one or more *indices* [CP88; Dyb94], as in the following inductive type of vectors of elements of A of length n. Here A is a parameter, while n is an index.

$$
\begin{array}{l}
A : \mathsf{U} \gg \textbf{inductive}\ \mathsf{Vec}(A, n : \mathsf{Nat})\ \textbf{where} \\
\mid \mathsf{nil} \in \mathsf{Vec}(A, \mathsf{zero}) \\
\mid \mathsf{cons}(n : \mathsf{Nat}, a : A, v : \mathsf{Vec}(A, n)) \in \mathsf{Vec}(A, \mathsf{suc}(n))
\end{array}
$$

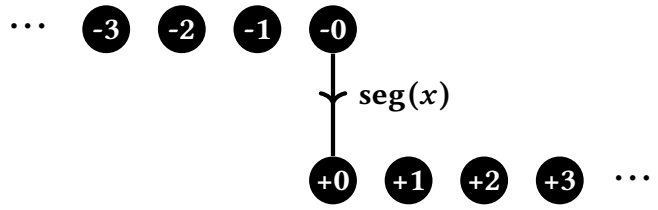
An index is distinguished from a parameter in that the constructors may introduce elements at different indices. In this case, nil constructs the empty vector, which has length zero, while $\mathsf{cons}(n, a, v)$ takes a vector v of length n as input and constructs a vector of length $\mathsf{suc}(n)$ by appending a new element a. By contrast, the parameter A is uniform across all constructors. We call inductive types with indices *indexed inductive types*; they are also known as *inductive families*.

The concept of (indexed) inductive type admits various further generalizations, in particular to *inductive-inductive types* [NS10] and *inductive-recursive types* [Dyb00], which permit the interleaving of multiple inductive and recursive definitions in a certain way. Our objective in this thesis is to generalize in a different direction, to higher inductive types, by adding the ability to declare *path constructors*. We take the class of indexed inductive types as our starting point for this generalization for two reasons. First, some ingenuity is required to give a computational interpretation for inductive types with indices in cubical type theory, in particular to interpret coercion in these types. By contrast, our—admittedly untested—expectation is that the implementation of coercion in indexed inductive types generalizes straightforwardly to inductive-inductive and inductive-recursive types. Second, Martin-Löf's identity type is an indexed inductive type. By interpreting indexed inductive types, we are therefore able to interpret Martin-Löf's intensional type theory (Section 2.2) *en passant*; because cubical type theory also validates the univalence axiom, this makes cubical type theory a constructive interpretation of **HoTT**.

Path constructors Higher inductive types, conceived at a 2011 workshop in Oberwolfach by Bauer, Lumsdaine, Shulman, and Warren, enlarge the class of inductive types by allowing a new form of constructor, the path constructor. Originally proposed for **HoTT** and thus specified in terms of identity types, we describe HITs here in terms of the cubical interval. A path constructor is exactly what it sounds like: a term that constructs paths in an inductive type. As a simple and somewhat contrived first example, we might define the type of *integers* as a type with two ordinary ("point") constructors and one path constructor.

$$\begin{aligned}&\textbf{inductive}\ \mathsf{Int}\ \textbf{where}\\ &\mid \mathsf{neg}(n : \mathsf{Nat}) \in \mathsf{Int}\\ &\mid \mathsf{pos}(n : \mathsf{Nat}) \in \mathsf{Int}\\ &\mid \mathsf{seg}(x : \mathbb{I}) \in \mathsf{Int} \quad [x \equiv 0 \hookrightarrow \mathsf{neg}(\mathsf{zero}) \mid x \equiv 1 \hookrightarrow \mathsf{pos}(\mathsf{zero})]\end{aligned}$$

The first constructor defines a "negative" integer for every natural number, while the second defines a "positive" integer for every natural number. The final constructor expresses that negative zero is the same as positive zero by defining a path between them: a term $x{:}\mathbb{I} \gg \mathsf{seg}(x) \in \mathsf{Int}$ such that $\mathsf{seg}(0) = \mathsf{neg}(\mathsf{zero}) \in \mathsf{Int}$ and $\mathsf{seg}(1) = \mathsf{pos}(\mathsf{zero}) \in \mathsf{Int}$. Pictorially, the elements of Int look something like the following.



In this way, we define Int as the quotient of two copies of Nat by the relation that relates the two zeroes. We think of the point constructors as "zero-dimensional" elements of the type, while path constructors are "one-dimensional" elements; we can more generally consider n-dimensional constructors for $n > 1$, which would construct paths between paths and so on. The only special feature of path constructors, from a specification perspective, is that we should be able to specify their *boundary*, a collection of equations on interval terms at which the constructor should reduce. In the case of $\mathsf{seg}(x)$, these are the requirements that $\mathsf{seg}(0) = \mathsf{neg}(\mathsf{zero})$ and $\mathsf{seg}(1) = \mathsf{pos}(\mathsf{zero})$.

Elimination from a higher inductive type also treats path constructors simply as ordinary constructors with boundary conditions. To define a function $(z : \mathsf{Int}) \to D$, we must explain what to do on each of the constructor. For the negative elements, we need a term $n : \mathsf{Nat} \gg T_{\mathsf{neg}} \in D[\mathsf{neg}(n)/z]$; for the positive elements, a term $n : \mathsf{Nat} \gg T_{\mathsf{pos}} \in D[\mathsf{pos}(n)/z]$. For the path constructor, we require a path $x : \mathbb{I} \gg T_{\mathsf{seg}} \in D[\mathsf{seg}(x)/z]$ such that $T_{\mathsf{seg}}[0/x] = T_{\mathsf{neg}}[0/n]$ and $T_{\mathsf{seg}}[1/x] = T_{\mathsf{pos}}[0/n]$, mimicking the boundary conditions on the path constructor. In other words, we need functions for the positive and negative

cases with a path in D connecting the images of the two zeroes, *i.e.*, functions on the elements that respect the quotienting equality.

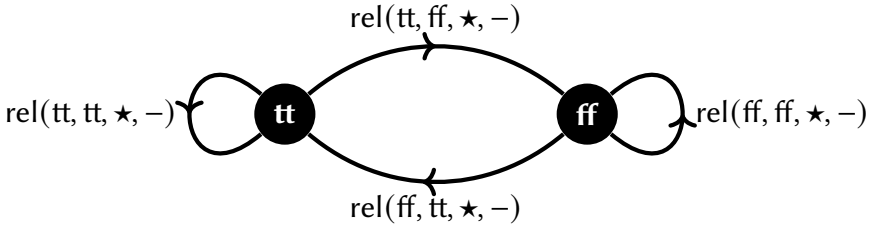
Thinking more generally, we can express the general idea of quotienting by a relation with a higher inductive type that takes the relation as a parameter. Given a type $A \in \mathsf{U}$, a binary relation on A is a family of types $R \in A \times A \to \mathsf{U}$: for each pair $a, a' \in A$, we think of $R\,\langle a, a'\rangle \in \mathsf{U}$ as the type of proofs that a and a' are related by R. Given such a relation, we can define the quotient of A by R as follows.

$$
\begin{array}{l}
A : \mathsf{U}, R : A \times A \to \mathsf{U} \gg \textbf{inductive}\ A /\!\!/ R\ \textbf{where} \\
\mid \mathsf{pt}(a : A) \in A /\!\!/ R \\
\mid \mathsf{rel}(a : A, a' : A, r : R\,\langle a, a'\rangle, x : \mathbb{I}) \in A /\!\!/ R \quad [x \equiv 0 \hookrightarrow \mathsf{pt}(a) \mid x \equiv 1 \hookrightarrow \mathsf{pt}(a')]
\end{array}
$$

The quotient of A by R has a point for every point of A, and draws a path between $\mathsf{pt}(a)$ and $\mathsf{pt}(a')$ whenever there is some $r \in R\,\langle a, a'\rangle$. To define a map out of this type into some D, we specify a map from A to D together with a path between the image of every pair of related elements.

It would seem at first glance that this single higher inductive type is the only one we need, if we are only interested in constructing quotients. The reality is not quite so simple. In a system where equality is contentful, the quotient $A /\!\!/ R$ can behave in ways that are unintuitive to the contentless mind. Although many of the more complex higher inductive types *can* be encoded using only inductive types and $- /\!\!/ -$, we contend that the full generality of higher inductive types is the more natural abstraction in a type theory with contentful equality, as demonstrated by the following example.

The higher structure of cubical types To get a feeling for the potentially surprising behavior of $A /\!\!/ R$, let us consider an example. Suppose that we have a *unit type*, Unit, a type with exactly one element ($\star$), and a *boolean type* Bool, a type with exactly two elements (tt and ff). For any type $A \in \mathsf{U}$, the constant function $\mathsf{Tot}(A) \coloneqq \lambda\_.\,\mathsf{Unit} \in A \times A \to \mathsf{U}$ is the *total relation* on A, which relates each pair of elements $a, a' \in A$ by way of the witness $\star \in \mathsf{Tot}(A)\,\langle a, a'\rangle$. If we take the quotient $\mathsf{Bool} /\!\!/ \mathsf{Tot}(\mathsf{Bool})$, we might expect to get a type which is isomorphic to Unit; we have equated every pair of elements in Bool, after all. Instead, we get a type that looks something like the following picture.



Indeed, we have a path between every pair of elements—but because some of these paths are redundant, we wind up with a type that contains non-trivial loops. This type is *not* isomorphic to Unit; we can even prove as much inside the type theory, using techniques we sketch below. In short, the types of cubical type theory are not characterized purely by their zero-dimensional elements: they have *higher structure*. The zero-dimensional structure of $\mathsf{Bool} /\!\!/ \mathsf{Tot}(\mathsf{Bool})$ matches that of Unit—every point is connected by some path to $\mathsf{pt}(\mathsf{tt})$—but their one-dimensional structures differ.

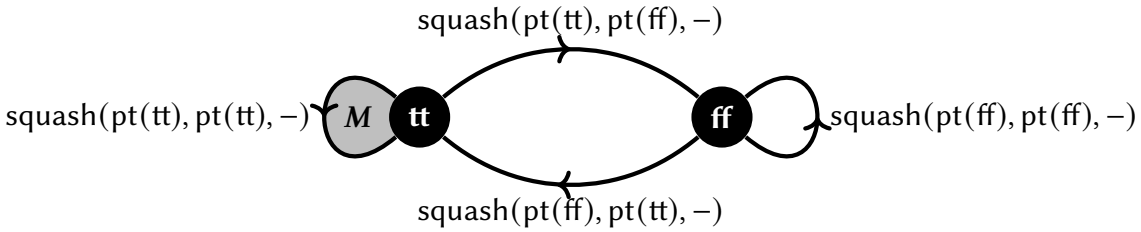
If what we want is an operator on types that collapses the structure of its input, then, quotienting by the total relation will not suffice. Instead, we can take what is called the *propositional truncation* [Uni13, §3.7], a higher inductive type that is not obviously expressible as a quotient.

$$
\begin{array}{l}
A : \mathsf{U} \gg \textbf{inductive } \|A\| \textbf{ where} \\
\mid \mathsf{pt}(a : A) \in \|A\| \\
\mid \mathsf{squash}(t : \|A\|, t' : \|A\|, x : \mathbb{I}) \in \|A\| \quad [x \equiv 0 \hookrightarrow t \mid x \equiv 1 \hookrightarrow t']
\end{array}
$$

Like $A /\!\!/ \mathsf{Tot}(A)$, the propositional truncation contains an element $\mathsf{pt}(a)$ for every $a \in A$. While the former's path constructor identifies every pair of elements coming from A, the latter's identifies every pair of elements of $\|A\|$, that is, every pair of elements in the very type being defined. The effect is that the squash constructor does not only collapse the zero-dimensional structure, but can be used recursively to collapse first the one-dimensional structure, then the two-dimensional structure, and so on. For example, consider the following term $x : \mathbb{I}, y : \mathbb{I} \gg M(x, y) \in \|\mathsf{Bool}\|$.

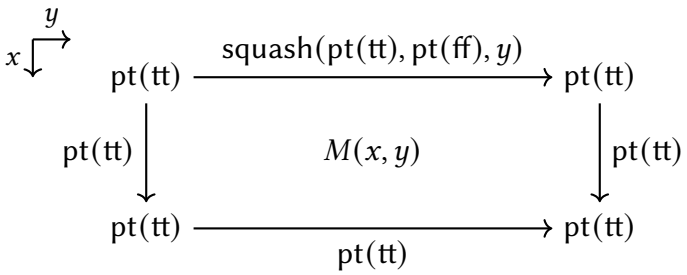
$$M(x, y) \coloneqq \mathsf{squash}(\mathsf{squash}(\mathsf{pt}(\mathsf{tt}), \mathsf{pt}(\mathsf{ff}), y), \mathsf{pt}(\mathsf{tt}), x)$$

When $x = 0$, the outer squash term is equal to its first argument, $\mathsf{squash}(\mathsf{pt}(\mathsf{tt}), \mathsf{pt}(\mathsf{ff}), y)$; when $x = 1$, it is equal to $\mathsf{pt}(\mathsf{tt})$. This term therefore constructs a homotopy, a path between paths, connecting the constant path $\lambda^{\mathbb{I}}\_.\, \mathsf{pt}(\mathsf{tt}) \in \mathsf{Path}(\|\mathsf{Bool}\|, \mathsf{pt}(\mathsf{tt}), \mathsf{pt}(\mathsf{tt}))$ and the "redundant" loop $\lambda^{\mathbb{I}} y.\, \mathsf{squash}(\mathsf{pt}(\mathsf{tt}), \mathsf{pt}(\mathsf{ff}), y)$ of the same type. Pictorially, M "fills in" the loop at $\mathsf{pt}(\mathsf{tt})$.



Similar applications of squash fill in the other two holes in this picture. A bit more abstractly, we can visualize M as a square varying in the two axes x and y, with edges and

vertices given by the terms shown below. (In this case, three of the four edges are actually degenerate.)



Of course, the squash constructor also creates many redundant homotopies, just as it creates redundant loops, but these too can be filled in by further iterations of squash. In the end, $\|\mathsf{Bool}\|$ does turn out to be isomorphic to Unit, as is $\|A\|$ for any type that contains at least one element. (If A is empty, then so too is $\|A\|$).

In truth, propositional truncations—and many other HITs, though not all [LS20, §9]—can be indirectly obtained by constructions that rely only the quotient HIT [Doo16; Kra16; Rij17, §7]. However, these constructions are fairly involved; while it is useful to know they are possible, providing general higher inductive types as primitive is much more convenient for the cubical programmer.[1]

The upshot of this example is that identifying elements of a type—a process cleanly accomplished by quotienting in classical mathematics—is a more delicate business when equality is contentful, because we must consider *how* we identify those elements. We can think of this as an acceptable cost of univalence and (what we will see to be) effective quotients. That cost can be mitigated; for one, we can define a *set truncation* HIT that collapses the *higher* structure of a type in the same way that $\|-\|$ collapses *all* structure, and use this to destroy any higher-dimensional structure we inadvertantly create.[2] However, we can also see the higher structure of cubical type theory as a benefit in and of itself, allowing us to use type theory as a language for higher-dimensional mathematics.

Synthetic homotopy theory Algebraic topology and homotopy theory are closely related mathematical fields that study objects carrying higher structure: their properties, techniques for classifying them, and so on. Right from the origins of higher-dimensional

[1] Also, the computation rules for path constructors in these encodings often only hold up to path equality, whereas we can get exact equalities with a direct construction.

[2] Taking a different tack along these lines, the type theories **OTT** [AMS07] and **XTT** [SAG19] include contentful equality but nevertheless require that all types have trivial higher structure: proofs of equalities do carry computational content, but all such proofs are interchangeable. Univalent universes do not fit into this paradigm, but effectivity of quotients can be obtained.

type theories—Awodey and Warren's work on modeling identity types with weak factorization systems [War08; AW09]—a guiding motivation has been their potential as a language for proving results in topology and homotopy theory. This program, labelled *synthetic homotopy theory*, took center stage in the seminal **HoTT** Book [Uni13], and has continued to expand since, even generating new classical results [FFLL16; ABFJ20].

To get a taste of the kind of topological results we can state and prove in cubical type theory, let us consider a simple HIT with non-trivial higher structure: a *circle*.

inductive Circle **where**
| base ∈ Circle
| loop($x : \mathbb{I}$) ∈ Circle [$x \equiv 0 \hookrightarrow$ base | $x \equiv 1 \hookrightarrow$ base]

This circle has a single base point and a single loop at that point, that is, a path from that point to itself. One way we can analyze the circle is by characterizing the type Path(Circle, base, base) of paths from the base point to itself; this is the *fundamental group* of (Circle, base).[3] Calculating fundamental groups is one of the most basic tools for classifying spaces in algebraic topology; for example, we can see that Circle is not isomorphic to Unit or to Circle × Circle by comparing their fundamental groups.

We certainly expect there to be at least two paths from base to base: the constant path, $\lambda^{\mathbb{I}} x.$ base, and the path given by the loop constructor, $\lambda^{\mathbb{I}} x.$ loop(x). In fact, because paths (like equalities) are invertible and composable, there are integer-many paths: we wind up with an isomorphism Path(Circle, base, base) ≃ Int. The path that goes around the loop "forward" n times corresponds to the positive integer n, while the path that goes around the loop "backward" n times corresponds to $-n$.

We will not give a proof of this isomorphism here—see [LS13] or [Uni13, §8.1]—but we do want to call attention to one aspect. To construct the isomorphism, we must of course be able to define a function Path(Circle, base, base) → Int. That is, we must be able to *extract data* (an integer) *from a path*. Thus we arrive again at the problem at the root of effectivity of quotients. This time, with contentful equality in hand, we will be able to solve it.

Descent and effectivity Both the characterization of Path(Circle, base, base) and effectivity of quotients depend on a more fundamental *descent* property, a concept which originates in category theory and arises in cubical type theory from a combination of univalence and the ability to define types by case analysis.

Observe that we have an isomorphism $I \in \mathsf{Int} \simeq \mathsf{Int}$ whose forward map sends the integer n to its successor $n + 1$. By univalence, this isomorphism induces a path in the

[3] A bit more precisely, this is the *loop space*, while the fundamental group is the set truncation $\|\mathsf{Path}(\mathsf{Circle}, \mathsf{base}, \mathsf{base})\|_0$ of this type. In this case, the two are isomorphic, as the circle has no structure above dimension one.

universe, which we will call $\mathsf{UA}\,I \in \mathsf{Path}(\mathsf{U}, \mathsf{Int}, \mathsf{Int})$. By case analysis, we can use this path to define a function $\mathsf{Code} \in \mathsf{Circle} \to \mathsf{U}$ as follows.

$$\mathsf{Code}\,c \coloneqq \left[\begin{array}{l} \textbf{case } c \textbf{ of} \\ \mid \mathsf{base} \mapsto \mathsf{Int} \\ \mid \mathsf{loop}(x) \mapsto \mathsf{UA}\,I\,x \end{array}\right]$$

That is, we draw a picture of a circle in U by choosing the type Int as our point and $\mathsf{UA}\,I$ as our loop at that point. The transformation of the data (Int, I) into a function $\mathsf{Circle} \to \mathsf{U}$, instrumented by the circle eliminator, is the aforementioned descent.

Now suppose we have an arbitrary path $p \in \mathsf{Path}(\mathsf{Circle}, \mathsf{base}, \mathsf{base})$. By applying Code to this path pointwise, we obtain a path $\lambda^{\mathbb{I}} x.\, \mathsf{Code}\,(p\,x) \in \mathsf{Path}(\mathsf{U}, \mathsf{Int}, \mathsf{Int})$. We define our candidate integer corresponding to p by coercing $0 \in \mathsf{Int}$ along this path.

$$\mathsf{encode}\,p \coloneqq \mathsf{coe}^{0\to 1}_{x.\mathsf{Code}\,(p\,x)}(0) \in \mathsf{Int}$$

This gives us an integer; is it the integer we want? If we apply encode to the constant path, we have a coercion along a constant path, which we can show corresponds to the identity function. (We henceforth use $\rightsquigarrow$ as an informal infix notation for paths.)

$$\mathsf{encode}\,(\lambda^{\mathbb{I}} x.\,\mathsf{base}) = \mathsf{coe}^{0\to 1}_{x.\mathsf{Int}}(0) \rightsquigarrow 0 \in \mathsf{Int}$$

On the other hand, if we supply $\lambda^{\mathbb{I}} x.\,\mathsf{loop}(x)$, coercion on a path formed by univalence transforms into an application of the underlying isomorphism, which sends n to $n+1$.

$$\mathsf{encode}\,(\lambda^{\mathbb{I}} x.\,\mathsf{loop}(x)) = \mathsf{coe}^{0\to 1}_{x.\mathsf{UA}\,I\,x}(0) \rightsquigarrow 0 + 1 = 1 \in \mathsf{Int}$$

So encode does, at least, distinguish between the constant and single loop paths. Although we cannot inspect its behavior further without getting into the nature of composition and inversion of paths, suffice to say that we indeed have $\mathsf{encode}\,(\lambda^{\mathbb{I}} x.\,\mathsf{loop}^n(x)) \rightsquigarrow n \in \mathsf{Int}$ for every integer n, where loop^n is an n-fold composition of the loop constructor.

By the same technique, we can extract witnesses from paths in our example of a quotient from Chapter 1, the integers modulo n. Recall that we wanted Int_n to be the quotient of Int by the following relation.

$$m_0 \approx m_1 \coloneqq (p : \mathsf{Int}) \times \mathsf{Id}(\mathsf{Int}, m_1 - m_0, p \cdot n)$$

We can write a definition of Int_n as a higher inductive type as follows.

$$\begin{array}{l} n : \mathsf{Nat} \gg \textbf{inductive } \mathsf{Int}_n \textbf{ where} \\ \mid \mathsf{int}(m : \mathsf{Int}) \in \mathsf{Int}_n \\ \mid \mathsf{mod}(m : \mathsf{Int}, x : \mathbb{I}) \in \mathsf{Int}_n \quad [x \equiv 0 \hookrightarrow \mathsf{int}(m) \mid x \equiv 1 \hookrightarrow \mathsf{int}(m+n)] \end{array}$$

This is a bit different from the naive quotient $\mathsf{Int} /\!\!/ \approx$: to avoid introducing redundant higher structure, we only construct paths $m \rightsquigarrow m + n$, not paths $m \rightsquigarrow m + p \cdot n$ for all $p : \mathsf{Int}$. The latter are instead obtained by interated composition of the mod constructor: $m \rightsquigarrow m + 1 \cdot n \rightsquigarrow \cdots \rightsquigarrow m + (p-1) \cdot n \rightsquigarrow m + p \cdot n$.

For any $m_0, m_1 \in \mathsf{Int}$, we have an isomorphism $I\, m_0\, m_1 \in (m_0 \approx m_1) \simeq (m_0 \approx m_1 + n)$: if m_0 and m_1 differ by a multiple of n, then so do m_0 and $m_1 + n$, and vice versa. Using this, we can define a type family $\mathsf{Code} \in \mathsf{Int} \to \mathsf{Int}_n \to \mathsf{U}$ that takes m_0 and $\mathsf{int}(m_1)$ to $m_0 \approx m_1$ by case analysis.

$$\mathsf{Code}\, m_0\, t_1 \coloneqq \left[\begin{array}{l} \textbf{case}\ t_1\ \textbf{of} \\ \mid \mathsf{int}(m_1) \mapsto m_0 \approx m_1 \\ \mid \mathsf{mod}(m_1, x) \mapsto \mathsf{UA}\,(I\, m_0\, m_1)\, x \end{array}\right]$$

Now, given any path $p \in \mathsf{Path}(\mathsf{Int}, \mathsf{int}(m_0), \mathsf{int}(m_1))$, we can extract a proof of $m_0 \approx m_1$ by coercing the element $\langle 0, P\rangle \in m_0 \approx m_0$ (where P is some proof that $m_0 - m_0 \rightsquigarrow 0 \cdot n$) along the line of types obtained by applying $\mathsf{Code}\ m_0$ to p pointwise.

$$\mathsf{encode}\, p \coloneqq \mathsf{coe}^{0\to 1}_{x.\mathsf{Code}\, m_0\, (p\, x)}(\langle 0, P\rangle) \in m_0 \approx m_1$$

So we have $\mathsf{encode} \in \mathsf{Path}(\mathsf{Int}, \mathsf{int}(m_0), \mathsf{int}(m_1)) \to m_0 \approx m_1$. With a bit more coding, we can show that encode is even an isomorphism; Int_n is an effective quotient of Int by $- \approx -$.

The computational content of paths is essential to this argument: Code examines the content of path constructors to convert them into univalence-wrapped isomorphisms, and coercion in turn inspects its input type line to extract an isomorphism and apply it.

Outline In the following chapters, we realize the promise of higher inductive types sketched above, defining a class of specifications that include such types as Int_n and $\|A\|$ and explaining each such specification as a computational object in a cubical type theory.

In Chapter 5, we begin by considering a number of representative examples of higher inductive types in more detail, exploring in particular how to implement coercion for each. In addition to proper higher inductive types, we also consider indexed inductive types; implementing coercion for these requires similar techniques despite the absence of explicit higher structure.

Chapter 6 is the meat of this part: we define a schema for specifying indexed higher inductive types, show that we can construct type systems closed under these types, prove that they support coercion and composition, and formulate and prove their introduction and elimination principles.

We close with a discussion of related and future work in Chapter 7.

Chapter 5

Case studies

A reasonably complete theory of higher inductive types must accommodate a number of features, each of which introduces its own complications. Fortunately, each of these features can be examined in isolation. In this chapter, we consider a collection of representative examples that highlight those features in turn. To get a computational interpretation off the ground, the main problem we need to solve is the interpretation of the Kan operations, coercion and composition. Finding solutions to that problem will in turn require us to think carefully about what the values of a higher inductive type should be, and how an eliminator should behave when applied to each value.

5.1 Quotients and pushouts

Integers modulo two Our most basic examples of higher inductive types are quotients. As a first cut, then, let us consider a single concrete instance of a quotient, the integers modulo 2. (We will suppose that we already have an integer type, which could be implemented in any number of ways.)

$$\begin{array}{l}\textbf{inductive}\ \mathsf{Int}_2\ \textbf{where}\\ \mid \mathsf{int}(m : \mathsf{Int}) \in \mathsf{Int}_2\\ \mid \mathsf{mod}(m : \mathsf{Int}, x : \mathbb{I}) \in \mathsf{Int}_2 \quad [x \equiv 0 \hookrightarrow \mathsf{int}(m) \mid x \equiv 1 \hookrightarrow \mathsf{int}(m+2)]\end{array}$$

Note that this definition gives us the free (higher-dimensional) equivalence relation generated by the path constructors: we only put in paths $\mathsf{int}(m) \rightsquigarrow \mathsf{int}(m+2)$, but we should be able to derive paths $\mathsf{int}(m) \rightsquigarrow \mathsf{int}(m+2\cdot p)$ from the general properties of path equality.

A first interesting property of higher inductive specifications is that a constructor may depend on previous constructors: we cannot state the boundary of mod without knowledge of the int. By contrast, the constructors of an ordinary indexed inductive type are always independent of each other.

To give a computational definition of this type, we have a number of pieces to assemble: its operational semantics (including both the constructors and eliminator), the relation named by Int_2, and its coercion and composition operators. To start with, it is at least clear that the constructors of the inductive type should be values, with the exception that the boundary of a path constructor should reduce as specified.

$$\frac{}{\mathsf{int}(M)\ \mathsf{val}} \qquad \frac{}{\mathsf{mod}(M,x)\ \mathsf{val}} \qquad \frac{}{\mathsf{mod}(M,0) \longmapsto \mathsf{int}(M)} \qquad \frac{}{\mathsf{mod}(M,1) \longmapsto \mathsf{int}(M+2)}$$

Naively, we might take the terms $\mathsf{int}(M)$ and $\mathsf{mod}(M,x)$ as the sole values of Int_2. For this to be sensible, however, we need to check that the Kan operations are definable. Because we have restricted our attention to a concrete HIT with no parameters, we have no problem with coercion: the only line of the form $x.\mathsf{Int}_2$ is the constant line, so we can define coercion across it as the identity function.

$$\frac{}{\mathsf{coe}^{r\to s}_{x.\mathsf{Int}_2}(M) \longmapsto M}$$

It is with composition that we find ourselves in hot water. Recall that hcom in particular implements symmetry and transitivity of the path relation. In order for these to be implementable in $\mathsf{int}(M)$, they must hold on the level of values; for one, given any value $x : \mathbb{I} \gg V \in \mathsf{Int}_2$, there should be some inverse value $x : \mathbb{I} \gg W \in \mathsf{Int}_2$ with $W[0/x] = V[1/x] \in \mathsf{Int}_2$ and $W[1/x] = V[0/x] \in \mathsf{Int}_2$. But this clearly fails for our choice of values: we always have a path $\mathsf{int}(M) \rightsquigarrow \mathsf{int}(M+2)$, but there is no value that provides a path $\mathsf{int}(M+2) \rightsquigarrow \mathsf{int}(M)$. Likewise, our selection of values fails to be transitive, there being no paths $\mathsf{int}(M) \rightsquigarrow \mathsf{int}(M+4)$ or $\mathsf{int}(M) \rightsquigarrow \mathsf{int}(M+6)$.

We therefore have no choice but to revise our choice of values, adding new terms to stand for these values obtained by composition. Of course, we need to account not only for the special cases of symmetry and transitivity, but for all kinds of composition. We therefore introduce *formal composite* values, fhcom, that provide composites for every possible composition problem.

$$\frac{r \neq s \qquad (\nexists i)\ \xi_i\ \mathsf{satisfied}}{\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})\ \mathsf{val}} \qquad \frac{(\nexists i)\ \xi_i\ \mathsf{satisfied}}{\mathsf{fhcom}^{r\to r}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto M}$$

$$\frac{(\nexists i < k)\ \xi_i\ \mathsf{satisfied} \qquad \xi_k\ \mathsf{satisfied}}{\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto N_k[s/x]}$$

The reduction rules for fhcom line up exactly with the equations imposed in the definition of composition (Definition 3.1.27). By adding fhcom values to our definition of the

relation for Int_2, we can use them to define composition in Int_2.

$$\frac{}{\mathsf{hcom}^{r\to s}_{\mathsf{Int}_2}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto \mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})}$$

Translating this into a precise definition in the framework of Section 3.1, we would define the $\cdot$-relation $[\![\mathsf{Int}_2]\!]$ to be the least closed under the following three principles.

- $\mathsf{int}(M) \approx \mathsf{int}(M') \in [\![\mathsf{Int}_2]\!]\langle\psi\rangle$ for all $\Psi \Vdash \psi \in \cdot$ and $\Psi \Vdash M = M' \in \mathsf{Int}$.
- $\mathsf{mod}(M, x) \approx \mathsf{mod}(M', x) \in [\![\mathsf{Int}_2]\!]\langle\psi\rangle$ for all $\Psi \Vdash \psi \in \cdot$, $x \in \Psi$, and $\Psi \Vdash M = M' \in \mathsf{Int}$.
- $\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx \mathsf{fhcom}^{r\to s}('M; \overrightarrow{\xi'_i \hookrightarrow x.N'_i}) \in [\![\mathsf{Int}_2]\!]\langle\psi\rangle$ whenever the arguments are well-formed and equal composition problems in $\Downarrow[\![\mathsf{Int}_2]\!]\langle\psi\rangle$, as specified in Definition 3.1.27.

In short, we are now defining Int_2 as freely generated by the constructors *together with their composites*.

In order to justify throwing new values into the definition of Int_2, however, we must check that we can still define the expected eliminator for the higher inductive type. That is, we must be able to construct a function $(a : \mathsf{Int}_2) \to D$ given a clause for each constructor as follows.

- $m : \mathsf{Int} \gg T_{\mathsf{int}} \in D[\mathsf{int}(m)/a]$.
- $m : \mathsf{Int}, x : \mathbb{I} \gg T_{\mathsf{mod}} \in D[\mathsf{mod}(m, x)/a]$ satisfying
  - $m : \mathsf{Int} \gg T_{\mathsf{mod}}[0/x] = T_{\mathsf{int}} \in D[\mathsf{int}(m)/a]$
  - $m : \mathsf{Int} \gg T_{\mathsf{mod}}[1/x] = T_{\mathsf{int}}[m + 2/m] \in D[\mathsf{int}(m)/a]$.

Given these and an element $M \in \mathsf{Int}_2$, let us use the syntax $\mathsf{elim}(a.D; M; m.T_{\mathsf{int}}, m.x.T_{\mathsf{mod}})$ for the application of the eliminator with these clauses to M. It is clear how the eliminator should evaluate when applied to constructor values: simply step to the appropriate provided clause.

$$\frac{}{\mathsf{elim}(a.D; \mathsf{int}(M); m.T_{\mathsf{int}}, m.x.T_{\mathsf{mod}}) \longmapsto T_{\mathsf{int}}[M/m]}$$

$$\frac{}{\mathsf{elim}(a.D; \mathsf{mod}(M, y); m.T_{\mathsf{int}}, m.x.T_{\mathsf{mod}}) \longmapsto T_{\mathsf{mod}}[M/m, y/x]}$$

The boundary equations on T_{mod} will ensure that this definition evaluates coherently on a mod term, as is necessary to establish well-typedness.

But how should the eliminator evaluate on a formal composite? We are rescued by the assumption that $a : \mathsf{Int}_2 \gg D$ type—in particular, that D itself supports coercion and composition. We may therefore evaluate an eliminator applied to a composite by stepping to a composite of the same shape in the target type. For example, if we apply the eliminator to the inverse of a path P, the result will be the inverse of the same eliminator applied to P; if we apply it to the composition of two paths P and Q, the result will be the composite of the eliminator applied to P and the eliminator applied to Q.

Formally, this intention is expressed by the following operational semantics rule, which uses the heterogeneous composition operator com introduced in Definition 3.1.30.

$$\frac{r \neq s \qquad (\nexists i)\ \xi_i \text{ satisfied} \qquad D_y \coloneqq D[\mathsf{fhcom}^{r\to y}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})/a]}{\begin{array}{c}\mathsf{elim}(a.D; \mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}); m.T_{\mathsf{int}}, m.x.T_{\mathsf{mod}}) \\ \longmapsto \\ \mathsf{com}^{r\to s}_{y.D_y}(\mathsf{elim}(a.D; M; m.T_{\mathsf{int}}, m.x.T_{\mathsf{mod}}); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{elim}(a.D; N_i; m.T_{\mathsf{int}}, m.x.T_{\mathsf{mod}})})\end{array}}$$

In words, by applying the eliminator to each component of the composition problem, we obtain a composition problem in D. In particular, this problem lies over the filler $y.\mathsf{fhcom}^{r\to y}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})$: at $r = y$ we have $\mathsf{elim}(a.D; M; m.T_{\mathsf{int}}, m.x.T_{\mathsf{mod}})$ in $D[M/a]$, while at each ξ_i we have $y.\mathsf{elim}(a.D; N_i; m.T_{\mathsf{int}}, m.x.T_{\mathsf{mod}})$ in $y.D[N_i/a]$. The composite of these terms is then a term in $D[\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})/a]$ as required. The evaluation is moreover coherent in D when the inputs are well-typed, thanks to the equations com is required to satisfy. To summarize, because the composites in Int_2 are freely generated and any target type has its own composites, we have a canonical way to extend any function covering the constructors to also cover formal composites.

Remark 5.1.1. We motivated our need to introduce homogeneous composites with the observation that the elements introduced by mod do not otherwise satisfy symmetry or transitivity. It may be tempting, therefore, to instead simply require that the data for a path constructor specifies an equivalence relation, disallowing mod but allowing a constructor $\mathsf{mod}'(m, p, -)$ that connects $\mathsf{int}(m) \rightsquigarrow \mathsf{int}(m + 2 \cdot p)$ for each integer p. However, such a definition will also fail to be Kan "one dimension up". For example, path induction implies the existence of the following square not derivable from mod′ alone.

$$\begin{array}{ccc} & \overset{y}{\nearrow} & \\ x \downarrow & & \\ \mathsf{int}(m) & \xrightarrow{\mathsf{mod}'(m,1,y)} & \mathsf{int}(m+2) \\ \Big\downarrow{\scriptstyle \mathsf{mod}'(m,1,x)} & & \Big\downarrow{\scriptstyle \mathsf{int}(m+2)} \\ \mathsf{int}(m+2) & \xrightarrow[\mathsf{int}(m+2)]{} & \mathsf{int}(m+2) \end{array}$$

Indeed, we would need to require that the input relation is an ∞-*equivalence relation* [Rij18, Chapter 13], a serious burden to place on the programmer.

General relations We have now seen how to handle a single concrete higher inductive type, in particular how to interpret composition in such a type. The next step is to consider a higher inductive type with parameters, in which case we will also have a non-trivial coercion operation to implement. Let us start with a particularly simple case: quotienting by a type-valued relation, as introduced in Chapter 4.

$$A : \mathsf{U}, R : A \times A \to \mathsf{U} \gg \textbf{inductive}\ A /\!\!/ R\ \textbf{where}$$
$$\mid \mathsf{pt}(a : A) \in A /\!\!/ R$$
$$\mid \mathsf{rel}(a : A, a' : A, r : R\,\langle a, a'\rangle, x : \mathbb{I}) \in A /\!\!/ R \quad [x \equiv 0 \hookrightarrow \mathsf{pt}(a) \mid x \equiv 1 \hookrightarrow \mathsf{pt}(a')]$$

As before, the constructors alone are not sufficient to interpret composition. We do not know that R is symmetric or transitive, for example. (Indeed, the higher inductive type produced is really the quotient of A by the higher equivalence relation generated by R, which is captured by the addition of free composites.) Accordingly, we once again implement composition with formal composites; the addition of parameters has no effect here.

For coercion, consider the situation where we have a line of types $x.(A /\!\!/ R)$, made up of lines $x : \mathbb{I} \gg A \in \mathsf{U}$ and $x : \mathbb{I} \gg R \in A \times A \to \mathsf{U}$ in the parameters. Given some element $M \in A[r/x] /\!\!/ R[r/x]$ at some $r \in \mathbb{I}$, we are charged with producing an element of $A[s/x] /\!\!/ R[s/x]$ for any other $s \in \mathbb{I}$. The natural course of action is to do so by case analysis on the value of M. We begin by evaluating the argument of the coercion to a value.

$$\frac{M \longmapsto M'}{\mathsf{coe}^{r\to s}_{x.A/\!\!/R}(M) \longmapsto \mathsf{coe}^{r\to s}_{x.A/\!\!/R}(M')}$$

Once we arrive at a value V, we inspect it. If V is a point $\mathsf{pt}(N)$ where $A : A[r/x]$, for example, we can coerce by coercing the inner argument along $x.A$ and then repackaging it with pt.

$$\frac{}{\mathsf{coe}^{r\to s}_{x.A/\!\!/R}(\mathsf{pt}(N)) \longmapsto \mathsf{pt}(\mathsf{coe}^{r\to s}_{x.A}(N))}$$

The output of the inner coercion has type $A[s/x]$, and so the reduct is indeed of type $A[s/x] /\!\!/ R[s/x]$. The same works for the path constructor: coerce each argument and

repackage.

$$\frac{}{\begin{array}{c}\mathsf{coe}^{r\to s}_{x.A/\!\!/R}(\mathsf{rel}(N, N', P, y)) \\ \longmapsto \\ \mathsf{rel}(\mathsf{coe}^{r\to s}_{x.A}(N), \mathsf{coe}^{r\to s}_{x.A}(N'), \mathsf{coe}^{r\to s}_{x.R\langle \mathsf{coe}^{r\to y}_{x.A}(N), \mathsf{coe}^{r\to y}_{x.A}(N')\rangle}(P), y)\end{array}}$$

While the above looks a bit more complicated thanks to the dependency of the type of P on N and N', the essential idea is the same. As required, the reduction is coherent: if, for example, we reduce and then instantiate y with 0, the result steps to $\mathsf{pt}(\mathsf{coe}^{r\to s}_{x.A}(N))$, which is the same as the result we would arrive at by instantiating y with 0 and then reducing.

Finally, we must consider coercion on a formal composite. In this case, the idea is the same as with the eliminator. A coercion applied to a formal composite becomes a composite of coercions in the target type—which is to say, another formal composite.

$$\frac{}{\mathsf{coe}^{r\to s}_{x.A/\!\!/R}(\mathsf{fhcom}^{t\to u}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})) \longmapsto \mathsf{fhcom}^{t\to u}(\mathsf{coe}^{r\to s}_{x.A/\!\!/R}(M); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{coe}^{r\to s}_{x.A/\!\!/R}(N_i)})}$$

Again, it is easy to see that this is a coherent definition, and it completes our specification of coercion in the quotient type.

Note that we do not have the option of defining coercion in the quotient type by simply introducing formal coercions, as we did with composition—at least, not without imposing severe restrictions on the parameters. The problem is that, unlike composition, coercion moves between different instantiations of the parameters of a type. If we introduced formal coercion values, the values of a given $A /\!\!/ R$ would depend on the values of $A' /\!\!/ R'$ for every A', R' connected by paths to A, R; as such, we would have to define the semantics of the quotient type for every possible parameter instantiation simultaneously. This dependency on the complete parameter space precludes including $A /\!\!/ R$ in the same universe as its type parameters; given $A \in \mathsf{U}_n$ and $R \in A \times A \to \mathsf{U}_n$, we would not have $A /\!\!/ R \in \mathsf{U}_n$ but only $A /\!\!/ R \in \mathsf{U}_{n+1}$. This issue was encountered by Lumsdaine and Shulman in their semantics of higher inductive types in simplicial model categories, where it manifests as the failure of a fibrant replacement operation to preserve size. It is thus an important feature of the cubical setting that the Kan operations can be divided into homogeneous composition, which can be added formally in HITs, and coercion, which is definable in HITs. In the simplicial setting, there is instead only an operation analogous to com which is not known to decompose in such a way.[1]

Elimination, meanwhile, is not complicated by the addition of parameters; we may state the elimination principle and define its computational behavior in the same way as

[1]Shulman has recently proposed an alternative (yet unpublished) technique for realizing higher inductive types in the simplicial case.

in the concrete case, arriving at the following typing rule.

$$\frac{\begin{array}{c} q : A /\!\!/ R \gg D\ \mathsf{type} \qquad M \in A /\!\!/ R \qquad a : A \gg T_{\mathsf{pt}} \in D[\mathsf{pt}(a)/q] \\ a : A, a' : A, r : R\,\langle a, a'\rangle, x : \mathbb{I} \gg T_{\mathsf{rel}} \in D[\mathsf{rel}(a, a', r, x)/q] \\ a : A, a' : A, r : R\,\langle a, a'\rangle \gg T_{\mathsf{rel}}[0/x] = T_{\mathsf{pt}} \in D[\mathsf{pt}(a)/q] \\ a : A, a' : A, r : R\,\langle a, a'\rangle \gg T_{\mathsf{rel}}[1/x] = T_{\mathsf{pt}}[a'/a] \in D[\mathsf{pt}(a')/q] \end{array}}{\mathsf{elim}(q.D; M; a.T_{\mathsf{pt}}, a.a'.r.x.T_{\mathsf{rel}}) \in D[M/q]}$$

Pushouts The quotient type is actually a deceptively simple example of a parameterized higher inductive type, at least as far as coercion is concerned. It has the special property that the naive definition of coercion for the path constructor—coerce each argument and repackage—produces coherent results. This is not the case in general, as demonstrated by our next example: the *pushout* [Uni13, §6.8].

The pushout of a pair of maps $F \in C \to A$ and $G \in C \to B$ is the coproduct of A and B "modulo C": it has an element $\mathsf{inl}(a)$ for every $a : A$ and an element $\mathsf{inr}(b)$ for every $b : B$, but also identifies $\mathsf{inl}(F\,c)$ with $\mathsf{inr}(G\,c)$ for every $c : C$.

$$\begin{array}{l} A, B, C : \mathsf{U}, f : C \to A, g : C \to B \gg \textbf{inductive}\ \mathsf{Push}(A, B, C, f, g)\ \textbf{where} \\ \mid \mathsf{inl}(a : A) \in \mathsf{Push}(A, B, C, f, g) \\ \mid \mathsf{inr}(b : B) \in \mathsf{Push}(A, B, C, f, g) \\ \mid \mathsf{push}(c : C, x : \mathbb{I}) \in \mathsf{Push}(A, B, C, f, g) \quad [x \equiv 0 \hookrightarrow \mathsf{inl}(f\,c) \mid x \equiv 1 \hookrightarrow \mathsf{inr}(g\,c)] \end{array}$$

The notable feature of this definition, in comparison to what we have seen before, is that the boundary of the path constructor push depends on the parameters f, g to the type. (Strictly speaking, push should be annotated with f and g so that these are available for the boundary reductions in the operational semantics, but we will suppress such annotations here for readability.)

As with the quotient, we can define composition in $\mathsf{Push}(A, B, C, F, G)$ using formal composites, and we can define coercion on the point constructors by moving the coercion inside the constructor.

$$\frac{}{\mathsf{coe}^{r\to s}_{x.\mathsf{Push}(A,B,C,F,G)}(\mathsf{inl}(M)) \longmapsto \mathsf{inl}(\mathsf{coe}^{r\to s}_{x.A}(M))}$$

$$\frac{}{\mathsf{coe}^{r\to s}_{x.\mathsf{Push}(A,B,C,F,G)}(\mathsf{inr}(N)) \longmapsto \mathsf{inr}(\mathsf{coe}^{r\to s}_{x.B}(N))}$$

It is tempting to do the same for the path constructor.

$$\frac{}{\mathsf{coe}^{r\to s}_{x.\mathsf{Push}(A,B,C,F,G)}(\mathsf{push}(P, y)) \longmapsto \mathsf{push}(\mathsf{coe}^{r\to s}_{x.C}(P), y)}\ \text{✘}$$

Looking closely at this definition, however, we notice that it fails to be coherent. If we instantiate y with 0, the left side will reduce to $\mathsf{coe}^{r\to s}_{x.\mathsf{Push}(A,B,C,F,G)}(\mathsf{inl}(F[r/x]\,P))$,

which in turn reduces to $\mathsf{inl}(\mathrm{coe}^{r\to s}_{x.A}(F[r/x]\ P))$ by our rules for coercion of point constructors. If, on the other hand, we instantiate y with 0 on the *right* side, we instead obtain $\mathsf{inl}(F[s/x]\ (\mathrm{coe}^{r\to s}_{x.C}(P)))$. That is, depending on our ordering of reduction and interval substitution, we apply F and coercion in different orders. Nothing guarantees that $\mathrm{coe}^{r\to s}_{x.A}(F[r/x]\ P)$ and $F[s/x]\ (\mathrm{coe}^{r\to s}_{x.C}(P))$ will be equal as elements of $A[s/x]$, and so this reduction rule fails to give us a well-typed coercion operator.[2]

Fortunately, although we have a mismatch up to *exact* equality, these two terms *are* the same up to path equality, which means we will be able to correct the definition using composition. Consider the following term varying in $x : \mathbb{I}$.

$$x : \mathbb{I} \gg \mathrm{coe}^{x\to s}_{x.A}(F\ (\mathrm{coe}^{r\to x}_{x.C}(P))) \in A$$

When we instantiate x with r, the inner coercion vanishes, and so the term simplifies to $\mathrm{coe}^{r\to s}_{x.A}(F[r/x]\ P)$. Conversely, when we instantiate x with s, the outer coercion disappears and we are left with $F[s/x]\ (\mathrm{coe}^{r\to s}_{x.C}(P))$. We can use this adjustment path, together with the corresponding path for G, as the tube of a formal composition that "fixes" the boundary of our naive definition.

$$\frac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{\mathrm{coe}^{r\to s}_{x.\mathsf{Push}(A,B,C,F,G)}(\mathsf{push}(P,y)) \longmapsto \mathsf{fhcom}^{s\to r}\left(\mathsf{push}(\mathrm{coe}^{r\to s}_{x.C}(P), y);\ \begin{array}{l} y \equiv 0 \hookrightarrow x.\mathsf{inl}(\mathrm{coe}^{x\to s}_{x.A}(F\ (\mathrm{coe}^{r\to x}_{x.C}(P)))) \\ y \equiv 1 \hookrightarrow x.\mathsf{inr}(\mathrm{coe}^{x\to s}_{x.B}(G\ (\mathrm{coe}^{r\to x}_{x.C}(P)))) \end{array}\right)}\ \checkmark$$

This rectified definition satisfies the coherence conditions we require, and still simplifies to $\mathsf{push}(P, y)$ when $r = s$. This shape of coercion implementation—with coherence ensured by formal composition—will suffice for all *non-indexed* higher inductive types. For indexed higher inductive types (indeed, for indexed inductive types more generally), another adjustment will be necessary, as we will see in Section 5.3.

5.2 Truncations

For our next class of examples, we examine the role of recursive constructors by considering the propositional truncation and more generally the *higher truncations*. These

[2]Indeed, in order for this condition to hold in general, we would need exact uniqueness of identity proofs in A. Consider the case where F is a constant function $\lambda\_.\, M$ for some $M \in A$. The coherence condition then requires that $\mathrm{coe}^{r\to s}_{x.A}(M[r/x])$ is exactly $M[s/x]$ for all r, s, which implies in particular that $y : \mathbb{I} \gg M[y/x] = \mathrm{coe}^{0\to y}_{x.A}(M[0/x]) \in A[y/x]$. Were this true for all M, any path $Q \in \mathsf{Path}(x.A, M_0, M_1)$ would be equal to the path $\lambda y.\, \mathrm{coe}^{0\to y}_{x.A}(M_0)$.

examples do not present any new difficulties as far as implementing the Kan operations is concerned. Rather, they demonstrate constructor shapes we want to be able to express in our schema.

Propositional truncation Recall the following specification of the propositional truncation from Chapter 4.

$$
\begin{array}{l}
A : \mathsf{U} \gg \textbf{inductive}\ \|A\|\ \textbf{where} \\
\mid \mathsf{pt}(a : A) \in \|A\| \\
\mid \mathsf{squash}(t : \|A\|, t' : \|A\|, x : \mathbb{I}) \in \|A\| \quad [x \equiv 0 \hookrightarrow t \mid x \equiv 1 \hookrightarrow t']
\end{array}
$$

Like the suc constructor for the natural numbers, the squash constructor is recursive: it takes arguments from the type being constructed. In the case of a path constructor, we therefore also want to allow recursive arguments to occur in the boundary of a path constructor, as they do in squash.

For the eliminator, we aim to satisfy the following rule.

$$
\frac{\begin{array}{c}
q : \|A\| \gg D\ \mathsf{type} \qquad M \in \|A\| \qquad a : A \gg T_{\mathsf{pt}} \in D[\mathsf{pt}(a)/q] \\
t : \|A\|, t' : \|A\|, x : \mathbb{I}, r : D[t/q], r' : D[t'/q] \gg T_{\mathsf{squash}} \in D[\mathsf{squash}(t, t', x)/q] \\
t : \|A\|, t' : \|A\|, r : D[t/q], r' : D[t'/q] \gg T_{\mathsf{squash}}[0/x] = r \in D[t/q] \\
t : \|A\|, t' : \|A\|, r : D[t/q], r' : D[t'/q] \gg T_{\mathsf{squash}}[1/x] = r' \in D[t/q]
\end{array}}{\mathsf{elim}(q.D; M; a.T_{\mathsf{pt}}, a.a'.x.r.r'.T_{\mathsf{squash}}) \in D[M/q]}
$$

To the squash clause, we supply not only the arguments $t : \|A\|, t' : \|A\|, x : \mathbb{I}$ to the constructor, but also the results $r : D[t/q], r' : D[t'/q]$ of applying the eliminator to those terms, just as in the suc case of the natural number eliminator. In the operational semantics, these hypotheses are instantiated by recursive calls as in the following rule.

$$
\frac{\begin{array}{c}
N \coloneqq \mathsf{elim}(q.D; M; a.T_{\mathsf{pt}}, a.a'.x.r.r'.T_{\mathsf{squash}}) \\
N' \coloneqq \mathsf{elim}(q.D; M'; a.T_{\mathsf{pt}}, a.a'.x.r.r'.T_{\mathsf{squash}})
\end{array}}{\begin{array}{c}
\mathsf{elim}(q.D; \mathsf{squash}(M, M', y); a.T_{\mathsf{pt}}, a.a'.x.r.r'.T_{\mathsf{squash}}) \\
\longmapsto \\
T_{\mathsf{squash}}[M/t, M'/t', y/x, N/r, N'/r']
\end{array}}
$$

The equations in the elimination rule require the endpoints of the squash clause to agree with the two recursive calls, ensuring that the reduction rule above is sufficiently coherent.

Set truncation The propositional truncation sits at the bottom of a tower of truncation operators that cut off the higher structure of a type at some dimensionality. Where the propositional truncation identifies all *points* of a type, the *set truncation* collapses all *paths* between each pair of elements in a type [Uni13, §6.9]. We can express the set truncation by the following specification.

$$
\begin{array}{l}
A : \mathsf{U} \gg \textbf{inductive}\ \|A\|_0\ \textbf{where} \\
\mid \mathsf{pt}_0(a : A) \in \|A\|_0 \\
\mid \mathsf{squash}_0(t, t' : \|A\|_0, p, p' : \mathsf{Path}(\|A\|_0, t, t'), x : \mathbb{I}, y : \mathbb{I}) \in \|A\|_0 \\
\quad [x \equiv 0 \hookrightarrow p\ y \mid x \equiv 1 \hookrightarrow p'\ y \mid y \equiv 0 \hookrightarrow t \mid y \equiv 1 \hookrightarrow t']
\end{array}
$$

The constructor squash_0 is our first example of a 2-dimensional constructor. Given two paths, it creates a higher path (*i.e.*, square) identifying them: abstracting, we have $\lambda^{\mathbb{I}} y.\, \mathsf{squash}_0(t, t', p, p', 0, y) = p \in \mathsf{Path}(\|A\|_0, t, t')$ and $\lambda^{\mathbb{I}} y.\, \mathsf{squash}_0(t, t', p, p', 1, y) = p' \in \mathsf{Path}(\|A\|_0, t, t')$. The cubical notation for specifying boundaries generalizes gracefully to the greater-than-one-dimensional case.

Note that the arguments of squash_0 now draw not only from the *elements* of the type being defined, but from its *path types*. In particular, supporting this specification requires that we allow dependencies between recursive arguments—here, the dependency in the types of p and p' on t and t'. This is atypical in a schema for indexed inductive types, where recursive arguments are usually completely independent of each other. (Dependency among recursive arguments does, however, arise in schemata for inductive-inductive and inductive-recursive types.) We must also be able to apply the path arguments p and p' to interval terms (here y) in order to specify the boundary of the squash_0 constructor.

General truncation The propositional truncation and set truncation are also known as the (-1)-truncation and 0-truncation respectively; more generally, the n-truncation trivializes the structure of a type above dimension n. We could continue defining individual n-truncations using n-dimensional constructors, but many applications require that we have a single, parameterized definition of n-truncation uniformly in $n : \mathsf{Nat}$.

The HoTT Book proposes a fairly direct general definition of n-truncation using what is called a *hub-and-spoke construction* [Uni13, §6.7]. This definition relies on our ability to construct *n-sphere* types uniformly in $n : \mathsf{Nat}$, generalizing the circle (*i.e.*, 1-sphere) we saw in Chapter 4, by iteratively applying a *suspension* construction.

$$
\begin{array}{l}
A : \mathsf{U} \gg \textbf{inductive}\ \mathsf{Susp}(A)\ \textbf{where} \\
\mid \mathsf{north} \in \mathsf{Susp}(A) \\
\mid \mathsf{south} \in \mathsf{Susp}(A) \\
\mid \mathsf{merid}(a : A, x : \mathbb{I}) \in \mathsf{Susp}(A) \quad [x \equiv 0 \hookrightarrow \mathsf{north} \mid x \equiv 1 \hookrightarrow \mathsf{south}]
\end{array}
$$

If we take the type Circle defined in Chapter 4 and apply the suspension, we get a type Susp(Circle) with two "poles" (north and south) and a "line of longitude" (merid) from pole to pole for every "point on the equator" (element of the circle). This type is the 2-sphere. Iterating the suspension construction produces the n-spheres for every n. Actually, we can start this definition from -1, defining $\mathsf{Sphere}(-1) :\equiv \mathsf{Void}$ and $\mathsf{Sphere}(n+1) :\equiv \mathsf{Susp}(\mathsf{Sphere}(n))$ for $n \geq -1$; the type Susp(Void) is isomorphic to Bool, and Susp(Bool) to Circle.

Given a type A, a map $F \in \mathsf{Sphere}(0) \to A$ picks out two points in A, namely F north and F south. A map $F \in \mathsf{Sphere}(1) \to A$ picks out two points and two paths in A between them: F north, F south, $\lambda^{\mathbb{I}} x.\, F\,(\mathsf{merid}(\mathsf{north}, x))$, and $\lambda^{\mathbb{I}} x.\, F\,(\mathsf{merid}(\mathsf{south}, x))$. These collections of data are exactly the inputs to the squash and squash_0 constructors respectively, which inspires the following general definition of n-truncation.

$$
\begin{array}{l}
n : \mathsf{Nat}, A : \mathsf{U} \gg \textbf{inductive}\ \|A\|_n\ \textbf{where} \\
\mid \mathsf{pt}_n(a : A) \in \|A\|_n \\
\mid \mathsf{hub}_n(f : \mathsf{Sphere}(n) \to \|A\|_n) \in \|A\|_n \\
\mid \mathsf{spoke}_n(f : \mathsf{Sphere}(n) \to \|A\|_n, s : \mathsf{Sphere}(n), x : \mathbb{I}) \in \|A\|_n \\
\quad [x \equiv 0 \hookrightarrow \mathsf{hub}_n(f) \mid x \equiv 1 \hookrightarrow f\ s]
\end{array}
$$

For each diagram to be squashed, *i.e.*, map $\mathsf{Sphere}(n) \to \|A\|_n$, the n-truncation type adds a point $\mathsf{hub}_n(f)$ and draws a path $\lambda^{\mathbb{I}} x.\, \mathsf{spoke}_n(f, s, x)$ from the hub to each element $f\ s$ of the diagram, thus filling it in.

From a schema design perspective, the notable feature of this specification is its use of recursive arguments of function type—in this case, maps from $\mathsf{Sphere}(n)$ into the type being constructed—and likewise the application of these in the definition of the boundary, paralleling the use of paths in our previous definition of $\|A\|_0$. Such recursive arguments are called *generalized recursive arguments* in Dybjer's schema for (non-higher) indexed inductive types [Dyb94].

Note that not all function types involving the type being defined should be permissible in an inductive definition. For example, the existence of a type inductively defined by the following specification is contradictory.

$$
\begin{array}{l}
\textbf{inductive}\ \blacktriangleright\!\!\!\!\times\ \textbf{where} \\
\mid \mathsf{fold}(f : \blacktriangleright\!\!\!\!\times \to \mathsf{Bool}) \in \blacktriangleright\!\!\!\!\times
\end{array}
$$

The non-existence of this type can be blamed on the fact that $\blacktriangleright\!\!\!\!\times$ occurs *negatively* in the arguments to fold, that is, in the domain of a function type. Thus, for one, the existence of a fixed point is not guaranteed by theorems such as Theorem 2.1.20 that rely on monotonicity. Following the standard approaches for inductive type schemata, therefore, we will restrict recursive argument types to a *strictly positive* grammar, only allowing the type being defined to occur in the codomain of function types. Note that the

path type is monotone in its sole type argument, and so does not pose a problem in this regard.

5.3 Identity types

Our final illustrative example of the schema we wish to implement is not a higher inductive type at all, merely an ordinary indexed inductive type: Martin Löf's identity type with its J elimination rule (Section 2.1.5.4). While non-indexed inductive types such as Nat can be adapted to the cubical setting without any change, indexed inductive types are a different story; we will see that the implementation of their Kan operations requires tactics similar to those we have used for higher inductive types.

The identity type at A is a type indexed by two elements of A, which is inhabited by refl when those two elements are the same.

$$A : \mathsf{U} \gg \textbf{inductive}\ \mathsf{Id}(A, a : A, a' : A)\ \textbf{where}$$
$$\mid\ \mathsf{refl}(a : A) \in \mathsf{Id}(A, a, a)$$

Deriving the elimination rule for this type following Dybjer [Dyb94], we arrive at the so-called J rule, the elimination rule previously described in Section 2.1.5.4.

$$\frac{\begin{array}{c} a_0 : A, a_1 : A, p : \mathsf{Id}(A, a_0, a_1) \gg B\ \mathsf{type} \\ M_0 \in A \qquad M_1 \in A \qquad P \in \mathsf{Id}(A, M_0, M_1) \qquad a : A \gg N \in B[a/a_0, a/a_1, \mathsf{refl}(a)/p] \end{array}}{\mathsf{elim}(a_0.a_1.p.B; M_0, M_1; P; a.N) \in B[M_0/a_0, M_1/a_1, P/p]}$$

$$\frac{\begin{array}{c} a_0 : A, a_1 : A, p : \mathsf{Id}(A, a_0, a_1) \gg B\ \mathsf{type} \\ M \in A \qquad a : A \gg N \in B[a/a_0, a/a_1, \mathsf{refl}(a)/p] \end{array}}{\mathsf{elim}(a_0.a_1.p.B; M, M; \mathsf{refl}(M); a.N) = N[M/a] \in B[M/a_0, M/a_1, \mathsf{refl}(M)/p]}$$

In words, in order to construct a map into a type B predicated on two elements of A and an identity between them, it suffices to provide a clause for the refl case. Note that this elimination principle does not, for example, directly provide any way of constructing functions into type families such as $a : A, p : \mathsf{Id}(A, a, a) \gg B'$ type.

Naively, we might try to interpret $\mathsf{Id}(A, M_0, M_1)$ as the relation consisting only of a refl value whenever M_0 and M_1 are exactly equal in A. However, this definition fails to support coercion. We can see this by observing that coercion requires $\mathsf{Id}(A, M_0, M_1)$ to be inhabited not only when M_0 and M_1 are exactly equal, but whenever there is a path $P \in \mathsf{Path}(A, M_0, M_1)$ between them.

$$\mathsf{coe}^{0\to 1}_{x.\mathsf{Id}(A,M_0,P\,x)}(\mathsf{refl}(M_0)) \in \mathsf{Id}(A, M_0, M_1)$$

Conversely, one can easily use the eliminator for the identity type to transform identities $Q \in \mathsf{Id}(A, M_0, M_1)$ into paths, starting the existence of reflexive paths.

$$\mathsf{elim}(a_0.a_1.p.\mathsf{Path}(A, a_0, a_1); M_0, M_1; Q; a.\lambda^{\mathbb{I}} x.\, a) \in \mathsf{Path}(A, M_0, M_1)$$

In fact, one may straightforwardly show that these functions would constitute an isomorphism between $\mathsf{Path}(A, M_0, M_1)$ and $\mathsf{Id}(A, M_0, M_1)$. This would seem to suggest that we may *define* identity types to be path types—after all, we intended path types to play the role of identity types from the start. To do so, we would have to give some definition of the identity type eliminator as an operator on path types, a term satisfying the following typing rule.

$$\frac{\begin{array}{c} a_0 : A, a_1 : A, p : \mathsf{Path}(A, a_0, a_1) \gg B\ \mathsf{type} \\ M_0 \in A \qquad M_1 \in A \qquad P \in \mathsf{Path}(A, M_0, M_1) \qquad a : A \gg N \in B[a/a_0, a/a_1, \lambda^{\mathbb{I}}\_.\, a/p] \end{array}}{\text{“}\mathsf{elim}\text{”}(a_0.a_1.p.B; M_0, M_1; P; a.N) \in B[M_0/a_0, M_1/a_1, P/p]}$$

This much is possible: we may define the eliminator directly as follows.

$$\begin{gathered} \text{“}\mathsf{elim}\text{”}(a_0.a_1.p.B; M_0, M_1; P; a.N) \coloneqq \mathsf{coe}^{0\to 1}_{x.B[M_0/a_0, H_x\ 1/a_1, H_x/p]}(N[M_0/a]) \\ \text{where } H_x \coloneqq \lambda^{\mathbb{I}} y.\, \mathsf{hcom}^{0\to y}_{A}(M_0; x \equiv 0 \hookrightarrow y.M_0, x \equiv 1 \hookrightarrow y.P\ y) \end{gathered}$$

The term H_x here is constructed so that $H_0 = \lambda^{\mathbb{I}}\_.\, M_0$ and $H_1 = P$, allowing us to transfer terms over the former to terms over the latter by coercion.

While this term has the correct *type*, however, it fails to satisfy the *equation* required of an identity eliminator: the reduction rule $\text{“}\mathsf{elim}\text{”}(a_0.a_1.p.B; M, M; \lambda^{\mathbb{I}}(M); a.N) = N[M/a]$. The equation can be shown to hold up to a path, but there is no reason it should hold up to exact equality in general. The representative counterexamples involve composition in the universe, so we will not present them here; a detailed walkthrough can be found in [Ang19, §3.4]. The situation is actually quite dire: Swan has shown that, under certain basic assumptions, the semantic path types in cubical set models cannot be used constructively as an interpretation of identity types [Swa18b].

Of course, this does not mean that there is no way to construct identity types, only that they will not coincide with path types. As we saw above, the problem with the naive construction is that it is not closed under coercion. This parallels the issue we encountered back in Section 5.1, where we found that the naive interpretation of quotients failed to be closed under composition. The solution will be the same: introduce formal coercions.

As mentioned in Section 5.1, formal coercions are not a satisfactory general solution to coercion in higher inductive types: they require the resulting type to be as large as the types of its parameters, which precludes, *e.g.*, universes closed under quotients. However, we can improve on this “worst case” by introducing only formal coercions *between indices*

of an indexed inductive type, implementing general coercion with a combination of formal coercion between indices and non-formal coercion between parameters. In the case of an inductive type with trivial indexing, this reduces to the non-formal coercion solution we have used so far. With this approach, the size of an inductive type is only dependent on the size of its indices, not its parameters: $\mathsf{Id}(A, M_0, M_1)$ will be as large as A, this being the type of M_0 and M_1, but not as large as the *type of A* (some U).

In the case of identity types, formal coercion takes the following form, allowing us to coerce between different instantiations of M_0 and M_1 but not of A.

$$\frac{A\ \mathsf{type} \qquad x : \mathbb{I} \gg M_0, M_1 \in A \qquad r, s \in \mathbb{I} \qquad P \in \mathsf{Id}(A, M_0[r/x], M_1[r/x])}{\mathsf{fcoe}^{r\to s}_{x.(M_0,M_1)}(P) \in \mathsf{Id}(A, M_0[s/x], M_1[s/x])}$$

As is our custom, we also impose the equation $\mathsf{fcoe}^{r\to r}_{x.(M_0,M_1)}(P) = P$. Operationally, formal coercions are values unless trivial.

$$\frac{r \neq s}{\mathsf{fcoe}^{r\to s}_{x.(M_0,M_1)}(P)\ \mathsf{val}} \qquad\qquad \frac{}{\mathsf{fcoe}^{r\to r}_{x.(M_0,M_1)}(P) \longmapsto P}$$

To implement general coercion, we combine formal index coercion with a *parameter coercion* operator implemented by case analysis, satisfying the following typing rule.

$$\frac{x : \mathbb{I} \gg A\ \mathsf{type} \qquad M_0, M_1 \in A[r/x] \qquad r, s \in \mathbb{I} \qquad P \in \mathsf{Id}(A[r/x], M_0, M_1)}{\mathsf{pcoe}^{r\to s}_{x.A\blacktriangleright\mathsf{Id}}(P) \in \mathsf{Id}(A[s/x], \mathsf{coe}^{r\to s}_{x.A}(M_0), \mathsf{coe}^{r\to s}_{x.A}(M_1))}$$

In this case, we have a line $x.A$ in the type parameter, but indices M_0, M_1 only at the departure point r. These input indices are coerced along the parameter path $x.A$ in order to produce the indices for the output.

The general coercion operation is then derived by combining the parameter and index coercions: we first coerce to the correct parameters, then adjust the indices using a formal coercion.

$$\frac{}{\mathsf{coe}^{r\to s}_{x.\mathsf{Id}(A,M_0,M_1)}(P) \longmapsto \mathsf{fcoe}^{r\to s}_{x.(\mathsf{coe}^{x\to s}_{x.A}(M_0),\mathsf{coe}^{x\to s}_{x.A}(M_1))}(\mathsf{pcoe}^{r\to s}_{x.A\blacktriangleright\mathsf{Id}}(P))}$$

To complete the picture, we just need to implement parameter coercion. For this, we follow the pattern of coercion in higher inductive types: evaluate the argument to a value and push the coercion inside. For identity types, there will be three types of values: refl terms, formal coercions, and formal composites. Note that despite the lack of explicit path constructors, formal composite values become necessary in order to ensure that the paths

created by formal coercion are composable.

$$\frac{}{\mathsf{pcoe}^{r\to s}_{x.A\blacktriangleright\mathsf{Id}}(\mathsf{refl}(M)) \longmapsto \mathsf{refl}(\mathsf{coe}^{r\to s}_{x.A}(M))}$$

$$\frac{}{\mathsf{pcoe}^{r\to s}_{x.A\blacktriangleright\mathsf{Id}}(\mathsf{fcoe}^{t\to u}_{y.(M_0,M_1)}(P)) \longmapsto \mathsf{fcoe}^{t\to u}_{y.(\mathsf{coe}^{r\to s}_{x.A}(M_0),\mathsf{coe}^{r\to s}_{x.A}(M_1))}(\mathsf{pcoe}^{r\to s}_{x.A\blacktriangleright\mathsf{Id}}(P))}$$

$$\frac{}{\begin{array}{c}\mathsf{pcoe}^{r\to s}_{x.A\blacktriangleright\mathsf{Id}}(\mathsf{fhcom}^{t\to u}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})) \\ \longmapsto \\ \mathsf{fhcom}^{t\to u}(\mathsf{pcoe}^{r\to s}_{x.A\blacktriangleright\mathsf{Id}}(M); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{pcoe}^{r\to s}_{x.A\blacktriangleright\mathsf{Id}}(N_i)})\end{array}}$$

In the general case, the constructor case becomes slightly more complex in the same way that coercion in pushouts is more involved than in quotients: if the index of a constructor involves outside parameters, then an additional formal coercion is necessary in order to commute coercion past uses of those parameters. A simple example where this is needed is the *fiber family* $\mathsf{Fib}(A, B, f, -)$ of a function $f \in A \to B$, a family indexed by B whose elements at index $b : B$ are the elements of A mapped to b by f.

$$\begin{array}{l} A : \mathsf{U}, B : \mathsf{U}, f : A \to B \gg \textbf{inductive}\ \mathsf{Fib}(A, B, f, b : B)\ \textbf{where} \\ \mid\ \mathsf{fib}(a : A) \in \mathsf{Fib}(A, B, f, f\ a) \end{array}$$

In this example, parameter coercion of the constructor along a line $x : \mathbb{I} \gg F \in A \to B$ can be defined as follows, with an fcoe applying the necessary adjustment.

$$\frac{}{\mathsf{pcoe}^{r\to s}_{x.(A,B,F)\blacktriangleright\mathsf{Fib}}(\mathsf{fib}(M)) \longmapsto \mathsf{fcoe}^{r\to s}_{x.\mathsf{coe}^{x\to s}_{x.B}(F\ (\mathsf{coe}^{r\to x}_{x.A}(M)))}(\mathsf{fib}(\mathsf{coe}^{r\to s}_{x.A}(M)))}$$

To wrap up our definition of identity types, we still need to implement the eliminator. In addition to the refl constructor, we must handle the two formal Kan operator values. The same pattern used for composition in higher inductive types applies to both coercion and composition here: convert formal Kan operations in the domain into "real" Kan operations in the codomain, as shown for formal coercion below.

$$\frac{H^x := \mathsf{fcoe}^{r\to x}_{x.(M_0',M_1')}(P)}{\begin{array}{c}\mathsf{elim}(a_0.a_1.p.B; M_0, M_1; \mathsf{fcoe}^{r\to s}_{x.(M_0',M_1')}(P); a.N) \\ \longmapsto \\ \mathsf{coe}^{r\to s}_{x.B[M_0'/a_0,M_1'/a_1,H^x/p]}(\mathsf{elim}(a_0.a_1.p.B; M_0'[r/x], M_1'[r/x]; P; a.N))\end{array}}$$

If applied to the refl constructor, of course, the eliminator should simply apply the provided clause $a.N$, straightforwardly validating the reduction rule that cannot be achieved

with path types.

$$\frac{}{\mathsf{elim}(a_0.a_1.p.B; M_0, M_1; \mathsf{refl}(M); a.N) \longmapsto N[M/a]}$$

Comparing MLTT and cubical identity types In Martin-Löf type theory, the J eliminator is unnecessarily weak from the perspective of the computational semantics. Indeed, as noted in Section 2.1.5.4, the semantics justify an equality reflection rule that recovers *exact* equalities from elements of the identity type. It is worth examining why our definition of identity types for cubical type theory fails to satisfy the same principles.

The equality reflection rule in MLTT, for one, relies on the fact that the only values of the identity type are $\mathsf{refl}(M)$ terms. Thus, the only way a type $\mathsf{Id}(A, M_0, M_1)$ can be inhabited in an empty context is if the elements M_0 and M_1 are exactly equal. In cubical type theory, on the other hand, an element of an identity type may be an fcoe or fhcom term, in which case it does not follow that the two indices are exactly equal.

A fortiori, the MLTT semantics of identity types also validates the "K" rule, which constructs elements of type families over loops (identities from a given a to a) rather than arbitrary identities.

$$\frac{\begin{array}{c} a : A, p : \mathsf{Id}(A, a, a) \gg B\ \mathsf{type} \\ M \in A \qquad P \in \mathsf{Id}(A, M, M) \qquad a : A \gg N \in B[\mathsf{refl}(a)/p] \end{array}}{\mathsf{K}(a.p.B; M; P; a.N) \in B[M/a, P/p]}\ (\text{MLTT})$$

K is implemented by the reduction $\mathsf{K}(a.p.B; M; \mathsf{refl}(M'); a.N) \longmapsto N[M'/a]$. The K rule does not imply equality reflection on its own, but does imply that any loop $P \in \mathsf{Id}(A, M, M)$ is equal to $\mathsf{refl}(M)$ up to higher identity.

If we were to try to implement the K eliminator for our cubical identity types, on the other hand, we find ourselves stuck at the case $\mathsf{K}(a.p.B; M; \mathsf{fcoe}^{r \to s}_{x.M_0,M_1}(P); a.N)$. While the compound term $\mathsf{fcoe}^{r \to s}_{x.M_0,M_1}(P)$ is a loop by assumption, the inner identity P need not be. We cannot therefore progress by recursively applying K to P. In short, because of the presence of fcoe terms, applying the eliminator at one index may require the recursive application of the eliminator at a different index. This provides some justification for the form of the J eliminator for identity types and eliminators for indexed inductive types more generally: we require a type family $a_0 : A, a_1 : A, p : \mathsf{Path}(A, a_0, a_1) \gg B$ type defined on all possible indices, not only on those indices that can be inhabited using the generating constructors.

Chapter 6

General higher inductive types

We now assemble the strategies used for the examples in Chapter 5 into a general schema for higher inductive types and a computational interpretation for the instances thereof.

We begin in Section 6.1 by defining a specification schema: a formal language making precise the pseudo-code "**inductive**" declarations we have used thus far. The specification of a higher inductive type involves both types (the types of arguments to constructors) and terms (the boundaries of constructors), and so the specification language itself has the structure of a formal type theory, restricted so that its instances describe monotone operators on indexed relations.

In Section 6.2, we define an interpretation of this formal type theory into the underlying computational type theory. We use the interpretation to define the monotone operator on relations corresponding to an inductive specification. The relation for the inductive type in question is then defined as the least fixed-point of said operator. Using this definition on the level of relations, we construct a type system closed under indexed higher inductive pretypes. In the process, we show that the inductive relations are value-coherent, which amounts to proving the introduction rules for each constructor and for formal composition and coercion terms.

Next, in Section 6.3, we establish that the higher inductive pretypes support the Kan operations, making them full-fledged types. We easily dispense with composition, having already shown that the inductive type is closed under formal compositions. Defining and checking the well-typedness of coercion is much more involved. The definition requires a combination of the techniques used to handle path constructors (Section 5.1) and indices (Section 5.3) above. The proof of well-typedness, meanwhile, requires careful staging to manage the recursive structure of inductive types, stemming both from recursive constructors and the recursive formal Kan operators.

We complete the picture in Section 6.4 by establishing elimination principles for our higher inductive types. This again proceeds in several stages: defining the data required to eliminate from a given inductive type, defining the operational semantics of an elimina-

Judgment	Reading
$\Gamma \gg \Delta$ tel	Δ is a telescope of types
$\Gamma \gg \delta \in \Delta$	δ is an instantiation of Δ
$\Gamma \gg \Delta \blacktriangleright \mathcal{K}$ spec	$\mathcal{K}$ specifies a Δ-indexed HIT
$\Gamma \gg \Delta \blacktriangleright \mathcal{K} \twoheadrightarrow \mathcal{K}'$	$\mathcal{K}'$ is a prefix of $\mathcal{K}$
$\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \mathcal{C}$ constr	$\mathcal{C}$ is a constructor definition over $\mathcal{K}$
$\Gamma \gg \Delta \blacktriangleright \mathcal{K} \mathbin{@} \ell \Rightarrow (\mathcal{K}' \mid \mathcal{C})$	$\mathcal{C}$ appears in $\mathcal{K}$ with label ℓ, preceded by $\mathcal{K}'$
$\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \Theta$ actx	Θ is a context of recursive argument types over $\mathcal{K}$
$\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{a}$ atype	$\textsc{a}$ is a recursive argument type over $\mathcal{K}$ and Θ
$\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{m} \in \textsc{a}$	$\textsc{m}$ is an argument term of type $\textsc{a}$
$\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \theta \in \Theta$	θ is an argument substitution from Θ' to Θ

Figure 6.1: Judgments used in the definition of HIT specifications

tor given such data, and showing that said eliminator is indeed well-typed and validates the expected reduction rules. Like coercion, the final step requires an argument by the universal property of the inductive type relation as a least fixed-point.

Finally, in Section 6.5, we describe the *validity restriction*, an adjustment to the homogeneous composition operation introduced by Angiuli, Favonia, and Harper [AFH18] that enables a stronger characterization of the values of a higher inductive type.

6.1 Specifications

To give a computational interpretation of higher inductive types, we must first settle on a definition of higher inductive type. We define a class of HIT specifications relative to a value type system by way of a judgment $\Gamma \gg \Delta \blacktriangleright \mathcal{K}$ spec, read "$\mathcal{K}$ is a higher inductive type specification indexed by Δ in context (*i.e.*, with parameters) Γ". To define this judgment, we make use of a series of auxiliary judgments, catalogued in Figure 6.1, that define the well-formed constructors, recursive argument types, and boundary terms. (Each of the judgments in this figure is the unary form of a binary judgment.) These judgments are all defined relative to an ambient value type system, which we leave implicit in this section. The raw grammar of specifications, constructors, and so on is shown in Figure 6.2; the judgments operate on syntax drawn from said grammar.

The final four judgments of Figure 6.1—contexts, types, terms, and substitutions—constitute a small formal type theory within which the recursive parts of a specification are defined. For lack of a better name, we refer to these as *argument* contexts, types, terms, and substitutions respectively. Argument types are used to specify the types of recursive arguments to constructors, while argument terms appear in two places: as in-

$$\Delta, \Omega ::= \cdot \mid (\Delta, a : A) \qquad \delta, \omega ::= \cdot \mid (\delta, M/a) \qquad \mathcal{K} ::= \cdot \mid (\mathcal{K}, \ell : C)$$

$$C ::= \Phi.\Omega.[\delta; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}] \qquad \Theta ::= \cdot \mid (\Theta, a : \mathrm{A}) \qquad \theta ::= \cdot \mid (\theta, \mathrm{M}/a)$$

$$\mathrm{A}, \mathrm{B} ::= \textsc{ind}(\delta) \mid (a : A) \to \mathrm{B} \mid \textsc{Path}(x.\mathrm{A}, \mathrm{M}_0, \mathrm{M}_1)$$

$$\begin{aligned} \mathrm{M} \quad ::= \quad & \textsc{intro}_\ell(\phi; \omega; \theta) \mid \textsc{fcoe}^{r \to s}_{x.\delta}(\mathrm{M}) \mid \textsc{fhcom}^{r \to s}_{\delta}(\mathrm{M}; \overrightarrow{\xi_i \hookrightarrow x.\mathrm{N}_i}) \\ & \mid \lambda a.\, \mathrm{M} \mid \mathrm{M}\, N \mid \lambda^{\mathbb{I}} x.\, \mathrm{M} \mid \mathrm{M}\, r \end{aligned}$$

Figure 6.2: Grammar of HIT specifications

dices to dependent argument types (specifically, path types), and as the boundary terms for path constructors.

Before getting into the meat of the specification language, we first define the telescope judgment. A *(type) telescope* is a list of types $\Delta = (a_1 : A_1, \ldots, a_n : A_n)$ in a context Γ, each dependent on its predecessors. In other words, a telescope is a context (here, consisting only of types) over a context. We use these to specify the indices to a HIT as well as the non-recursive arguments to a constructor.

Definition 6.1.1 (Telescopes). The type telescopes, $\Gamma \gg \Delta = \Delta'$ tel, are inductively generated by the following rules.

$$\frac{}{\Gamma \gg \cdot = \cdot \text{ tel}} \qquad \frac{\Gamma \gg \Delta = \Delta' \text{ tel} \qquad \Gamma, \Delta \gg A = A' \text{ type}}{\Gamma \gg (\Delta, a : A) = (\Delta', a : A') \text{ tel}}$$

Note that we have (Γ, Δ) ctx whenever Γ ctx and $\Gamma \gg \Delta$ tel. Also, a telescope over the empty context is simply a context.

A telescope, like a context, can be instantiated by a list of terms.

Definition 6.1.2 (Instantiations). The telescope instantiations, $\Gamma \gg \delta = \delta' \in \Delta$, are inductively generated by the following rules.

$$\frac{}{\Gamma \gg \cdot = \cdot} \qquad \frac{\Gamma \gg \delta = \delta' \in \Delta \qquad \Gamma \gg M = M' \in A\delta}{\Gamma \gg (\delta, M/a) = (\delta', M'/a) \in (\Delta, a : A)}$$

We have a substitution $\Gamma' \gg (\gamma, \delta) \in \Gamma$ whenever $\Gamma' \gg \gamma \in \Gamma$ and $\Gamma' \gg \delta \in \Delta\gamma$. Given $\Gamma \gg \Delta$ tel, we always have a canonical "variable" instantiation, for which we write $\Gamma, \Delta \gg \overline{v}_\Delta \in \Delta$; given an instantiation $\Gamma \gg \delta \in \Delta$, we will also write $\overline{v}_\delta$ for $\overline{v}_\Delta$.

We start at the top-level with the definition of the specification judgment, then progressively go deeper into the auxiliary judgments; formally, of course, the dependency goes in the opposite direction. A specification is simply a list of named constructors, each of which may depend on previous constructors. (We adopt a global convention that the constructor names in a specification, like variables in a context, are mutually distinct.)

Definition 6.1.3 (Specifications). The specifications $\Gamma \gg \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec are generated by the following rules.

$$\frac{}{\Gamma \gg \Delta \blacktriangleright \cdot = \cdot \text{ spec}} \qquad \frac{\Gamma \gg \Delta \blacktriangleright \mathcal{K} = \mathcal{K}' \text{ spec} \qquad \Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright C = C' \text{ constr}}{\Gamma \gg \Delta \blacktriangleright (\mathcal{K}, \ell : C) = (\mathcal{K}', \ell : C') \text{ spec}}$$

A constructor over some $\Delta \blacktriangleright \mathcal{K}$ spec, as defined below, consists of several components: interval arguments (specified by an interval context Φ), non-recursive arguments (a telescope Ω), the index of the type in which it constructs an element (an instantiation δ of Δ), and a boundary (constraints ξ_i associated with boundary terms M_i).

Definition 6.1.4 (Constructors). The constructors $\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright C = C'$ constr are generated by the following rule.

$$\frac{\begin{array}{c}\Phi \text{ ictx} \\ \Gamma, \Phi \gg \Omega = \Omega' \text{ tel} \qquad \Gamma, \Phi, \Omega \gg \delta = \delta' \in \Delta \qquad \Gamma, \Phi, \Omega \gg \Delta \mid \mathcal{K} \blacktriangleright \Theta = \Theta' \text{ actx} \\ (\forall i)\ \Phi \Vdash \xi_i \in \mathbb{F} \qquad (\forall i, j)\ \Gamma, \Phi, \Omega, \xi_i, \xi_j \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M}_i = \mathrm{M}'_j \in \textsc{ind}(\delta)\end{array}}{\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \Phi.\Omega.[\delta; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}] = \Phi.\Omega'.[\delta'; \Theta'.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}'_i}] \text{ constr}}$$

While we have generally written the interval arguments as the final arguments of a constructor in our examples, we place them at the start for the general case; this is slightly more general, as it allows the types of subsequent arguments to depend on the interval variables. For the most part, each piece of a constructor may depend on what comes before: the non-recursive arguments can mention the interval arguments, the index and recursive arguments may depend on the non-recursive arguments, and the boundary can depend on all arguments.

Remark 6.1.5. Most of the data of a constructor can freely depend on the ambient context Γ. However, we require that the constraints ξ_i specifying the boundary mention only interval variables that are arguments to the constructor (Φ above), not external variables. We can see why this restriction is necessary by examining the following malformed specification, in which the boundary of b depends on an external parameter $x : \mathbb{I}$.

$$\begin{array}{l} x : \mathbb{I} \gg \textbf{inductive } \blacktriangleright(x) \textbf{ where} \\ \mid \mathrm{a}_x \in \blacktriangleright(x) \\ \mid \mathrm{b}_x \in \blacktriangleright(x) \quad [x \equiv 0 \hookrightarrow \mathrm{a}] \end{array}$$

Suppose that such a type *does* exist, and consider the coercion $\mathrm{coe}^{0\to 1}_{x.\blacktriangleright(x)}(\mathrm{b}_0)$. On the one hand, we have a path $\lambda y.\, \mathrm{coe}^{y\to 1}_{x.\blacktriangleright(x)}(\mathrm{b}_y) \in \mathrm{coe}^{0\to 1}_{x.\blacktriangleright(x)}(\mathrm{b}_0) \rightsquigarrow \mathrm{b}_1$. On the other hand, b_0 is equal to a_0 by definition, so we also have a path $\lambda y.\, \mathrm{coe}^{y\to 1}_{x.\blacktriangleright(x)}(\mathrm{a}_y) \in \mathrm{coe}^{0\to 1}_{x.\blacktriangleright(x)}(\mathrm{b}_0) \rightsquigarrow \mathrm{a}_1$. Combining these with composition, we should have a path $a_1 \rightsquigarrow b_1$. However, there is no such path among the values of $\blacktriangleright(1)$, which consist only of a and b constructors and formal homogeneous composites.

In order to speak of the constructors within a given specification, we define constructor lookup and sub-specification judgments.

Definition 6.1.6 (Constructor lookup). We define $\Gamma \gg \Delta \blacktriangleright \mathcal{K} \mathbin{@} \ell \Rightarrow (\mathcal{K}' \mid C)$ (specification $\mathcal{K}$ contains C at label ℓ, preceded by the sub-specification $\mathcal{K}'$) as generated by the following rules.

$$\frac{\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright C = C' \ \mathsf{constr}}{\Gamma \gg \Delta \blacktriangleright (\mathcal{K}, \ell : C) \mathbin{@} \ell \Rightarrow (\mathcal{K} \mid C')}$$

$$\frac{\Gamma \gg \Delta \blacktriangleright \ell \mathbin{@} \mathcal{K} \Rightarrow (\mathcal{K}' \mid C) \qquad \Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright C \ \mathsf{constr}}{\Gamma \gg \Delta \blacktriangleright (\mathcal{K}, \ell' : C') \mathbin{@} \ell \Rightarrow (\mathcal{K}' \mid C)}$$

Definition 6.1.7 (Sub-specifications). We define $\Gamma \gg \Delta \blacktriangleright \mathcal{K} \twoheadrightarrow \mathcal{K}'$ (specification $\mathcal{K}$ extends $\mathcal{K}'$) as generated by the following rules.

$$\frac{}{\Gamma \gg \Delta \blacktriangleright \mathcal{K} \twoheadrightarrow \cdot} \qquad \frac{\Gamma \gg \Delta \blacktriangleright \mathcal{K} \mathbin{@} \ell \Rightarrow (\mathcal{K}' \mid C)}{\Gamma \gg \Delta \blacktriangleright \mathcal{K} \twoheadrightarrow (\mathcal{K}', \ell : C)}$$

Finally, we come to the definition of the formal argument type theory, which consists of mutually inductively defined context, substitution, type, and term judgments. We define the argument type theory as a formalism rather than a computational theory because we want to be able to analyze the *syntax* of argument types and terms, in particular to generate the eliminator premise induced by a constructor.

Definition 6.1.8. We define $\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \Theta = \Theta' \ \mathsf{actx}$, $\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \theta = \theta' \in \Theta$, $\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{A} = \mathrm{A}' \ \mathsf{atype}$, and $\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{M}' \in \mathrm{A}$ as mutually inductively generated by the rules in Figures 6.3 and 6.4.

The type theory so defined is fairly minimal: it has function types and path types with their associated introduction and elimination forms, and it has a stand-in $\textsc{ind}(-)$ for the inductive family being defined with its own associated introduction forms (constructors and formal Kan operators). Each constructor is equipped with a reduction rule for each

Contexts

$$\frac{}{\Delta \mid \mathcal{K} \blacktriangleright \cdot \ \mathsf{actx}} \qquad \frac{\Delta \mid \mathcal{K} \blacktriangleright \Theta \ \mathsf{actx} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{A} \ \mathsf{atype}}{\Delta \mid \mathcal{K} \blacktriangleright (\Theta, a : \mathrm{A}) \ \mathsf{actx}}$$

Substitutions

$$\frac{}{\Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \cdot \in \cdot} \qquad \frac{\Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \theta = \theta' \in \Theta \qquad \Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \mathrm{M} = \mathrm{M}' \in \mathrm{A}}{\Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright (\theta, \mathrm{M}/a) = (\theta', \mathrm{M}'/a) \in (\Theta, a : \mathrm{A})}$$

Types

$$\frac{\delta = \delta' \in \Delta}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{IND}(\delta) = \mathrm{IND}(\delta') \ \mathsf{atype}}$$

$$\frac{A = A' \ \mathsf{type} \qquad a : A \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{B} = \mathrm{B}' \ \mathsf{atype}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright (a : A) \to \mathrm{B} = (a : A') \to \mathrm{B}' \ \mathsf{atype}}$$

$$\frac{\begin{array}{c} x : \mathbb{I} \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{A} = \mathrm{A}' \ \mathsf{atype} \\ \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M}_0 = \mathrm{M}_0' \in \mathrm{A}[0/x] \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M}_1 = \mathrm{M}_1' \in \mathrm{A}[1/x] \end{array}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{PATH}(x.\mathrm{A}, \mathrm{M}_0, \mathrm{M}_1) = \mathrm{PATH}(x.\mathrm{A}', \mathrm{M}_0', \mathrm{M}_1') \ \mathsf{atype}}$$

Figure 6.3: Inductive definition of the argument contexts, substitutions, and types. The ambient context Γ is omitted for readability.

boundary constraint. In order to ensure that the recursive arguments to a constructor are strictly positive in the type being defined, the domain of an argument function type $(a : A) \to \mathrm{B}$ is not itself an argument type but an ordinary "external" type.

This language could be straightforwardly extended to include product types $(a : \mathrm{A}) \times \mathrm{B}$, for example, without significantly disrupting the development that follows. Our choice of a fairly minimal theory is motivated by a desire to avoid huge definitions and proofs by case analysis in what follows, not by any fundamental concerns about particular extensions.

We will take for granted standard admissibility theorems such as weakening and stability under substitution for the formal type theory; the theory does not contain any features that would interfere with standard proofs of such results.

Constructors

$$\frac{\begin{array}{c}\Gamma \gg \Delta \blacktriangleright \mathcal{K} \mathbin{@} \ell \Rightarrow (\_ \mid \Phi.\Omega.[\delta; \Theta'.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \\ \phi = \phi' \in \Phi \qquad \omega = \omega' \in \Omega\phi \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \theta = \theta' \in \Theta'\phi\omega\end{array}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{intro}_\ell(\phi; \omega; \theta) = \textsc{intro}_\ell(\phi'; \omega'; \theta') \in \textsc{ind}(\delta(\phi, \omega))}$$

$$\frac{\begin{array}{c}\Gamma \gg \Delta \blacktriangleright \mathcal{K} \mathbin{@} \ell \Rightarrow (\_ \mid \Phi.\Omega.[\delta; \Theta'.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \\ \phi = \phi' \in \Phi \qquad \omega = \omega' \in \Omega\phi \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \theta = \theta' \in \Theta'\phi\omega \qquad \xi_j\phi \text{ satisfied}\end{array}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{intro}_\ell(\phi; \omega; \theta) = \mathrm{M}_j(\phi, \omega, \theta) \in \textsc{ind}(\delta(\phi, \omega))}$$

Coercion

$$\frac{x : \mathbb{I} \gg \delta = \delta' \in \Delta \qquad r = r' \in \mathbb{I} \qquad s = s' \in \mathbb{I} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{M}' \in \textsc{ind}(\delta[r/x])}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{fcoe}^{r \to s}_{x.\delta}(\mathrm{M}) = \textsc{fcoe}^{r' \to s'}_{x.\delta'}(\mathrm{M}') \in \textsc{ind}(\delta[s/x])}$$

$$\frac{x : \mathbb{I} \gg \delta \in \Delta \qquad r \in \mathbb{I} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} \in \textsc{ind}(\delta[r/x])}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{fcoe}^{r \to r}_{x.\delta}(\mathrm{M}) = \mathrm{M} \in \textsc{ind}(\delta[r/x])}$$

Composition

$$\frac{\begin{array}{c}\delta = \delta' \in \Delta \qquad r = r' \in \mathbb{I} \qquad s = s' \in \mathbb{I} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{M}' \in \textsc{ind}(\delta) \\ (\forall i)\ \xi_i = \xi'_i \in \mathbb{F} \qquad (\forall i, j)\ \xi_i, \xi_j, x : \mathbb{I} \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{N}_i = \mathrm{N}'_j \in \textsc{ind}(\delta) \\ (\forall i)\ \xi_i \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{N}_i[r/x] \in \textsc{ind}(\delta)\end{array}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{fhcom}^{r \to s}_{\delta}(\mathrm{M}; \overrightarrow{\xi_i \hookrightarrow x.\mathrm{N}_i}) = \textsc{fhcom}^{r' \to s'}_{\delta'}(\mathrm{M}'; \overrightarrow{\xi'_i \hookrightarrow x.\mathrm{N}'_i}) \in \textsc{ind}(\delta)}$$

$$\frac{\begin{array}{c}\delta \in \Delta \qquad r \in \mathbb{I} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} \in \textsc{ind}(\delta) \\ (\forall i)\ \xi_i \in \mathbb{F} \qquad (\forall i, j)\ \xi_i, \xi_j, x : \mathbb{I} \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{N}_i = \mathrm{N}_j \in \textsc{ind}(\delta) \\ (\forall i)\ \xi_i \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{N}_i[r/x] \in \textsc{ind}(\delta)\end{array}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{fhcom}^{r \to r}_{\delta}(\mathrm{M}; \overrightarrow{\xi_i \hookrightarrow x.\mathrm{N}_i}) = \mathrm{M} \in \textsc{ind}(\delta)}$$

$$\frac{\begin{array}{c}\delta \in \Delta \qquad r, s \in \mathbb{I} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} \in \textsc{ind}(\delta) \\ (\forall i)\ \xi_i \in \mathbb{F} \qquad (\forall i, j)\ \xi_i, \xi_j, x : \mathbb{I} \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{N}_i = \mathrm{N}_j \in \textsc{ind}(\delta) \\ (\forall i)\ \xi_i \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{N}_i[r/x] \in \textsc{ind}(\delta) \qquad \xi_k \text{ satisfied}\end{array}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \textsc{fhcom}^{r \to s}_{\delta}(\mathrm{M}; \overrightarrow{\xi_i \hookrightarrow x.\mathrm{N}_i}) = \mathrm{N}_k[s/x] \in \textsc{ind}(\delta)}$$

Figure 6.4: Inductive definition of argument terms (constructors and Kan operations). The ambient context Γ is omitted for readability.

Functions

$$\frac{a : A \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{N} = \mathrm{N}' \in \mathrm{B}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \lambda a.\, \mathrm{N} = \lambda a.\, \mathrm{N}' \in \mathrm{B}} \qquad \frac{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{F} = \mathrm{F}' \in (a : A) \to \mathrm{B} \qquad M = M' \in A}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{F}\, M = \mathrm{F}'\, M' \in \mathrm{B}[M/a]}$$

$$\frac{a : A \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{N} \in \mathrm{B} \qquad M \in A}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright (\lambda a.\, \mathrm{N})\, M = \mathrm{N}[M/a] \in \mathrm{B}[M/a]}$$

$$\frac{A \text{ type} \qquad a : A \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{B} \text{ atype} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{F} \in (a : A) \to \mathrm{B}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{F} = \lambda a.\, \mathrm{F}\, a \in (a : A) \to \mathrm{B}}$$

Paths

$$\frac{x : \mathbb{I} \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{M}' \in \mathrm{A}}{x : \mathbb{I} \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \lambda^{\mathbb{I}} x.\, \mathrm{M} = \lambda^{\mathbb{I}} x.\, \mathrm{M}' \in \textsc{Path}(x.\mathrm{A}, \mathrm{M}[0/x], \mathrm{M}'[1/x])}$$

$$\frac{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{P} = \mathrm{P}' \in \textsc{Path}(x.\mathrm{A}, \mathrm{M}_0, \mathrm{M}_1) \qquad r = r' \in \mathbb{I}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{P}\, r = \mathrm{P}'\, r' \in \mathrm{A}[r/x]}$$

$$\frac{x : \mathbb{I} \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{N} = \mathrm{N}' \in \mathrm{A} \qquad r \in \mathbb{I}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright (\lambda^{\mathbb{I}} x.\, N)\, r = N[r/x] \in A[r/x]}$$

$$\frac{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{P} = \mathrm{P}' \in \textsc{Path}(\mathrm{A}, \mathrm{M}_0, \mathrm{M}_1)}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{P} = \lambda^{\mathbb{I}} x.\, \mathrm{P}\, x \in \textsc{Path}(x.\mathrm{A}, \mathrm{M}_0, \mathrm{M}_1)} \qquad \frac{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{P} \in \textsc{Path}(x.\mathrm{A}, \mathrm{M}_0, \mathrm{M}_1)}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{P}\, \varepsilon = \mathrm{M}_\varepsilon \in \mathrm{A}[\varepsilon/x]}$$

Structural

$$\frac{(a : \mathrm{A}) \in \Theta}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright a \in \mathrm{A}} \qquad \frac{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} \in \mathrm{A} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{A} = \mathrm{B} \text{ atype}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} \in \mathrm{B}}$$

$$\frac{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{N} \in \mathrm{A}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{N} = \mathrm{M} \in \mathrm{A}} \qquad \frac{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{N} \in \mathrm{A} \qquad \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{N} = \mathrm{P} \in \mathrm{A}}{\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{P} \in \mathrm{A}}$$

Figure 6.4: Inductive definition of argument terms (functions, paths, and structural rules). The ambient context Γ is omitted for readability.

Proposition 6.1.9 (Standard admissibilities).

- *Term substitution.* If $\Gamma' \gg \gamma = \gamma' \in \Gamma$ and $\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \Theta = \Theta'$ actx, then we have $\Gamma' \gg \Delta\gamma \mid \mathcal{K}\gamma \blacktriangleright \Theta\gamma = \Theta'\gamma'$ actx; the argument substitutions, types, and terms are likewise stable under substitution.
- *Argument substitution.* If $\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \theta = \theta' \in \Theta$ and $\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{A} = \mathrm{A}'$ atype, then we have $\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \mathrm{A}\theta = \mathrm{A}'\theta'$ atype; the argument substitutions and terms are likewise stable under argument substitution.
- *Specification weakening.* If $\Gamma \gg \Delta \blacktriangleright \mathcal{K} \twoheadrightarrow \mathcal{K}'$ and $\Gamma \gg \Delta \mid \mathcal{K}' \blacktriangleright \Theta = \Theta'$ actx, then we have $\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \Theta = \Theta'$ actx; the argument substitutions, types, and terms are likewise stable under specification extension.

As with ordinary substitutions, the argument substitutions include identity and weakening substitutions, each of which is the identity on raw terms.

6.2 Interpreting specifications

To get from a specification language to a type system closed under HITs, the first step is to define the interpretation of the formal argument type theory. There are two sides of such an interpretation: the syntactic and the semantic (*i.e.*, relational).

On the one hand, an argument type or term may be interpreted as a piece of syntax in the untyped programming language. On the other hand, each argument type may also be interpreted as an operator on indexed value relations, taking an interpretation for the inductive family $\textsc{ind}(-)$ as input and producing an interpretation of the compound type. For example, given $\Delta \blacktriangleright \mathcal{K}$ spec and a Δ-indexed relation R, we would interpret $A \longrightarrow \textsc{ind}(\delta)$ at R as relating λ-values that take terms in A to terms in the instantiation of R at δ. Using the interpretation of argument types as operators on relations, we likewise build up an interpretation of constructors $\mathcal{C}$ and then specifications $\mathcal{K}$ as operators on relations. Finally, we define the inductive relation associated to $\mathcal{K}$ to be the least fixed point of its interpretation as a relational operator. With this definition in hand, it becomes straightforward to construct a type system which is closed under (and supports universes closed under) such relations.

6.2.1 Syntactic interpretation

We start with the syntactic interpretation of the argument type theory, defined here as a collection of operations taking raw terms of that language to terms in the untyped programming language. We show later on that these raw operations preserve well-typedness in the right circumstances.

Type former

$$\frac{}{\mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta)\ \mathsf{val}}$$

Constructors

$$\frac{(\ell : \Psi.\Omega.[\delta; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K} \qquad (\nexists i)\ \xi_i\ \mathsf{satisfied}}{\mathsf{intro}^{\mathcal{K}}_{\ell}(\phi; \omega; \chi)\ \mathsf{val}}$$

$$\frac{(\ell : \Psi.\Omega.[\delta; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K} \qquad (\nexists i < k)\ \xi_i\ \mathsf{satisfied} \qquad \xi_k\ \mathsf{satisfied}}{\mathsf{intro}^{\mathcal{K}}_{\ell}(\phi; \omega; \chi) \longmapsto (\!|\Theta.\mathrm{M}_k[\phi, \omega]|\!)_{\mathcal{K}}(\chi)}$$

Formal coercions

$$\frac{r \neq s}{\mathsf{fcoe}^{r \to s}_{x.\delta}(M)\ \mathsf{val}} \qquad\qquad \frac{}{\mathsf{fcoe}^{r \to r}_{x.\delta}(M) \longmapsto M}$$

Formal composites

$$\frac{r \neq s \qquad (\nexists i)\ \xi_i\ \mathsf{satisfied}}{\mathsf{fhcom}^{r \to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})\ \mathsf{val}} \qquad \frac{(\nexists i)\ \xi_i\ \mathsf{satisfied}}{\mathsf{fhcom}^{r \to r}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto M}$$

$$\frac{(\nexists i < k)\ \xi_i\ \mathsf{satisfied} \qquad \xi_k\ \mathsf{satisfied}}{\mathsf{fhcom}^{r \to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto N_k[s/x]}$$

Formal heterogeneous composites

$$\mathsf{fcom}^{r \to s}_{x.\delta}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \coloneqq \mathsf{fhcom}^{r \to s}(\mathsf{fcoe}^{r \to s}_{x.\delta}(M); \overrightarrow{\xi_i \hookrightarrow x.\mathsf{fcoe}^{x \to s}_{x.\delta}(x.N_i)})$$

Figure 6.5: Operational semantics for formation and introduction in HITs

At this point, of course, we need to begin making assumptions about the untyped programming language in question. Henceforth, we assume that the programming language supports the operational semantics shown in Figure 6.5, in addition to the usual operational semantics for terms associated with function and path types. The operational semantics contains no surprises: constructors are values except on their boundary, while formal coercions and composites are values except where reduction is required by the Kan operation equations.

Argument types, terms, and substitutions are all defined relative to an argument context Θ; to interpret them, we therefore require an interpretation of the variables in Θ as input. Writing $(\!|-|\!)^{\Delta}_{\mathcal{K}}$ for interpretation relative to a specification $\Delta \blacktriangleright \mathcal{K}$ spec (dropping the Δ annotation where unnecessary), the intent is that interpretation produces well-typed results in the following fashion.

$$\begin{array}{rcll} & \Delta \mid \mathcal{K} \blacktriangleright \Theta \text{ actx} & \Rightarrow & (\!|\Theta|\!)^{\Delta}_{\mathcal{K}} \text{ tel} \\ \chi \in (\!|\Theta|\!)^{\Delta}_{\mathcal{K}} \quad \wedge & \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{A} \text{ atype} & \Rightarrow & (\!|\Theta.\mathrm{A}|\!)^{\Delta}_{\mathcal{K}}(\chi) \text{ type} \\ \chi \in (\!|\Theta|\!)^{\Delta}_{\mathcal{K}} \quad \wedge & \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} \in \mathrm{A} & \Rightarrow & (\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi) \in (\!|\Theta.\mathrm{A}|\!)^{\Delta}_{\mathcal{K}}(\chi) \\ \chi \in (\!|\Theta'|\!)^{\Delta}_{\mathcal{K}} \quad \wedge & \Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \theta \in \Theta & \Rightarrow & (\!|\Theta'.\theta|\!)_{\mathcal{K}}(\chi) \in (\!|\Theta|\!)^{\Delta}_{\mathcal{K}} \end{array}$$

All of these interpretations are exceedingly straightforward; we transform each argument term or type into its "real" equivalent.

Definition 6.2.1 (Interpretation of terms and substitutions). Let $\mathcal{K}$ be a specification, Θ be an argument context, and M be an argument term. Let χ be an instantiation for the variables in Θ. We define the interpretation of M at Θ, written $(\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi)$, as follows.

$$\begin{aligned} (\!|\Theta.a|\!)_{\mathcal{K}}(\chi) &\coloneqq a\chi \\ (\!|\Theta.\textsc{intro}_{\ell}(\phi;\omega;\theta)|\!)_{\mathcal{K}}(\chi) &\coloneqq \mathsf{intro}^{\mathcal{K}}_{\ell}(\phi;\omega;(\!|\Theta.\theta|\!)_{\mathcal{K}}(\chi)) \\ (\!|\Theta.\textsc{fcoe}^{r\to s}_{x.\delta}(\mathrm{M})|\!)_{\mathcal{K}}(\chi) &\coloneqq \mathsf{fcoe}^{r\to s}_{x.\delta}((\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi)) \\ (\!|\Theta.\textsc{fhcom}^{r\to s}_{\delta}(\mathrm{M};\overrightarrow{\xi_i \hookrightarrow x.\mathrm{N}_i})|\!)_{\mathcal{K}}(\chi) &\coloneqq \mathsf{fhcom}^{r\to s}((\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi);\overrightarrow{\xi_i \hookrightarrow x.(\!|\Theta.\mathrm{N}_i|\!)_{\mathcal{K}}(\chi)}) \\ (\!|\Theta.\lambda a.\,\mathrm{N}|\!)_{\mathcal{K}}(\chi) &\coloneqq \lambda a.\,(\!|\Theta.\mathrm{N}|\!)_{\mathcal{K}}(\chi) \\ (\!|\Theta.\mathrm{F}\ M|\!)_{\mathcal{K}}(\chi) &\coloneqq ((\!|\Theta.\mathrm{F}|\!)_{\mathcal{K}}(\chi))\ M \\ (\!|\Theta.\lambda^{\mathbb{I}} x.\,\mathrm{M}|\!)_{\mathcal{K}}(\chi) &\coloneqq \lambda^{\mathbb{I}} x.\,(\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi) \\ (\!|\Theta.\mathrm{P}\ r|\!)_{\mathcal{K}}(\chi) &\coloneqq ((\!|\Theta.\mathrm{P}|\!)_{\mathcal{K}}(\chi))\ r \end{aligned}$$

We define $(\!|\Theta.\theta|\!)_{\mathcal{K}}(\chi)$ for argument substitutions θ elementwise.

$$\begin{aligned} (\!|\Theta.\cdot|\!)_{\mathcal{K}}(\chi) &\coloneqq \cdot \\ (\!|\Theta.(\theta,\mathrm{M}/a)|\!)_{\mathcal{K}}(\chi) &\coloneqq ((\!|\Theta.\theta|\!)_{\mathcal{K}}(\chi),(\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi)/a) \end{aligned}$$

Definition 6.2.2 (Syntactic interpretation of types and contexts). Let a telescope Δ, specification $\mathcal{K}$, argument context Θ, and argument type $\textsc{b}$ be given. Let χ be an instantiation for the variables in Θ. We define the syntactic interpretation of $\textsc{b}$ at χ, written $(\!|\Theta.\textsc{b}|\!)^{\Delta}_{\mathcal{K}}(\chi)$, as follows.

$$\begin{aligned}
(\!|\Theta.\textsc{ind}(\delta)|\!)^{\Delta}_{\mathcal{K}}(\chi) &:= \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta) \\
(\!|\Theta.(b : B) \to \textsc{c}|\!)^{\Delta}_{\mathcal{K}}(\chi) &:= (b : B) \to (\!|\Theta.\textsc{c}|\!)^{\Delta}_{\mathcal{K}}(\chi) \\
(\!|\Theta.\textsc{Path}(x.\textsc{b}, \textsc{m}_0, \textsc{m}_1)|\!)^{\Delta}_{\mathcal{K}}(\chi) &:= \mathsf{Path}(x.(\!|\Theta.\textsc{b}|\!)^{\Delta}_{\mathcal{K}}(\chi), M_0, M_1) \\
&\qquad \text{where } M_\varepsilon := (\!|\Theta.\textsc{m}_\varepsilon|\!)_{\mathcal{K}}(\chi) \text{ for } \varepsilon \in \{0,1\}
\end{aligned}$$

We define a telescope $(\!|\Theta|\!)^{\Delta}_{\mathcal{K}}$, the syntactic interpretation of Θ, as follows.

$$\begin{aligned}
(\!|\cdot|\!)^{\Delta}_{\mathcal{K}} &:= \cdot \\
(\!|\Theta, a : \textsc{a}|\!)^{\Delta}_{\mathcal{K}} &:= (\!|\Theta|\!)^{\Delta}_{\mathcal{K}}, a : (\!|\Theta.\textsc{a}|\!)^{\Delta}_{\mathcal{K}}(\overline{v}_{(\!|\Theta|\!)^{\Delta}_{\mathcal{K}}})
\end{aligned}$$

6.2.2 Relational interpretation

Next, we have a second interpretation of argument types and contexts as operators on indexed relations. Given a specification $\Psi \Vdash \Delta \blacktriangleright \mathcal{K}$ spec and (Ψ, Δ)-relation R, we can interpret any argument type $\Psi \Vdash \Delta \mid \mathcal{K} \mid \cdot \blacktriangleright \textsc{a}$ atype as a Ψ-relation by interpreting instances of $\textsc{ind}(-)$ with R and interpreting compound types by their usual relational definitions. (Recall that we defined Γ-relations for arbitrary contexts Γ in Definition 3.1.25).

First, we define some notation for the function and path type formers as relational operators, following the definitions in Sections 2.1.4 and 3.1.5 respectively.

Definition 6.2.3. Given a term A and $(\Psi, a{:}A)$-relation R, we define a Ψ-relation $\mathit{Fun}(A, R)$ for $\Psi' \Vdash \psi \in \Psi$ as follows.

$$V \approx V' \in \mathit{Fun}(A,R)\langle\psi\rangle :\Longleftrightarrow \begin{cases} V = \lambda a.\, N \text{ and } V = \lambda a.\, N' \text{ with} \\ \Psi', a : A\psi \gg N \approx N' \in \Downarrow R(\psi, a/a) \end{cases}$$

Definition 6.2.4. Given a $(\Psi, x{:}\mathbb{I})$-relation R and terms M_0 and M_1, we define a Ψ-relation $\mathit{Path}(R, M_0, M_1)$ for $\Psi' \Vdash \psi \in \Psi$ as follows.

$$V \approx V' \in \mathit{Path}(R,M_0,M_1)\langle\psi\rangle :\Longleftrightarrow \begin{cases} V = \lambda^{\mathbb{I}} x.\, M \text{ and } V = \lambda^{\mathbb{I}} x.\, M' \text{ with} \\ M \approx M' \in \Downarrow R(\psi, x/x) \text{ and} \\ M[\varepsilon/x] \approx M_\varepsilon \psi \in \Downarrow R(\psi, \varepsilon/x) \text{ for } \varepsilon \in \{0,1\} \end{cases}$$

These components then assemble straightforwardly into an interpretation of argument types and contexts.

Definition 6.2.5. Let Δ be a telescope, $\mathcal{K}$ be a specification, Θ be an argument context, and $\textsc{a}$ be an argument type. Let R be a (Ψ, Δ)-relation and let χ be an instantiation for the variables in Θ. We define a Ψ-relation $\{\!|\Theta.\textsc{a}|\!\}_{\mathcal{K}}(R, \chi)$, the relational interpretation of $\Theta.\textsc{a}$ at R and χ, as follows.

$$\begin{aligned}
\{\!|\Theta.\textsc{ind}(\delta)|\!\}_{\mathcal{K}}(R, \chi) &\coloneqq R[\mathrm{id}_\Psi, \delta] \\
\{\!|\Theta.(b : A) \to \textsc{b}|\!\}_{\mathcal{K}}(R, \chi) &\coloneqq \mathit{Fun}(A, S) \\
\{\!|\Theta.\textsc{Path}(x.\textsc{a}, \textsc{m}_0, \textsc{m}_1)|\!\}_{\mathcal{K}}(R, \chi) &\coloneqq \mathit{Path}(\{\!|\Theta.\textsc{a}|\!\}_{\mathcal{K}}(R, \chi), (\!|\Theta.\textsc{m}_0|\!)_{\mathcal{K}}(\chi), (\!|\Theta.\textsc{m}_1|\!)_{\mathcal{K}}(\chi))
\end{aligned}$$

The $(\Psi, a : A)$-relation S in the function case is defined as follows.

$$\text{where } S\langle\psi, M/a\rangle \coloneqq \bigcap \{\{\!|\Theta.\textsc{b}\psi[M'/a]|\!\}_{\mathcal{K}}(R\psi, \chi\psi)\langle \mathrm{id}_{\Psi'}\rangle \mid \Psi' \Vdash M = M' \in A\psi\}$$

We use the intersection above to ensure that this is a well-defined $(\Psi, a{:}A)$-relation, these being required to respect equality in the indexing context.

Definition 6.2.6. Let Δ be a telescope, $\mathcal{K}$ be a specification, and let Θ be an argument context. Let R be a (Ψ, Δ)-relation. We define a Ψ-relation $\{\!|\Theta|\!\}_{\mathcal{K}}(R)$ on telescope instantiations, the relational interpretation of Θ at R, as inductively generated by the following rules.

$$\frac{}{\cdot \approx \cdot \in \{\!|\cdot|\!\}_{\mathcal{K}}(R)\langle\psi\rangle} \qquad \frac{\chi \approx \chi' \in \{\!|\Theta|\!\}_{\mathcal{K}}(R)\langle\psi\rangle \qquad M \approx M' \in \Downarrow\{\!|\Theta.\textsc{a}\psi|\!\}_{\mathcal{K}\psi}(R\psi, \chi)}{(\chi, M/a) \approx \chi', M'/a) \in \{\!|\Theta, a : \textsc{a}|\!\}_{\mathcal{K}}(R)\langle\psi\rangle}$$

(Note that this is a relation on terms, not on values—coherent evaluation is built in.)

In order to check that our ultimate construction of the inductive relation generated by a specification is a PER, we will need the simple observation.

Lemma 6.2.7. For any (Ψ, Δ)-relation R, we have $\{\!|\Theta|\!\}_{\mathcal{K}}(\mathit{Sym}^+(R)) \subseteq \mathit{Sym}^+(\{\!|\Theta|\!\}_{\mathcal{K}}(R))$ and $\{\!|\Theta|\!\}_{\mathcal{K}}(\mathit{Trans}^+(R)) \subseteq \mathit{Trans}^+(\{\!|\Theta|\!\}_{\mathcal{K}}(R))$.

Proof. By induction on the shape of Θ and the types within. □

6.2.3 The inductive relation

We now define the (Ψ, Δ)-relation $\mathit{Ind}_{\mathcal{K}}$ induced by a specification $\Psi \Vdash \Delta \blacktriangleright \mathcal{K}$ spec, which we intend to install in a value type system as the interpretation of the syntactic type family $\mathsf{Ind}^{\Delta}_{\mathcal{K}}(-)$. $\mathit{Ind}_{\mathcal{K}}$ will be defined as the least fixed point of an operator $\mathit{Step}^{\mathcal{K}}$ that takes a relation and adds one "layer" of introduction forms. For an inductive type, the introduction forms consist of formal Kan operators and constructors. Thus, $\mathit{Step}^{\mathcal{K}}(R)$ is the union of relations $\mathit{Fcoe}(R)$, $\mathit{Fhcom}(R)$, and $\mathit{Intro}^{\mathcal{K}}_{\ell}(R)$ for each $\ell \in \mathcal{K}$; it is useful to define each of these explicitly so that we can work with them individually in future proofs.

Definition 6.2.8. Given a value (Ψ, Δ)-relation R, some $\mathcal{K}$ in context Ψ, and a label ℓ, we define the value (Ψ, Δ)-relation $\mathit{Intro}_\ell^{\mathcal{K}}(R)$ as inductively generated by the principle $\mathsf{intro}_\ell^{\mathcal{K}'}(\phi; \omega; \chi) \approx \mathsf{intro}_\ell^{\mathcal{K}''}(\phi'; \omega'; \chi') \in \mathit{Intro}_\ell^{\mathcal{K}}(R)\langle\psi, \delta\rangle$ for $\Psi' \Vdash (\psi, \delta) \in (\Psi, \Delta)$ whenever the following hold.

- $\Psi' \Vdash \Delta\psi \blacktriangleright \mathcal{K}\psi = \mathcal{K}'$ spec and $\Psi' \Vdash \Delta\psi \blacktriangleright \mathcal{K}\psi = \mathcal{K}''$ spec.
- $(\ell : \Phi.\Omega.[\delta'; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K}\psi$ for some such constructor data.
- $\Psi' \Vdash \delta'[\phi, \omega] = \delta \in \Delta\psi$.
- $\Psi' \Vdash \phi = \phi' \in \Phi$.
- $\Psi' \Vdash \omega = \omega' \in \Omega\phi$.
- $\chi \approx \chi' \in \{\!|\Theta[\phi, \omega]|\!\}_{\mathcal{K}\psi}(R\psi)$ and $\{\!|\Theta[\phi, \omega]|\!\}_{\mathcal{K}\psi}(R\psi) = \{\!|\Theta[\phi', \omega']|\!\}_{\mathcal{K}\psi}(R\psi)$.
- There is no ξ_i such that $\Psi' \Vdash \xi_i$ satisfied holds.

Definition 6.2.9. Given a (Ψ, Δ)-relation R, we define a (Ψ, Δ)-relation $\mathit{Fcoe}(R)$ as inductively generated by the principle $\mathsf{fcoe}_{x.\delta'}^{r \to s}(M) \approx \mathsf{fcoe}_{x.\delta''}^{r \to s}(M') \in \mathit{Fcoe}(R)\langle\psi, \delta\rangle$ for $\Psi' \Vdash (\psi, \delta) \in (\Psi, \Delta)$ whenever the following hold.

- $\Psi', x : \mathbb{I} \Vdash \delta' = \delta'' \in \Delta\psi$ with $\Psi' \Vdash \delta'[s/x] \in \Delta\psi$.
- $\Psi' \Vdash r, s \in \mathbb{I}$ with $r \neq s$.
- $M \approx M' \in \Downarrow R[\psi, \delta'[r/x]]$.

Definition 6.2.10. Given a Γ-relation R, we define a Γ-relation $\mathit{Fhcom}(R)$ as inductively generated by $\mathsf{fhcom}^{r \to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx \mathsf{fhcom}^{r \to s}(M'; \overrightarrow{\xi_i \hookrightarrow x.N_i'}) \in \mathit{Fhcom}(R)\langle\gamma\rangle$ for $\Psi' \Vdash \gamma \in \Gamma$ whenever the following hold.

- $\Psi' \Vdash r, s \in \mathbb{I}$ with $r \neq s$.
- $M \approx M' \in \Downarrow R\gamma$.
- $\Psi' \Vdash \xi_i \in \mathbb{F}$ for each i, and there is no ξ_i such that $\Psi' \Vdash \xi_i$ satisfied holds.
- $\Psi', \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N_j' \in \Downarrow R\gamma$ for all i, j.
- $\Psi', \xi_i \gg M \approx N_i[r/x] \in \Downarrow R\gamma$ for all i.

The following, which we again need to check that the inductive relation is a PER, is straightforward to check.

Proposition 6.2.11. For any (Ψ, Δ)-relation R, we have $Fcoe(Sym^+(R)) \subseteq Sym^+(Fcoe(R))$ and $Fcoe(Trans^+(R)) \subseteq Trans^+(Fcoe(R))$, and likewise for $Fhcom(-)$ and $Intro_\ell^{\mathcal{K}}(-)$.

Definition 6.2.12. Given a (Ψ, Δ)-relation R and $\Psi \Vdash \Delta \blacktriangleright \mathcal{K}$ spec, we define a (Ψ, Δ)-relation $Step^{\mathcal{K}}(R)$ as follows.

$$Step^{\mathcal{K}}(R) \coloneqq Fcoe(R) \cup Fhcom(R) \cup \bigcup_{\ell \in \mathcal{K}} Intro_\ell^{\mathcal{K}}(R)$$

$Step^{\mathcal{K}}$ is a monotone operator on (Ψ, Δ)-relations. We may therefore define a (Ψ, Δ)-relation $Ind_{\mathcal{K}}$, the *higher inductive relation generated by* $\mathcal{K}$, to be the least fixed-point of $Step^{\mathcal{K}}$. Using Lemma 6.2.7 and Proposition 6.2.11, we see that $Ind_{\mathcal{K}}$ is a (Ψ, Δ)-PER.

6.2.4 Introduction

Although $Ind_{\mathcal{K}}$ is ultimately the relation we want to construct, it will prove prudent to state the introduction rules in greater generality, showing for example that any fixed point of *Fhcom* supports formal compositions.

Definition 6.2.13. Let F be a monotone operator on Γ-relations. We define two derived monotone operators on Γ-relations, $F?$ and F^*, as follows.

$$F?(R) \coloneqq R \cup F(R)$$
$$F^*(R) \coloneqq \mu(S \mapsto R \cup F(S))$$

Given a value (Ψ, Δ)-PER R, we see that formal coercions formed from elements in the coherent extension of R belong to the coherent extension of $Fcoe?(R)$. Note that such terms do not necessarily belong to $Fcoe(R)$ itself: in the case that a formal coercion is of the form $\mathrm{fcoe}^{r \to r}_{x.\delta}(M)$, its value is an element of R, not of $Fcoe(R)$.

Lemma 6.2.14 (Formal coercion introduction). Let R be a (Ψ, Δ)-PER. Then the following rules are validated for all $\Psi' \Vdash \psi \in \Psi$ and $\Psi', x : \mathbb{I} \Vdash \delta = \delta' \in \Delta\psi$.

$$\frac{M \approx M' \in \Downarrow R(\psi, \delta[r/x])}{\mathrm{fcoe}^{r \to s}_{x.\delta}(M) \approx \mathrm{fcoe}^{r \to s}_{x.\delta'}(M') \in \Downarrow Fcoe?(R)[\psi, \delta[s/x]]}$$

$$\frac{M \in \Downarrow R(\psi, \delta[r/x])}{\mathrm{fcoe}^{r \to r}_{x.\delta}(M) \approx M \in \Downarrow Fcoe?(R)[\psi, \delta[r/x]]}$$

Proof. The second rule follows immediately from coherent head expansion, as we have $\mathrm{fcoe}^{r \to r}_{x.\delta}(M)\psi' \longmapsto M\psi'$ for all ψ', together with the inclusion $R \subseteq Fcoe?(R)$. For the

first rule, we use coherent value introduction. Given any $\Psi'' \Vdash \psi' \in \Psi'$, we are in one of two cases. If $r\psi' = s\psi'$, then $\mathrm{fcoe}^{r\to s}_{x.\delta}(M)\psi' \approx \mathrm{fcoe}^{r\to s}_{x.\delta'}(M')\psi' \in \Downarrow R[\psi, \delta[s/x]]\psi'$ holds by $M\psi' \approx M'\psi' \in \Downarrow R[\psi, \delta[r/x]]\psi'$ combined with the reduction rule we have already proven on both sides. If $r\psi' \neq s\psi'$ then both sides are values, and $\mathrm{fcoe}^{r\to s}_{x.\delta}(M)\psi' \approx \mathrm{fcoe}^{r\to s}_{x.\delta'}(M')\psi' \in \mathit{Fcoe}?(R)[\psi, \delta[r/x]]\psi'$ holds by definition of $\mathit{Fcoe}(R)$. □

We can prove the same kind of lemma for constructing terms in the coherent extension of $\mathit{Fhcom}?(R)$ from terms in the extension of R.

Lemma 6.2.15 (Formal composite introduction). Let R be a Γ-PER. Then the following rules are validated for all $\Psi \Vdash \gamma \in \Gamma$, interval terms $\Psi \Vdash r, s \in \mathbb{I}$, and list of constraints $\Psi \Vdash \xi_i \in \mathbb{F}$ for $0 \leq i < n$.

(1)

$$\frac{M \approx M' \in \Downarrow R\gamma \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N'_j \in \Downarrow R\gamma \qquad (\forall i)\ \Psi, \xi_i \gg M \approx N_i[r/x] \in \Downarrow R\gamma}{\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx \mathsf{fhcom}^{r\to s}(M'; \overrightarrow{\xi_i \hookrightarrow x.N'_i}) \in \Downarrow \mathit{Fhcom}?(R)\gamma}$$

(2)

$$\frac{M \in \Downarrow R\gamma \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N_j \in \Downarrow R\gamma \qquad (\forall i)\ \Psi, \xi_i \gg M \approx N_i[r/x] \in \Downarrow R\gamma}{\mathsf{fhcom}^{r\to r}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx M \in \Downarrow \mathit{Fhcom}?(R)\gamma}$$

(3)

$$\frac{\Psi \Vdash \xi_k \text{ satisfied} \qquad M \in \Downarrow R\gamma \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N_j \in \Downarrow R\gamma \qquad (\forall i)\ \Psi, \xi_i \gg M \approx N_i[r/x] \in \Downarrow R\gamma}{\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx N_k[s/x] \in \Downarrow \mathit{Fhcom}?(R)\gamma}$$

Proof. We prove the three rules in reverse order.

(3) By coherent head expansion. For any $\Psi' \Vdash \psi \in \Psi$, there is some minimal $l \leq k$ such that $\xi_l\psi$ is satisfied. Then $\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})\psi \longmapsto N_l[s/x]\psi$, and we have $N_l[s/x] \approx N_k[s/x] \in \Downarrow R\gamma\psi$ by assumption.

(2) Again by coherent head expansion. For any $\Psi' \Vdash \psi \in \Psi$, we are in one of two cases. If there is some minimal k such that $\xi_l\psi$ is satisfied, then $\mathsf{fhcom}^{r\to r}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})\psi \longmapsto N_k[s/x]\psi$ and we have $N_k[s/x] \approx M \in \Downarrow R\gamma\psi$ by assumption. Otherwise, we have $\mathsf{fhcom}^{r\to r}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})\psi \longmapsto M\psi$, and $M\psi$ is in $\Downarrow R\gamma\psi$ by assumption.

(1) By coherent value introduction. For any $\Psi' \Vdash \psi \in \Psi$, we are in one of three cases. If there is some minimal k such that $\xi_l\psi$ is satisfied, then we apply (3) on either side of the equation $N_i\psi \approx N_i'\psi \in \Downarrow R\gamma\psi$, which holds by assumption. If there is no such k but we have $r\psi = s\psi$, then we do the same with (2) and $M\psi \approx M'\psi \in \Downarrow R\gamma\psi$. If we are in neither of these situations, then both terms are values and we have $\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})\psi \approx \mathsf{fhcom}^{r\to s}(M'; \overrightarrow{\xi_i \hookrightarrow x.N_i'})\psi \in \mathit{Fhcom}(R)\gamma\psi$ by definition of $\mathit{Fhcom}(R)$. □

The situation for constructor introduction is a bit more complicated: the coherence of an introduction term depends on the well-behavedness of argument term interpretation, which is used to define the boundary of the constructor in the operational semantics. Conversely, the well-behavedness of argument term interpretation depends on the well-behavedness of prior constructors. We therefore proceed by a mutually inductive argument.

Definition 6.2.16. Given a specification $\mathcal{K}$ and a label $\ell \in \mathcal{K}$, write $|\ell|_{\mathcal{K}}$ for the *height of ℓ in $\mathcal{K}$*, the index at which ℓ occurs in the list $\mathcal{K}$. Given $\mathcal{K}$ and an argument term $\textsc{m}$, define $|\textsc{m}|_{\mathcal{K}}$, the *height of $\textsc{m}$ in $\mathcal{K}$*, to be the maximum height in $\mathcal{K}$ among labels occurring in $\textsc{m}$. We likewise define $|\textsc{a}|_{\mathcal{K}}$ and $|\Theta|_{\mathcal{K}}$ for types and contexts.

Definition 6.2.17. Let $\Psi \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec, a (Ψ, Δ)-relation R, and $n \in \mathbb{N}$ be given. We say that R *interprets $\mathcal{K}, \mathcal{K}'$ below n* when the following two conditions hold for any $\Psi' \Vdash \psi \in \Psi$.

- Given
  - $\Psi' \Vdash \Delta\psi \mid \mathcal{K}\psi \mid \Theta \blacktriangleright \textsc{a} = \textsc{a}'$ atype with $|\Theta|_{\mathcal{K}\psi}$, $|\textsc{a}|_{\mathcal{K}\psi}$, $|\textsc{a}'|_{\mathcal{K}\psi}$ all less than n,
  - $\chi \approx \chi' \in \{\!|\Theta|\!\}_{\mathcal{K}\psi}(R\psi)$,

  we have $\{\!|\Theta.\textsc{a}|\!\}_{\mathcal{K}\psi}(R\psi, \chi) = \{\!|\Theta.\textsc{a}'|\!\}_{\mathcal{K}\psi}(R\psi, \chi')$.
- Given
  - $\Psi' \Vdash \Delta\psi \mid \mathcal{K}\psi \mid \Theta \blacktriangleright \textsc{m} = \textsc{m}' \in \textsc{a}$ with $|\Theta|_{\mathcal{K}\psi}$, $|\textsc{m}|_{\mathcal{K}\psi}$, $|\textsc{m}'|_{\mathcal{K}\psi}$, $|\textsc{a}|_{\mathcal{K}\psi}$ all less than n,
  - $\chi \approx \chi' \in \{\!|\Theta|\!\}_{\mathcal{K}\psi}(R\psi)$,

  we have $(\!|\Theta.\textsc{m}|\!)_{\mathcal{K}\psi}(\chi) \approx (\!|\Theta.\textsc{m}'|\!)_{\mathcal{K}'\psi}(\chi') \in \Downarrow \{\!|\Theta.\textsc{a}|\!\}_{\mathcal{K}\psi}(R\psi, \chi)$.

Note that these operations are always *well-defined*; the condition is that they preserve equality in their inputs (and in the latter case, are in the field of a relation). Note that when these conditions hold, it follows by induction that the interpretation functions for contexts and substitutions of height below n are likewise well-behaved.

Lemma 6.2.18 (Constructor introduction). Let $\Psi \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec, a constructor $(\ell : \Phi.\Omega.[\delta; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K}$, and a (Ψ, Δ)-PER R be given such that R interprets $\mathcal{K}, \mathcal{K}'$ below $|\ell|_{\mathcal{K}}$. Then the following rules are validated.

$$\frac{\Psi \Vdash \phi = \phi' \in \Phi \qquad \Psi \Vdash \omega = \omega' \in \Gamma\phi \qquad \chi \approx \chi' \in \{\!|\Theta[\phi, \omega]|\!\}_{\mathcal{K}}(R)}{\mathsf{intro}^{\mathcal{K}}_{\ell}(\phi; \omega; \chi) \approx \mathsf{intro}^{\mathcal{K}'}_{\ell}(\phi'; \omega'; \chi') \in \Downarrow \mathit{Intro}^{\mathcal{K}}_{\ell}?(R)[\mathsf{id}_{\Psi}, \delta[\phi, \omega]]} \quad (1)$$

$$\frac{\Psi \Vdash \xi_j \text{ satisfied} \qquad \Psi \Vdash \phi \in \Phi \qquad \Psi \Vdash \omega \in \Gamma\phi \qquad \chi \in \{\!|\Theta[\phi, \omega]|\!\}_{\mathcal{K}}(R)}{\mathsf{intro}^{\mathcal{K}}_{\ell}(\phi; \omega; \chi) \approx (\!|\Theta.\mathrm{M}_j[\phi, \omega]|\!)_{\mathcal{K}}(\chi) \in \Downarrow \mathit{Intro}^{\mathcal{K}}_{\ell}?(R)[\mathsf{id}_{\Psi}, \delta[\phi, \omega]]} \quad (2)$$

Proof. We prove the two rules in reverse order.

(2) By coherent head expansion. For any $\Psi' \Vdash \psi \in \Psi$, there is some minimal $k \leq j$ such that $\Psi' \Vdash \xi_k\psi$ satisfied, so $\mathsf{intro}^{\mathcal{K}}_{\ell}(\phi; \omega; \chi)\psi \longmapsto (\!|\Theta.\mathrm{M}_k[\phi, \omega]|\!)_{\mathcal{K}}(\chi)\psi$. From $\Psi \Vdash \Delta \blacktriangleright \mathcal{K}$ spec we can extract $\Psi, \Phi, \Omega, \xi_k, \xi_j \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M}_k = \mathrm{M}_j \in \textsc{ind}(\delta)$. As M_j and M_k come from the entry for ℓ in $\mathcal{K}$, we know that $|\mathrm{M}_k[\phi, \omega]\psi|_{\mathcal{K}} < |\ell|_{\mathcal{K}}$ and $|\mathrm{M}_j[\phi, \omega]\psi|_{\mathcal{K}} < |\ell|_{\mathcal{K}}$. By our assumption that R interprets $\mathcal{K}, \mathcal{K}'$ below $|\ell|_{\mathcal{K}}$, then, we conclude that $(\!|\Theta.\mathrm{M}_k[\phi, \omega]|\!)_{\mathcal{K}}(\chi)\psi \approx (\!|\Theta.\mathrm{M}_j[\phi, \omega]|\!)_{\mathcal{K}}(\chi)\psi \in \Downarrow R[\mathsf{id}_{\Psi}, \delta[\phi, \omega]]\psi$.

(1) By coherent value introduction. For any $\Psi' \Vdash \psi \in \Psi$, we are in one of two cases. If there is some minimal k such that $\Psi' \Vdash \xi_k\psi$ satisfied, then we combine the fact that we have $(\!|\Theta.\mathrm{M}_k[\phi, \omega]|\!)_{\mathcal{K}}(\chi)\psi \approx (\!|\Theta.\mathrm{M}_j(\phi', \omega')|\!)_{\mathcal{K}'}(\chi')\psi$ in the relation $\Downarrow R[\mathsf{id}_{\Psi}, \delta[\phi, \omega]]\psi$, which is derivable as in the proof of (2), with applications of (2) on either side to derive $\mathsf{intro}^{\mathcal{K}}_{\ell}(\phi; \omega; \chi)\psi \approx \mathsf{intro}^{\mathcal{K}'}_{\ell}(\phi'; \omega'; \chi')\psi \in \Downarrow \mathit{Intro}^{\mathcal{K}}_{\ell}?(R)[\mathsf{id}_{\Psi}, \delta[\phi, \omega]]\psi$. If there is no such k, then the terms are values related in $\mathit{Intro}^{\mathcal{K}}_{\ell}(R)[\mathsf{id}_{\Psi}, \delta[\phi, \omega]]\psi$ by definition. □

Lemma 6.2.19 (Interpretation of argument terms). Let $\Psi \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec and let R be a (Ψ, Δ)-PER such that $\mathit{Fcoe}(R) \subseteq R$, $\mathit{Fhcom}(R) \subseteq R$, and $\mathit{Intro}^{\mathcal{K}}_{\ell}(R) \subseteq R$ for every ℓ with $|\ell|_{\mathcal{K}} < n$. Then R interprets $\mathcal{K}, \mathcal{K}'$ below n.

Proof. By strong induction on $n \in \mathbb{N}$ and then an inner mutual induction on the derivation of the argument context, type, and term equalities hypothesized in the three conditions of Definition 6.2.17. We leave the details to the reader, as the proof is tedious but completely straightforward. Each rule for deriving argument term well-typedness corresponds to a rule for ordinary terms we have already established, whether for constructors, fhcom or fcoe terms, functions, or paths. In the case of an hcom and fcoe terms, we apply Lemmas 6.2.14 and 6.2.15, using $\mathit{Fcoe}(R) \subseteq R$ and $\mathit{Fhcom}(R) \subseteq R$—which imply $\mathit{Fcoe}?(R) \subseteq R$

and $\mathit{Fhcom}?(R) \subseteq R$—to get an equation in $\Downarrow R$. For a constructor term, we similarly apply Lemma 6.2.18, taking advantage of the induction hypothesis that R interprets $\mathcal{K}, \mathcal{K}'$ below m for every $m < n$ and the fact that $\mathit{Intro}_\ell^{\mathcal{K}}(R) \subseteq R$. □

Theorem 6.2.20. $\mathit{Ind}_{\mathcal{K}}$ interprets $\mathcal{K}, \mathcal{K}'$ below every $n \in \mathbb{N}$.

Proof. Immediate from the universal property of $\mathit{Ind}_{\mathcal{K}}$ and Lemma 6.2.19. □

Corollary 6.2.21. For any $\Psi \Vdash \Delta$ tel, $\Psi \Vdash \Delta \blacktriangleright \mathcal{K}$ spec, and $\Psi \Vdash \delta \in \Delta$, the Ψ-relation $\mathit{Ind}_{\mathcal{K}}[\mathsf{id}_\Psi, \delta]$ is value-coherent.

Proof. By Lemmas 6.2.14 and 6.2.15 for fcoe and fhcom values respectively, and the combination of Lemma 6.2.18 and Theorem 6.2.20 for intro values. □

6.2.5 Type systems closed under HITs

Having defined the (value-coherent) Ψ-PER corresponding to an inductive specification, it is now straightforward to create a type system closed under these.

Example 6.2.22 (Small type system with HITs). We define an operator H on candidate value type systems: given τ, $H(\tau)$ is generated by the following.

- $H(\tau) \vDash \Psi \Vdash \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta) \approx \mathsf{Ind}_{\mathcal{K}'}^{\Delta'}(\delta') \downarrow R$ whenever
  - $\tau \vDash \Psi \Vdash \Delta = \Delta'$ tel,
  - $\tau \vDash \Psi \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec,
  - $\tau \vDash \Psi \Vdash \delta = \delta' \in \Delta$,
  - $R = \mathit{Ind}_{\mathcal{K}}[\mathsf{id}_\Psi, \delta]$.

Here we sweep a detail under the rug: as we do not know our type system satisfies unicity in the midst of the fixed-point construction, it is important that we make a consistent choice of interpreting relations for the telescopes Δ, Δ' and the types occuring in $\mathcal{K}$ and $\mathcal{K}'$. We have not kept track of these in our definition of $\mathit{Ind}_{\mathcal{K}}[\mathsf{id}_\Psi, \delta]$; as it is clear how to do so, however, we leave this bookkeeping to the reader to imagine.

We define τ_0^H to be the least fixed point of $F \cup H$, where F is as defined in Example 3.1.32.

Example 6.2.23 (Type system with HITs and one universe). We define τ_1^H to be the least fixed point of $F \cup H \cup U(\tau_0^H)$, where U is as defined in Example 3.1.33.

Proposition 6.2.24. τ_0^H and τ_1^H are type systems.

Observe that the universe U of τ_1^H is closed under HITs whose index and types are drawn from U. The parameters of a family of HITs in U, on the other hand, are not required to be of types belonging to U. Suppose, for example, we have some family of HITs $\tau_1^H \vDash \Gamma \gg \Delta \blacktriangleright \mathcal{K}$ spec well-formed in the larger type system. For the induced family of inductive types $\Gamma, \Delta \gg \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\overline{v}_{\Delta})$ type to belong to U, we need only that $\tau_0^H \vDash \Psi \Vdash \Delta\gamma \blacktriangleright \mathcal{K}\gamma = \mathcal{K}\gamma'$ spec holds for every $\Psi \Vdash \gamma = \gamma' \in \Gamma$. This can be the case even if the types in Γ do not themselves belong to τ_0^H. As mentioned in Chapter 5, for example, we will have $A : \mathsf{U}, a_0 : A, a_1 : A \gg \mathsf{Id}(A, a_0, a_1) \in \mathsf{U}$, which requires that the type A of the indices belongs to U but not that the type U of the parameter belongs to U.

We work relative to the type system τ_1^H for the remainder of this chapter, in which we check that the inductive pretypes enjoy the rules we expect from them. The formation rule is immediate by coherent value introduction and stability of the specification judgment—the type and its substitution instances are all values—while the introduction follows from lemmas we have already proven.

Rule 6.2.25 (Pretype formation).

$$\frac{\Psi \Vdash \Delta = \Delta' \ \mathsf{tel} \qquad \Psi \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}' \ \mathsf{spec} \qquad \Psi \Vdash \delta = \delta' \in \Delta}{\Psi \Vdash \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta) = \mathsf{Ind}_{\mathcal{K}'}^{\Delta'}(\delta') \ \mathsf{pretype}}$$

Proof. By coherent value introduction. □

Rule 6.2.26 (Constructor introduction). Let $\Psi \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec and a constructor $(\ell : \Phi.\Omega.[\delta; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K}$ be given.

$$\frac{\Psi \Vdash \phi = \phi' \in \Phi \qquad \Psi \Vdash \omega = \omega' \in \Gamma\phi \qquad \Psi \Vdash \chi = \chi' \in (\!|\Theta[\phi, \omega]|\!)_{\mathcal{K}}^{\Delta}}{\Psi \Vdash \mathsf{intro}_{\ell}^{\mathcal{K}}(\phi; \omega; \chi) = \mathsf{intro}_{\ell}^{\mathcal{K}'}(\phi'; \omega'; \chi') \in \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta[\phi, \omega])}$$

$$\frac{\Psi \Vdash \xi_j \ \mathsf{satisfied} \qquad \Psi \Vdash \phi \in \Phi \qquad \Psi \Vdash \omega \in \Gamma\phi \qquad \Psi \Vdash \chi \in (\!|\Theta[\phi, \omega]|\!)_{\mathcal{K}}^{\Delta}}{\Psi \Vdash \mathsf{intro}_{\ell}^{\mathcal{K}}(\phi; \omega; \chi) = (\!|\Theta.\mathrm{M}_j[\phi, \omega]|\!)_{\mathcal{K}}(\chi) \in \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta[\phi, \omega])}$$

Proof. By Lemma 6.2.18. □

Rule 6.2.27 (Formal coercion introduction). Let $\Psi \Vdash \Delta \blacktriangleright \mathcal{K}$ spec.

$$\frac{\Psi, x : \mathbb{I} \Vdash \delta = \delta' \in \Delta \qquad \Psi \Vdash r, s \in \mathbb{I} \qquad \Psi \Vdash M = M' \in \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta[r/x])}{\Psi \Vdash \mathsf{fcoe}_{x.\delta}^{r \to s}(M) = \mathsf{fcoe}_{x.\delta'}^{r \to s}(M') \in \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta[s/x])}$$

$$\frac{\Psi, x : \mathbb{I} \Vdash \delta \in \Delta \qquad \Psi \Vdash r \in \mathbb{I} \qquad \Psi \Vdash M \in \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta[r/x])}{\Psi \Vdash \mathsf{fcoe}_{x.\delta}^{r \to r}(M) = M \in \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta[r/x])}$$

Proof. By Lemma 6.2.14. □

Rule 6.2.28 (Formal composition introduction). Let $\Psi \Vdash \Delta \blacktriangleright \mathcal{K}$ spec and $\Psi \Vdash \delta \in \Delta$ be given together with interval terms $\Psi \Vdash r, s \in \mathbb{I}$, and constraints $\Psi \Vdash \xi_i \in \mathbb{F}$ for $0 \leq i < n$.

$$\frac{\Psi \Vdash M = M' \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta) \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \Vdash N_i = N'_j \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta) \qquad (\forall i)\ \Psi, \xi_i \Vdash M = N_i[r/x] \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta)}{\Psi \Vdash \mathsf{fhcom}^{r \to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = \mathsf{fhcom}^{r \to s}(M'; \overrightarrow{\xi_i \hookrightarrow x.N'_i}) \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta)}$$

$$\frac{\Psi \Vdash M \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta) \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \Vdash N_i = N_j \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta) \qquad (\forall i)\ \Psi, \xi_i \Vdash M = N_i[r/x] \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta)}{\Psi \Vdash \mathsf{fhcom}^{r \to r}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = M \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta)}$$

$$\frac{\Psi \Vdash \xi_k \text{ satisfied} \qquad \Psi \Vdash M \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta) \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \Vdash N_i = N_j \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta) \qquad (\forall i)\ \Psi, \xi_i \Vdash M = N_i[r/x] \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta)}{\Psi \Vdash \mathsf{fhcom}^{r \to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = N_k[s/x] \in \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta)}$$

Proof. By Lemma 6.2.15. □

6.3 Kan operations

We now show that the inductive pretypes are indeed types by proving the typing rules and boundary conditions for the Kan operators at inductive type. The operational semantics for these operations are shown in Figure 6.6.

We dispatch with composition first: support follows immediately from the existence and well-typedness of formal composites in the inductive type.

Theorem 6.3.1 (Composition). $\Psi \Vdash \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta) = \mathsf{Ind}^{\Delta'}_{\mathcal{K}'}(\delta')$ pretype support homogeneous composition for any $\Psi \Vdash \Delta = \Delta'$ tel, $\Psi \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec, and $\Psi \Vdash \delta = \delta' \in \Delta$.

Proof. Any hcom in a higher inductive type reduces to an fhcom term, as shown in Figure 6.6. Support for homogeneous composition therefore follows immediately from Lemma 6.2.15. □

As one piece of coercion in inductive types, we must be able to apply coercion to a list of arguments. We therefore extend coercion from types to telescopes, coercing them as we might elements of product types. Defined in Figure 6.6, this operator satisfies the following rules.

Coercion along telescopes

$$\overline{\mathsf{coe}}^{r\to s}_{x.\cdot}(\,\cdot\,) \coloneqq \cdot$$
$$\overline{\mathsf{coe}}^{r\to s}_{x.(\Omega,a:A)}(\omega, M/a) \coloneqq (\overline{\mathsf{coe}}^{r\to s}_{x.\Omega}(\omega), \mathsf{coe}^{r\to s}_{x.A[\overline{\mathsf{coe}}^{r\to x}_{x.\Omega}(\omega)]}(M)/a)$$

Parameter coercion

$$\frac{M \longmapsto M'}{\mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(M) \longmapsto \mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(M')}$$

$$\frac{\begin{array}{c}(\ell : \Phi.\Omega.[\delta; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K} \\ (\nexists i)\ \xi_i \text{ satisfied} \qquad (\forall t)\ \omega^t \coloneqq \overline{\mathsf{coe}}^{r\to t}_{x.\omega\phi}(\omega) \qquad (\forall t)\ \chi^t \coloneqq \overline{\mathsf{coe}}^{r\to t}_{x.(\!|\Theta[\phi,\omega^x]|\!)^\Delta_{\mathcal{K}}}(\chi) \\ (\forall i)\ M_i^x \coloneqq \mathsf{pcoe}^{x\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}((\!|\Theta.\mathrm{M}_k[\phi,\omega^x]|\!)_{\mathcal{K}}(\chi^x)) \qquad \delta^x \coloneqq \overline{\mathsf{coe}}^{x\to s}_{x.\Delta}(\delta\omega^x)\end{array}}{\mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(\mathsf{intro}^{\mathcal{K}'}_{\ell}(\phi;\omega;\chi)) \longmapsto \mathsf{fcom}^{s\to r}_{x.\delta^x}(\mathsf{intro}^{\mathcal{K}[s/x]}_{\ell}(\phi;\omega^s;\chi^s); \overrightarrow{\xi_i\phi \hookrightarrow x.M_i^x})}$$

$$\frac{t \neq u \qquad (\nexists i)\ \xi_i \text{ satisfied}}{\begin{array}{c}\mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(\mathsf{fhcom}^{t\to u}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})) \\ \longmapsto \\ \mathsf{fhcom}^{t\to u}(\mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(M); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(N_i)})\end{array}}$$

$$\frac{t \neq u}{\mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(\mathsf{fcoe}^{t\to u}_{y.\delta}(M)) \longmapsto \mathsf{fcoe}^{t\to u}_{y.\mathsf{coe}^{r\to s}_{x.\Delta}(\delta)}(\mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(M))}$$

Coercion

$$\frac{}{\mathsf{coe}^{r\to s}_{x.\mathsf{Ind}^\Delta_{\mathcal{K}}(\delta)}(M) \longmapsto \mathsf{fcoe}^{r\to s}_{x.\overline{\mathsf{coe}}^{x\to s}_{x.\Delta}(\delta)}(\mathsf{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(M))}$$

Composition

$$\frac{}{\mathsf{hcom}^{r\to s}_{\mathsf{Ind}^\Delta_{\mathcal{K}}(\delta)}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto \mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})}$$

Figure 6.6: Operational semantics of coercion and composition in HITs

Lemma 6.3.2 (Telescope coercion). Given $\Psi, x : \mathbb{I} \Vdash \Omega = \Omega'$ tel, the following rules are validated.

$$\frac{\Psi \Vdash r, s \in \mathbb{I} \qquad \Psi \Vdash \omega = \omega' \in \Omega[r/x]}{\Psi \Vdash \overline{\mathsf{coe}}^{r \to s}_{x.\Omega}(\omega) = \overline{\mathsf{coe}}^{r \to s}_{x.\Omega'}(\omega') \in \Omega[s/x]} \qquad \frac{\Psi \Vdash r \in \mathbb{I} \qquad \Psi \Vdash \omega \in \Omega[r/x]}{\Psi \Vdash \overline{\mathsf{coe}}^{r \to r}_{x.\Omega}(\omega) = \omega \in \Omega[r/x]}$$

Proof. By induction on the length of the telescopes. In the case $\Psi, x : \mathbb{I} \Vdash \cdot = \cdot$ tel, the two rules are immediate, as $\omega = \omega' = \overline{\mathsf{coe}}^{r \to s}_{x.\Omega}(\omega) = \overline{\mathsf{coe}}^{r \to s}_{x.\Omega'}(\omega') = \cdot$. Otherwise, we are in the case $\Psi, x : \mathbb{I} \Vdash (\Omega, a : A) = (\Omega', a : A')$ tel where $\Psi, x : \mathbb{I} \Vdash \Omega = \Omega'$ tel and $\Psi, x : \mathbb{I} \Vdash A = A'$ type, so that $\omega = (\chi.M)$ and $\omega' = (\chi', M'/a)$. By induction hypothesis, we know that $\Psi, x : \mathbb{I} \Vdash \overline{\mathsf{coe}}^{r \to x}_{x.\Omega}(\chi) = \overline{\mathsf{coe}}^{r \to x}_{x.\Omega'}(\chi') \in \Omega$, so by substituting in A, A' we get $\Psi, x : \mathbb{I} \Vdash A[\overline{\mathsf{coe}}^{r \to x}_{x.\Omega}(\chi)] = A'[\overline{\mathsf{coe}}^{r \to x}_{x.\Omega'}(\chi')]$ type. As A and A' are Kan, we may coerce M and M' along these lines, obtaining the following.

$$\Psi \Vdash \mathsf{coe}^{r \to s}_{x.A[\overline{\mathsf{coe}}^{r \to x}_{x.\Omega}(\chi)]}(M) = \mathsf{coe}^{r \to s}_{x.A'[\overline{\mathsf{coe}}^{r \to x}_{x.\Omega'}(\chi')]}(M') \in A[\overline{\mathsf{coe}}^{r \to s}_{x.\Omega}(\chi), s/x]$$

Combining this with $\Psi, x : \mathbb{I} \Vdash \overline{\mathsf{coe}}^{r \to x}_{x.\Omega}(\chi) = \overline{\mathsf{coe}}^{r \to x}_{x.\Omega'}(\chi') \in \Omega$ and consulting the definition of $\overline{\mathsf{coe}}$, we thus have $\Psi \Vdash \overline{\mathsf{coe}}^{r \to s}_{x.\Omega, a:A}(\chi, M/a) = \overline{\mathsf{coe}}^{r \to s}_{x.\Omega', a:A'}(\chi', M'/a) \in (\Omega, a : A)[s/x]$. The second rule follows by a similar argument. □

It will be convenient to have a notion of when an open relation or relation on instantiations supports coercion at some syntactic type.

Definition 6.3.3. Given a Γ-relation R, we say that R *supports coercion at* A, A' when $R\gamma$ supports coercion at $A\gamma, A'\gamma'$ for every $\Psi \Vdash \gamma = \gamma' \in \Gamma$.

Definition 6.3.4. We say that a Ψ-PER R on instantiations *supports coercion at telescopes* Ω, Ω' when it validates the following rules for every $\Psi', x : \mathbb{I} \Vdash \psi \in \Psi$ and $\Psi' \Vdash r, s \in \mathbb{I}$.

$$\frac{\omega \approx \omega' \in R\psi[r/x]}{\overline{\mathsf{coe}}^{r \to s}_{x.\Omega\psi}(\omega) \approx \overline{\mathsf{coe}}^{r \to s}_{x.\Omega'\psi}(\omega') \in R\psi[s/x]} \qquad \frac{\omega \in R\psi[r/x]}{\overline{\mathsf{coe}}^{r \to r}_{x.\Omega\psi}(\omega) \approx \omega \in R\psi[r/x]}$$

Recall from Section 5.3 that we separate coercion into two parts: the formal coercions between indices of the family—which we have already shown are well-typed—and *parameter coercion*, which coerces along lines $\Psi, x : \mathbb{I} \Vdash \Delta$ tel and $\Psi, x : \mathbb{I} \Vdash \Delta \blacktriangleright \mathcal{K}$ spec in the specification (and thus in the parameters of the type). We intend the parameter coercion operator, pcoe, is intended to satisfy the following rules.

Declaration of Intent 6.3.5.

$$\frac{\Psi, x : \mathbb{I} \Vdash \Delta = \Delta' \text{ tel} \qquad \Psi, x : \mathbb{I} \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}' \text{ spec} \qquad \Psi \Vdash r, s \in \mathbb{I} \qquad \Psi \Vdash \delta \in \Delta[r/x] \qquad \Psi \Vdash M = M' \in \mathsf{Ind}^{\Delta[r/x]}_{\mathcal{K}[r/x]}(\delta)}{\Psi \Vdash \mathrm{pcoe}^{r\to s}_{x.\Delta\blacktriangleright x.\mathcal{K}}(M) = \mathrm{pcoe}^{r\to s}_{x.\Delta'\blacktriangleright x.\mathcal{K}'}(M') \in \mathsf{Ind}^{\Delta[s/x]}_{\mathcal{K}[s/x]}(\overline{\mathrm{coe}}^{r\to s}_{x.\Delta}(\delta))}$$

$$\frac{\Psi, x : \mathbb{I} \Vdash \Delta \text{ tel} \qquad \Psi, x : \mathbb{I} \Vdash \Delta \blacktriangleright \mathcal{K} \text{ spec} \qquad \Psi \Vdash r \in \mathbb{I} \qquad \Psi \Vdash \delta \in \Delta[r/x] \qquad \Psi \Vdash M \in \mathsf{Ind}^{\Delta[r/x]}_{\mathcal{K}[r/x]}(\delta)}{\Psi \Vdash \mathrm{pcoe}^{r\to r}_{x.\Delta\blacktriangleright x.\mathcal{K}}(M) = M \in \mathsf{Ind}^{\Delta[r/x]}_{\mathcal{K}[r/x]}(\delta)}$$

The proof of this will constitute the bulk of this section. For the remainder, we fix $\Psi \Vdash \Delta = \Delta'$ tel and $\Psi \Vdash \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec. We use the universal property of $\mathit{Ind}_{\mathcal{K}}$—its status as a least pre-fixed-point—to prove that its elements satisfy coercion, defining a sub-PER $\mathit{Pcoe}^{\nu} \subseteq \mathit{Ind}_{\mathcal{K}}$ of "coercible values" and showing it is a pre-fixed-point of $\mathit{Step}^{\mathcal{K}}$.

The reduction of a single coercion applied to a constructor of inductive type may cause component terms to be coerced multiple times; recall from Section 5.1 that we use terms such as $\mathrm{coe}^{x\to s}_{x.A}(F\,(\mathrm{coe}^{r\to x}_{x.C}(P)))$ to correct the boundary in coercion of path constructors. The relation of "once-coercible" values is therefore not necessarily closed under the $\mathit{Intro}^{\mathcal{K}}_{\ell}$ operators, so we define Pcoe^{ν} as relating *infinitely*-coercible values by way of a greatest fixed-point.

Lemma 6.3.6. If F is a monotone operator on Ψ-relations that takes Ψ-PERs to Ψ-PERs, then the greatest post-fixed-point νF is a Ψ-PER.

Proof. Given a Ψ-relation R, write $C(R)$ for the symmetric transitive closure of R. We have $\nu(C \circ F) \subseteq C(F(\nu(C \circ F)))$ by construction. As $\nu(C \circ F))$ is a Ψ-PER and F preserves Ψ-PERs, we have $C(F(\nu(C \circ F))) = F(\nu(C \circ F))$. Hence $\nu(C \circ F)$ is a post-fixed-point of F, and we deduce that $\nu(C \circ F) \subseteq \nu F$. On the other hand, νF is trivially a post-fixed-point of $C \circ F$, as $\nu F \subseteq F(\nu F) \subseteq C(F(\nu F))$. Thus $\nu F \subseteq \nu(C \circ F)$. In sum, we have $\nu F = \nu(C \circ F)$, and so νF is a Ψ-PER. □

Definition 6.3.7 (Coercibility relation). Let R be a value (Ψ, Δ)-relation. We define a new value (Ψ, Δ)-relation $\mathit{Pcoe}^{-1}(R)$ by declaring $V \approx V' \in \mathit{Pcoe}^{-1}(R)\langle\psi, \delta\rangle$ to hold for $\Psi' \Vdash (\psi, \delta) \in (\Psi, \Delta)$ whenever $V \approx V' \in \mathit{Ind}_{\mathcal{K}}\langle\psi, \delta\rangle$ and the following hold for all $\Psi', x : \mathbb{I} \Vdash \psi_x \in \Psi$ and $\Psi' \Vdash r, s \in \mathbb{I}$ such that $\psi_x[r/x] = \psi$.

- $\mathrm{pcoe}^{r\to s}_{x.\Delta\psi_x\blacktriangleright x.\mathcal{K}\psi_x}(W) \approx \mathrm{pcoe}^{r\to s}_{x.\Delta'\psi_x\blacktriangleright x.\mathcal{K}'\psi_x}(W') \in \Downarrow R[\psi_x[s/x], \overline{\mathrm{coe}}^{r\to s}_{x.\Delta\psi_x}(\delta)]$ for each pair of values $W, W' \in \{V, V'\}$.
- $\mathrm{pcoe}^{r\to r}_{x.\Delta\psi_x\blacktriangleright x.\mathcal{K}\psi_x}(W) \approx W \in \Downarrow R[\psi, \delta]$ for each $W \in \{V, V'\}$.

We define $Pcoe^{\nu}$ to be the greatest fixed-point of $Pcoe^{-1}$, which is a Ψ-PER by virtue of Lemma 6.3.6.

We quickly see that not only values but terms in $Pcoe^{-1}$ are coercible.

Lemma 6.3.8 (Parameter coercion from $Pcoe^{-1}$). Any (Ψ, Δ)-PER R validates the following rules for all $\Psi', x : \mathbb{I} \Vdash \psi \in \Psi$, $\Psi' \Vdash r, s \in \mathbb{I}$, and $\Psi' \Vdash \delta \in \Delta\psi[r/x]$.

$$\frac{M \approx M' \in \Downarrow Pcoe^{-1}(R)[\psi[r/x], \delta]}{\mathrm{pcoe}^{r\to s}_{x.\Delta\psi \blacktriangleright x.\mathcal{K}\psi}(M) \approx \mathrm{pcoe}^{r\to s}_{x.\Delta'\psi \blacktriangleright x.\mathcal{K}'\psi}(M') \in \Downarrow R[\psi[s/x], \overline{\mathrm{coe}}^{r\to s}_{x.\Delta\psi}(\delta)]}$$

$$\frac{M \in \Downarrow Pcoe^{-1}(R)(\psi[r/x], \delta)}{\mathrm{pcoe}^{r\to r}_{x.\Delta\psi \blacktriangleright x.\mathcal{K}\psi}(M) \approx M \in \Downarrow R[\psi[r/x], \delta]}$$

Proof. We apply our elimination lemma, Lemma 3.1.38. Here we use that $Pcoe^{-1}(R) \subseteq [\![\mathsf{Ind}^{\Delta}_{\mathcal{K}}(-)]\!]$ by definition. It then suffices to check that these rules hold when the input terms are values in $Pcoe^{-1}(R)[\psi[r/x], \delta]$, in which case the conclusions are also true by definition of $Pcoe^{-1}(R)$. □

We now show that $Pcoe^{\nu}$ is closed under the individual introduction form relations making up $Step^{\mathcal{K}}$: *Fcoe*, *Fhcom*, and $Intro^{\mathcal{K}}_{\ell}$ for $\ell \in \mathcal{K}$. In each case, we first show that parameter coercion applied to the introduction form can be reduced to some application of introduction forms to coercions of the arguments.

Lemma 6.3.9 (Reduction of pcoe on fcoe). The following holds for any (Ψ, Δ)-PER R, substitution $\Psi', x : \mathbb{I} \Vdash \psi \in \Psi$, $\Psi' \Vdash r, s \in \mathbb{I}$, and $\Psi', y : \mathbb{I} \Vdash \delta \in \Delta\psi[r/x]$.

$$\frac{\Psi' \Vdash t, u \in \mathbb{I} \qquad M \in \Downarrow Pcoe^{-1}(R)[\psi[r/x], \delta[t/y]]}{\begin{array}{c}\mathrm{pcoe}^{r\to s}_{x.\mathcal{K}\psi \blacktriangleright x.\Delta\psi}(\mathrm{fcoe}^{t\to u}_{y.\delta}(M)) \\ \approx \\ \mathrm{fcoe}^{t\to u}_{y.\mathrm{coe}^{r\to s}_{x.\Delta}(\delta)}(\mathrm{pcoe}^{r\to s}_{x.\Delta \blacktriangleright x.\mathcal{K}}(M)) \\ \in \\ \Downarrow Fcoe?(R)[\psi[s/x], \overline{\mathrm{coe}}^{r\to s}_{x.\Delta\psi}(\delta[u/y])]\end{array}}$$

Proof. By coherent head expansion. Let $\Psi'' \Vdash \psi' \in \Psi'$ be given. We are in one of two cases. If $t\psi' = u\psi'$, then we have $\mathrm{pcoe}^{r\to s}_{x.\mathcal{K}\psi \blacktriangleright x.\Delta\psi}(\mathrm{fcoe}^{t\to u}_{y.\delta}(M))\psi \longmapsto \mathrm{pcoe}^{r\to s}_{x.\mathcal{K}\psi \blacktriangleright x.\Delta\psi}(M)\psi$, and the reduct is related to our right side by Lemmas 6.3.8 and 6.2.14. If $t\psi' = u\psi'$, then the left side reduces to the right side, which is again in the relation by Lemmas 6.3.8 and 6.2.14.□

Corollary 6.3.10. For any (Ψ, Δ)-PER R, we have $Fcoe(Pcoe^{-1}(R)) \subseteq Pcoe^{-1}(Fcoe?(R))$.

Proof. Given any $V \approx V' \in Fcoe(Pcoe^{-1}(R))\langle\psi,\delta\rangle$, we have that V and V' are fcoe terms with well-typed boundaries. When we apply pcoe to the two, each reduces to a term to which it is related in $Fcoe?(R)$ by Lemma 6.3.9. These two reducts are in turn related to each other in $Fcoe?(R)$, and related to V and V' in $Fcoe?(R)$ when the pcoe is trivial, by Lemmas 6.3.8 and 6.2.14. Thus we have $V \approx V' \in Pcoe^{-1}(Fcoe?(R))\langle\psi,\delta\rangle$. □

Corollary 6.3.11. We have $Fcoe?(Pcoe^\nu) \subseteq Pcoe^\nu$.

Proof. It suffices to show that $Fcoe?$ is a post-fixed-point of $Pcoe^{-1}$. Using Corollary 6.3.10, we have $Fcoe?(Pcoe^\nu) = Fcoe?(Pcoe^{-1}(Pcoe^\nu)) \subseteq Pcoe^{-1}(Fcoe?(Pcoe^\nu))$. □

Lemma 6.3.12 (Reduction of pcoe on fhcom). The following rule is validated for any (Ψ,Δ)-PER R, substitution $\Psi', x : \mathbb{I} \Vdash \psi \in \Psi$, $\Psi' \Vdash r, s \in \mathbb{I}$, and $\Psi' \Vdash \delta \in \Delta\psi[r/x]$.

$$\frac{\begin{array}{c}\Psi' \Vdash t, u \in \mathbb{I} \qquad (\forall i)\ \Psi' \Vdash \xi_i \in \mathbb{F} \qquad M \in \Downarrow Pcoe^{-1}(R)[\psi[r/x],\delta] \\ (\forall i,j)\ \Psi', \xi_i, \xi_j, y : \mathbb{I} \gg N_i \approx N_j \in \Downarrow Pcoe^{-1}(R)[\psi[r/x],\delta] \\ (\forall i)\ \Psi', \xi_i \gg M \approx N_i[t/y] \in \Downarrow Pcoe^{-1}(R)[\psi[r/x],\delta]\end{array}}{\begin{array}{c}\mathsf{pcoe}^{r\to s}_{x.\mathcal{K}\psi \blacktriangleright x.\Delta\psi}(\mathsf{fhcom}^{t\to u}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})) \\ \approx \\ \mathsf{fhcom}^{t\to u}(\mathsf{pcoe}^{r\to s}_{x.\Delta\psi \blacktriangleright x.\mathcal{K}\psi}(M); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{pcoe}^{r\to s}_{x.\Delta \blacktriangleright x.\mathcal{K}}(N_i)}) \\ \in \\ \Downarrow Fhcom?(R)[\psi[s/x], \overline{\mathsf{coe}}^{r\to s}_{x.\Delta\psi}(\delta)]\end{array}}$$

Proof. Again by a straightforward application of coherent head expansion, now using Lemma 6.2.15 to check that the right side is in the desired relation. □

Corollary 6.3.13. For any R, we have $Fhcom(Pcoe^{-1}(R)) \subseteq Pcoe^{-1}(Fhcom?(R))$.

Proof. As with Corollary 6.3.10. □

Corollary 6.3.14. We have $Fhcom?(Pcoe^\nu) \subseteq Pcoe^\nu$.

Proof. As with Corollary 6.3.11. □

Definition 6.3.15. Define $Fcom(R) \coloneqq Fhcom(Fcoe?(R))$.

Lemma 6.3.16. For any R, we have $Fcom(Pcoe^{-1}(R)) \subseteq Pcoe^{-1}(Fcom?(R))$.

Proof. By Corollaries 6.3.10 and 6.3.13. □

Using the fact that $Pcoe^{\nu}$ is closed under formal coercions, we see that it supports not only parameter coercion but coercion in general.

Lemma 6.3.17. $Pcoe^{\nu}$ supports coercion at $\mathsf{Ind}^{\Delta\psi}_{\mathcal{K}\psi}(\overline{v}_\Delta), \mathsf{Ind}^{\Delta'\psi}_{\mathcal{K}'\psi}(\overline{v}_\Delta)$.

Proof. Per Figure 6.6, a coercion in a higher inductive type reduces to a parameter coercion followed by a formal coercion. By Lemma 6.3.8, $Pcoe^{\nu}$ is closed under parameter coercion, and it is likewise closed under formal coercion by Corollary 6.3.11. □

To coerce a constructor term, we must be able to coerce its arguments. Below, we see that we can coerce in the instantiation relation $\{\!|\Theta|\!\}_{\mathcal{K}}(R)$ induced by a relation R that itself supports coercion.

Lemma 6.3.18. Let $\Psi \Vdash \Delta \mid \mathcal{K} \blacktriangleright \Theta = \Theta'$ actx be given. If a (Ψ, Δ)-PER R supports coercion at $\mathsf{Ind}^{\Delta}_{\mathcal{K}}(\overline{v}_\Delta), \mathsf{Ind}^{\Delta'}_{\mathcal{K}'}(\overline{v}_\Delta)$, then $\{\!|\Theta|\!\}_{\mathcal{K}}(R)$ supports coercion at $(\!|\Theta|\!)^{\Delta}_{\mathcal{K}}, (\!|\Theta'|\!)^{\Delta'}_{\mathcal{K}'}$.

Proof. By induction on the derivation of $\Psi \Vdash \Delta \mid \mathcal{K} \blacktriangleright \Theta = \Theta'$ actx, proving an auxiliary lemma for support of coercion in argument types to handle the successor case. Argument types are built from the inductive family, function types, and path types; the proposition thus follows by the arguments that function and path types are Kan, together with the assumption that R supports coercion at the inductive types. □

Coercion in a constructor term applies coercions to the arguments and wraps the result in the same constructor, but also applies a formal heterogeneous composite (fcom, defined in Figure 6.5) to correct the boundary and index. As with the introduction rules, showing the reduction is well-typed for a constructor ℓ requires assuming $Pcoe^{\nu}$ is closed under the preceding constructors.

Lemma 6.3.19 (Reduction of pcoe on intro). Let $\ell \in \mathcal{K}$ be given. Suppose that we have $Intro^{\mathcal{K}}_{\ell'}(Pcoe^{\nu}) \subseteq Pcoe^{\nu}$ for every ℓ' with $|\ell'|_{\mathcal{K}} < |\ell|_{\mathcal{K}}$. Then the following rule is validated for any substitution $\Psi', x : \mathbb{I} \Vdash \psi \in \Psi$, terms $\Psi' \Vdash r, s \in \mathbb{I}$, and $\Psi' \Vdash \delta \in \Delta\psi[r/x]$.

$$\frac{\begin{array}{c}(\ell : \Phi.\Omega.[\delta; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K}\psi \qquad \Psi' \Vdash \Delta\psi[r/x] \blacktriangleright \mathcal{K}\psi[r/x] = \mathcal{K}' \text{ spec} \\ \Psi' \Vdash \phi \in \Phi[r/x] \qquad \Psi' \Vdash \omega \in \Omega[r/x]\phi \qquad \chi \in \Downarrow\{\!|\Theta[r/x][\phi, \omega]|\!\}_{\mathcal{K}}(Pcoe^{\nu}\psi[r/x]) \\ (\forall t)\ \omega^t := \overline{\mathsf{coe}}^{r\to t}_{x.\omega\phi}(\omega) \qquad (\forall t)\ \chi^t := \overline{\mathsf{coe}}^{r\to t}_{x.(\!|\Theta[\phi,\omega^x]|\!)^{\Delta\psi}_{\mathcal{K}\psi}}(\chi) \\ (\forall i)\ M^x_i := \mathsf{pcoe}^{x\to s}_{x.\Delta\psi\blacktriangleright x.\mathcal{K}\psi}((\!|\Theta.\mathrm{M}_k[\phi, \omega^x]|\!)_{\mathcal{K}\psi}(\chi^x)) \qquad \delta^x := \overline{\mathsf{coe}}^{x\to s}_{x.\Delta\psi}(\delta\omega^x)\end{array}}{\begin{array}{c}\mathsf{pcoe}^{r\to s}_{x.\Delta\psi\blacktriangleright x.\mathcal{K}\psi}(\mathsf{intro}^{\mathcal{K}'}_{\ell}(\phi; \omega; \chi)) \\ \approx \\ \mathsf{fcom}^{s\to r}_{x.\delta^x}(\mathsf{intro}^{\mathcal{K}\psi[s/x]}_{\ell}(\phi; \omega^s; \chi^s); \overrightarrow{\xi_i\phi \hookrightarrow x.M^x_i}) \\ \in \\ \Downarrow Fcom?(Intro^{\mathcal{K}}_{\ell}?(Pcoe^{\nu}))[\psi[s/x], \overline{\mathsf{coe}}^{r\to s}_{x.\Delta\psi}(\delta)]\end{array}}$$

Proof. We first check that the right side is well-typed, beginning by verifying that each of the auxiliary terms is well-typed.

- $\Psi', x : \mathbb{I} \Vdash \omega^x \in \Omega\phi$ and $\Psi' \Vdash \omega^r = \omega \in \Omega[r/x]\phi$ by Lemma 6.3.2.
- $\chi^x \in \Downarrow\{\!|\Theta^x|\!\}_{\mathcal{K}}(Pcoe^\nu\psi)$ and $\chi^r \approx \chi \in \Downarrow\{\!|\Theta^r|\!\}_{\mathcal{K}}(Pcoe^\nu\psi[r/x])$, where $\Theta^x = \Theta[\phi, \omega^x]$, by the above properties of ω^x and Lemmas 6.3.17 and 6.3.18.
- $\Psi', x : \mathbb{I} \Vdash \delta^x \in \Delta\psi$ and $\Psi', x : \mathbb{I} \Vdash \delta^s = \delta[s/x]\omega^s \in \Delta\psi[s/x]$ by Lemma 6.3.2.
- $M_i^x \approx M_j^x \in \Downarrow Pcoe^\nu[\psi, \delta^x]$ for all i, j by the above and Lemmas 6.2.19 and 6.3.8.

From the first three of these instantiated at $[s/x]$ and Lemma 6.2.18, we can conclude that $\mathsf{intro}_\ell^{\mathcal{K}\psi[s/x]}(\phi; \omega^s; \chi^s) \in \Downarrow Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)[\psi, \delta^s]$. Said lemma moreover gives us the following boundary equation for each i.

$$\Psi', \xi_i \gg \mathsf{intro}_\ell^{\mathcal{K}\psi[s/x]}(\phi; \omega^s; \chi^s) \approx M_i^s \in \Downarrow Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)[\psi, \delta^s]$$

It then follows from Lemmas 6.2.14 and 6.2.15, combined with Corollaries 6.3.11 and 6.3.14, that the right side is in $\Downarrow Fcom?(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu))[\psi[s/x], \overline{\mathsf{coe}}^{r\to s}_{x.\Delta\psi}(\delta)]$. Moreover, under the assumption of some constraint ξ_i, it is related to M_i^r, which is in turn related to $\mathsf{pcoe}^{r\to s}_{x.\Delta\psi \blacktriangleright x.\mathcal{K}\psi}(\{\!|\Theta.\mathsf{M}_k[r/x][\phi, \omega]|\!\}_{\mathcal{K}'}(\chi))$ by the above.

We proceed with a standard argument by coherent head expansion. Under any substitution $\Psi'' \Vdash \psi' \in \Psi'$, either some $\xi_k\psi'$ holds for a minimal k or there is no such k. In the latter case, the left side reduces to the right side. In the former case, it reduces to $\mathsf{pcoe}^{r\to s}_{x.\Delta\psi \blacktriangleright x.\mathcal{K}\psi}(\{\!|\Theta.\mathsf{M}_k[r/x][\phi, \omega]|\!\}_{\mathcal{K}'}(\chi))$, which we have just seen is equal to the right side in that circumstance. □

Corollary 6.3.20. For any $\ell \in \mathcal{K}$ such that $Intro_{\ell'}^{\mathcal{K}}(Pcoe^\nu) \subseteq Pcoe^\nu$ for every ℓ' with $|\ell'|_{\mathcal{K}} < |\ell|_{\mathcal{K}}$, we have $Intro_\ell^{\mathcal{K}}(Pcoe^\nu) \subseteq Pcoe^{-1}(Fcom?(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)))$.

Proof. As with Corollaries 6.3.10 and 6.3.13. □

Lemma 6.3.21. We have $Intro_\ell^{\mathcal{K}}(Pcoe^\nu) \subseteq Pcoe^\nu$ for all $\ell \in \mathcal{K}$.

Proof. By strong induction on the height of ℓ in $\mathcal{K}$. Suppose that $Intro_{\ell'}^{\mathcal{K}}(Pcoe^\nu) \subseteq Pcoe^\nu$ for every ℓ' with $|\ell'|_{\mathcal{K}} < |\ell|_{\mathcal{K}}$.

Rather than showing $Intro_\ell^{\mathcal{K}}(Pcoe^\nu) \subseteq Pcoe^\nu$ directly, we prove the stronger claim that $Fcom^*(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)) \subseteq Pcoe^\nu$. By the universal property of $Pcoe^\nu$, it suffices to show that $Fcom^*(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu))$ is a post-fixed-point of $Pcoe^{-1}$, *i.e.*, that the following holds.

$$Fcom^*(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)) \subseteq Pcoe^{-1}(Fcom^*(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)))$$

In turn, by universal property of $Fcom^*$, it is enough to show that the right side is a pre-fixed-point of $R \mapsto Intro_\ell^{\mathcal{K}}?(Pcoe^\nu) \cup Fcom(R)$. This splits into two inclusions, which we prove as follows.

- $Intro_\ell^{\mathcal{K}}?(Pcoe^\nu) \subseteq Pcoe^{-1}(Fcom^*(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)))$. This is true by Corollary 6.3.20, using the induction hypothesis.
- $Fcom(Pcoe^{-1}(Fcom^*(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)))) \subseteq Pcoe^{-1}(Fcom^*(Intro_\ell^{\mathcal{K}}?(Pcoe^\nu)))$. This is true by Corollaries 6.3.13 and 6.3.10. □

Theorem 6.3.22 (Coercion). $\Psi' \Vdash \mathsf{Ind}_{\mathcal{K}\psi}^{\Delta\psi}(\delta) = \mathsf{Ind}_{\mathcal{K}'\psi}^{\Delta'\psi}(\delta')$ pretype support coercion for any $\Psi' \Vdash \psi \in \Psi$ and $\Psi' \Vdash \delta = \delta' \in \Delta\psi$.

Proof. Combining Corollaries 6.3.14 and 6.3.11 and Lemma 6.3.21, we can conclude that $Step^{\mathcal{K}}(Pcoe^\nu) \subseteq Pcoe^\nu$ and therefore that $Ind_{\mathcal{K}} \subseteq Pcoe^\nu$. We have $Ind_{\mathcal{K}} \supseteq Pcoe^\nu$ by definition, so the two are equal. Thus this is exactly Lemma 6.3.17. □

Corollary 6.3.23 (Typehood). $\Psi' \Vdash \mathsf{Ind}_{\mathcal{K}\psi}^{\Delta\psi}(\delta) = \mathsf{Ind}_{\mathcal{K}'\psi}^{\Delta'\psi}(\delta')$ type for any $\Psi' \Vdash \psi \in \Psi$ and $\Psi' \Vdash \delta = \delta' \in \Delta\psi$.

6.4 Elimination

Finally, we establish the elimination principle for a higher inductive type. The operational semantics for the eliminator and associated operators are shown in Figure 6.7. The eliminator takes a list of clauses $\mathcal{E}$ of the following format as an argument.

$$\mathcal{E} = (\ell_1 : \overline{v}_{\mathrm{H}_1}.T_1, \ldots, \ell_n : \overline{v}_{\mathrm{H}_n}.T_n)$$

The clause for each constructor ℓ is an open term taking the arguments of that constructor as arguments as well as the results of recursive calls applied to each recursive argument. As an operator that evaluates its principal argument (the element of the inductive type), the proof of its well-typedness proceeds in a manner similar to that for pcoe, although this case is somewhat simpler.

Perhaps the more involved task is *stating* the elimination principle. In particular, we must define the the types of the results of recursive calls at compound types as well as the coherence conditions that the case branches provided for path constructors should satisfy. To do so, we define a new, *dependent* interpretation of the argument type theory.

We first define the dependent interpretation of terms. It is easiest to understand what this means for an argument term of the form $\Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} \in \textsc{ind}(\delta)$ where $\Theta = a_1 : \textsc{ind}(\delta_1), \ldots, a_n : \textsc{ind}(\delta_n)$. Recall that in such a case, we will have $(\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi) \in \mathsf{Ind}_{\mathcal{K}}^{\Delta}(\delta)$

Eliminator

$$\frac{M \longmapsto M'}{\mathsf{elim}(\overline{v}_\delta.h.D; \delta; M; \mathcal{E}) \longmapsto \mathsf{elim}(\overline{v}_\delta.h.D; \delta; M'; \mathcal{E})}$$

$$\frac{\begin{array}{c}(\ell : \Phi.\Omega.[\_; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K} \\ \rho \coloneqq \overline{\mathsf{act}}(\Theta; \overline{v}_\delta.h.\mathsf{elim}(\overline{v}_\delta.h.D; \delta; h; \mathcal{E}); \chi) \qquad (\ell : \overline{v}_\Phi.\overline{v}_\Omega.\overline{v}_\chi.\overline{v}_\rho.T) \in \mathcal{E}\end{array}}{\mathsf{elim}(\overline{v}_\delta.h.D; \delta; \mathsf{intro}^{\mathcal{K}}_\ell(\phi; \omega; \chi); \mathcal{E}) \longmapsto T[\phi, \omega, \chi, \rho]}$$

$$\frac{F_x \coloneqq \mathsf{fcoe}^{r \to x}_{x.\delta'}(M)}{\mathsf{elim}(\overline{v}_\delta.h.D; \delta; \mathsf{fcoe}^{r \to s}_{x.\delta'}(M); \mathcal{E}) \longmapsto \mathsf{coe}^{r \to s}_{x.D[\delta', F_x/h]}(\mathsf{elim}(\overline{v}_\delta.h.D; \delta'[r/x]; M; \mathcal{E}))}$$

$$\frac{F_x \coloneqq \mathsf{fhcom}^{r \to x}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})}{\begin{array}{c}\mathsf{elim}(\overline{v}_\delta.h.D; \delta; \mathsf{fhcom}^{r \to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}); \mathcal{E}) \\ \longmapsto \\ \mathsf{com}^{r \to s}_{x.D[\delta', F_x/h]}(\mathsf{elim}(\overline{v}_\delta.h.D; \delta; M; \mathcal{E}); \overrightarrow{\xi_i \hookrightarrow x.\mathsf{elim}(\overline{v}_\delta.h.D; \delta; N_i; \mathcal{E})})\end{array}}$$

Action of argument types

$$\frac{}{\mathsf{act}(\Theta.\textsc{ind}(\delta); \overline{v}_\delta.h.T; M) \longmapsto T[\delta, M/a]}$$

$$\frac{}{\mathsf{act}(\Theta.(a : A) \to \mathrm{B}; \overline{v}_\Delta.h.T; M) \longmapsto \lambda a.\, \mathsf{act}(\Theta.\mathrm{B}; \overline{v}_\Delta.h.T; M\, a)}$$

$$\frac{}{\mathsf{act}(\Theta.\textsc{Path}(x.\mathrm{A}, \mathrm{M}_0, \mathrm{M}_1); \overline{v}_\Delta.h.T; M) \longmapsto \lambda^{\mathbb{I}} x.\, \mathsf{act}(\Theta.\mathrm{A}; \overline{v}_\Delta.h.T; M\, x)}$$

Action of argument contexts

$$\begin{array}{rcl}\overline{\mathsf{act}}(\,\cdot\,; \overline{v}_\Delta.h.T; \chi) & \coloneqq & \cdot \\ \overline{\mathsf{act}}((\Theta, a : \mathrm{A}); \overline{v}_\Delta.h.T; (\chi, M/a)) & \coloneqq & (\overline{\mathsf{act}}(\Theta; \overline{v}_\Delta.h.T; \chi), \mathsf{act}(\Theta.\mathrm{A}; \overline{v}_\Delta.h.T; M)/a)\end{array}$$

Figure 6.7: Operational semantics of the eliminator and action of argument contexts

for any $\chi \in (a_1 : \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta_1), \ldots, a_n : \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta_n))$: we combine elements of the inductive family according to the shape of M. The dependent interpretation is similar, but we now combine *results of recursive calls* according to the shape of M. For example, suppose we have some target family $\Delta, h{:}\mathsf{Ind}^{\Delta}_{\mathcal{K}}(\overline{v}_\Delta) \gg D$ type into which we might eliminate and a second list $\rho \in (a'_1 : D[\delta_1, a_1\chi/h], \ldots, a'_n : \delta_n, a_n\chi/h)$. The dependent interpretation, written $(\!|\Theta.\mathrm{M}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) \in D[\delta, (\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi)/h]$, tells us what the result of calling the eliminator with clauses $\mathcal{E}$ on $(\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi)$ would be assuming that the results of the recursive calls on the terms χ are given by ρ. This is necessary to express the coherence conditions required of a clause for a path constructor with recursive arguments that appear in its boundary, as in the truncations of Section 5.2.

In general, we must handle hypotheses and terms of argument types other than $\textsc{ind}(\delta)$. Given a list of terms in some context $(\!|\Theta|\!)^{\Delta}_{\mathcal{K}}$, we can apply the eliminator to the elements of the inductive type inside it: given $F \in A \to \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\delta)$, for example, we may apply an eliminator into some family $\Delta, h : \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\overline{v}_\Delta) \gg D$ type pointwise to compute a term of type $(a : A) \to D[\delta, F\,a/h]$. The output type of the action of an eliminator on an instantiation $\chi \in (\!|\Theta|\!)^{\Delta}_{\mathcal{K}}$ is given by the dependent interpretation of argument contexts, written $(\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)$.

The general rule we intend to obtain is the following: given results of recursive calls for a context Θ' and a substitution from Θ' into Θ, we obtain the results of recursive calls for Θ.

$$\frac{\Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \theta \in \Theta \qquad \chi \in (\!|\Theta'|\!)^{\Delta}_{\mathcal{K}} \qquad \rho \in (\!|\Theta'|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)}{(\!|\Theta'.\theta|\!)^{\mathcal{K},\mathcal{E}}_{\Delta.h.D}(\chi;\rho) \in (\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)}$$

The action of argument contexts on the eliminator (and Δ-indexed families of functions more generally), meanwhile, is defined by the act and $\overline{\mathrm{act}}$ operators shown in Figure 6.7. Given a context Θ and map $\Delta, h : \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\overline{v}_\Delta) \gg T \in D$, we will obtain a function $\overline{\mathrm{act}}(\Theta; \overline{v}_\Delta.h.T; -)$ taking instantiations $\chi : (\!|\Theta|\!)^{\Delta}_{\mathcal{K}}$ to instantiations of $(\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)$.

As with non-dependent interpretation, we define the dependent interpretations first as operators on raw syntax.

Definition 6.4.1 (Dependent interpretation of terms). Let Δ be a telescope and $\Delta.h.D$ be a type, $\mathcal{K}$ and $\mathcal{E}$ be constructor and eliminator specifications, and let M be an argument term in context Θ. Let χ and ρ be instantiations for the variables in Θ. We define the de-

pendent interpretation of M, written $(\!|\Theta.\mathrm{M}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho)$, as follows.

$$
\begin{aligned}
(\!|\Theta.a|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) &\coloneqq a\rho \\
(\!|\Theta.\textsc{intro}_\ell(\phi;\omega;\theta)|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) &\coloneqq T[\phi,\omega,\chi',\rho'] \\
\text{where}\quad \chi' &\coloneqq (\!|\Theta.\theta|\!)_{\mathcal{K}}(\chi) \\
\rho' &\coloneqq (\!|\Theta.\theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) \\
&\quad (\ell : \overline{v}_\phi.\overline{v}_\omega.\overline{v}_{\chi'}.\overline{v}_{\rho'}.T) \in \mathcal{E} \\
(\!|\Theta.\textsc{fcoe}^{r\to s}_{x.\delta}(\mathrm{M})|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) &\coloneqq \mathsf{coe}^{r\to s}_{x.D[\delta,F_x/h]}((\!|\Theta.\mathrm{M}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho)) \\
\text{where}\quad F_x &\coloneqq \mathsf{fcoe}^{r\to x}_{x.\delta}((\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi)) \\
(\!|\Theta.\textsc{fhcom}^{r\to s}_{\delta}(\mathrm{M};\overrightarrow{\xi_i \hookrightarrow x.\mathrm{N}_i})|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) &\coloneqq \mathsf{com}^{r\to s}_{x.D[\delta,F_x/h]}(M;\overrightarrow{\xi_i \hookrightarrow N_i}) \\
\text{where}\quad M &\coloneqq (\!|\Theta.\mathrm{M}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) \\
N_i &\coloneqq (\!|\Theta.\mathrm{M}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) \\
F_x &\coloneqq \mathsf{fhcom}^{r\to x}((\!|\Theta.\mathrm{M}|\!)_{\mathcal{K}}(\chi);\overrightarrow{\xi_i \hookrightarrow (\!|\Theta.\mathrm{N}_i|\!)_{\mathcal{K}}(\chi)}) \\
(\!|\Theta.\lambda a.\,\mathrm{N}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) &\coloneqq \lambda a.\,(\!|\Theta.\mathrm{N}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) \\
(\!|\Theta.\mathrm{F}\ M|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) &\coloneqq ((\!|\Theta.\mathrm{F}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho))\,M \\
(\!|\Theta.\lambda^{\mathbb{I}}x.\,\mathrm{M}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) &\coloneqq \lambda^{\mathbb{I}}x.\,(\!|\Theta.\mathrm{N}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) \\
(\!|\Theta.\mathrm{P}\ r|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) &\coloneqq ((\!|\Theta.\mathrm{P}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho))\,r
\end{aligned}
$$

We define $(\!|\Theta.\theta|\!)^{\mathcal{K},\mathcal{E}}_{\Delta.h.D}(\chi;\rho)$ for argument substitutions θ elementwise.

$$
\begin{aligned}
(\!|\Theta.\cdot|\!)^{\mathcal{K},\mathcal{E}}_{\Delta.h.D}(\chi;\rho) &\coloneqq \cdot \\
(\!|\Theta.(\theta,\mathrm{M}/a)|\!)^{\mathcal{K},\mathcal{E}}_{\Delta.h.D}(\chi;\rho) &\coloneqq ((\!|\Theta.\theta|\!)^{\mathcal{K},\mathcal{E}}_{\Delta.h.D}(\chi;\rho),(\!|\Theta.\mathrm{M}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho)/a)
\end{aligned}
$$

Definition 6.4.2 (Dependent interpretation of types). Let Δ be a telescope and $\Delta.h.D$ be a type, $\mathcal{K}$ and $\mathcal{E}$ be constructor and eliminator specifications, and let A be an argument type in context Θ. Let χ and ρ be instantiations for the variables in Θ and let M be a term. We define the dependent interpretation of A, written $(\!|\Theta.\mathrm{A}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho;M)$, as follows.

$$
\begin{aligned}
(\!|\Theta.\textsc{ind}(\delta)|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho;M) &\coloneqq D[\delta,M/h] \\
(\!|\Theta.(a:A)\to \mathrm{B}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho;M) &\coloneqq (a:A)\to (\!|\Theta.\mathrm{B}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho;M\,a) \\
(\!|\Theta.\textsc{Path}(x.\mathrm{A},\mathrm{M}_0,\mathrm{M}_1)|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho;M) &\coloneqq \mathsf{Path}(x.(\!|\mathrm{A}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho;M\,x),M'_0,M'_1) \\
&\qquad \text{where } M'_\varepsilon \coloneqq (\!|\Theta.\mathrm{M}_0|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho)
\end{aligned}
$$

Given an argument context Θ, we define a telescope $(\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)$ elementwise.

$$(\!|\cdot|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\cdot) \coloneqq \cdot$$
$$(\!|\Theta, a : \mathrm{A}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi, M/a) \coloneqq ((\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi), a : (\!|\Theta.\mathrm{A}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi; \overline{v}_{(\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)}; M))$$

We can now state the criteria defining the well-formed lists of clauses $\mathcal{E}$ for the eliminator. To build these up inductively, we define a *partial* specification judgment $\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} \in [\mathcal{K}' \Rightarrow h.D]$, which states that the eliminator $\mathcal{E}$ contains clauses for eliminating from some prefix $\mathcal{K}'$ of a specification $\mathcal{K}$. A complete eliminator specification for $\mathcal{K}$ is then one satisfying $\Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} \in [\mathcal{K} \Rightarrow h.D]$. It is necessary to keep a reference to the complete $\mathcal{K}$ throughout, as recursive clauses should be able to handle arguments not from $\mathsf{Ind}^{\Delta}_{\mathcal{K}'}(-)$ but from $\mathsf{Ind}^{\Delta}_{\mathcal{K}}(-)$.

Definition 6.4.3 (Eliminator specification). The partial eliminator specification judgment, $\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} = \mathcal{E}' \in [\mathcal{K}' \Rightarrow h.D]$, is defined as follows.

$$\frac{}{\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \cdot = \cdot \in [\cdot \Rightarrow h.D]}$$

$$\frac{\begin{array}{c}\Gamma \gg \Delta \blacktriangleright \mathcal{K} \mathbin{@} \ell \Rightarrow (\mathcal{K}' \mid C) \qquad \Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} = \mathcal{E}' \in [\mathcal{K}' \Rightarrow h.D] \\ C = [\Phi; \Omega; \delta; \Theta; \overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}] \qquad \mathrm{R} \coloneqq (\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\overline{v}_{(\!|\Theta|\!)^{\Delta}_{\mathcal{K}}}) \\ \mathrm{H} \coloneqq (\Phi, \Omega, (\!|\Theta|\!)^{\Delta}_{\mathcal{K}}, \mathrm{R}) \qquad \Gamma, \mathrm{H} \gg T = T' \in D[\delta, \mathsf{intro}^{\mathcal{K}}_{\ell}(\overline{v}_{\Phi}; \overline{v}_{\Omega}; \overline{v}_{(\!|\Theta|\!)^{\Delta}_{\mathcal{K}}})/h] \\ (\forall i)\ \Gamma, \mathrm{H}, \xi_i \gg T = (\!|\mathrm{M}_i|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\overline{v}_{(\!|\Theta|\!)^{\Delta}_{\mathcal{K}}}; \overline{v}_{\mathrm{R}}) \in D[\delta, \mathsf{intro}^{\mathcal{K}}_{\ell}(\overline{v}_{\Phi}; \overline{v}_{\Omega}; \overline{v}_{(\!|\Theta|\!)^{\Delta}_{\mathcal{K}}})/h]\end{array}}{\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright (\mathcal{E}, \ell : \overline{v}_{\mathrm{H}}.T) = (\mathcal{E}', \ell : \overline{v}_{\mathrm{H}}.T') \in [(\mathcal{K}', \ell : C) \Rightarrow h.D]}$$

It is now straightforward to show that the dependent interpretation functions are well-behaved when supplied with a well-formed eliminator specification.

Lemma 6.4.4 (Dependent interpretation). Let $\Gamma \gg \Delta = \Delta'$ tel, $\Gamma \gg \Delta \blacktriangleright \mathcal{K} = \mathcal{K}'$ spec, $\Gamma, \Delta, h : \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\overline{v}_{\Delta}) \gg D = D'$ type, and $\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} = \mathcal{E}' \in [\mathcal{K} \Rightarrow h.D]$ be given. Then the following rules are validated.

$$\frac{\Gamma \gg \Delta \mid \mathcal{K} \blacktriangleright \Theta = \Theta'\ \mathsf{actx} \qquad \Gamma \gg \chi = \chi' \in (\!|\Theta|\!)^{\Delta}_{\mathcal{K}}}{\Gamma \gg (\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi) = (\!|\Theta'|\!)^{\Delta'.h.D'}_{\mathcal{K}',\mathcal{E}'}(\chi')\ \mathsf{tel}}$$

$$\frac{\begin{array}{c}\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{A} = \mathrm{A}'\ \mathsf{atype} \\ \Gamma \gg \chi = \chi' \in (\!|\Theta|\!)^{\Delta}_{\mathcal{K}} \qquad \Gamma \gg \rho = \rho' \in (\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi) \qquad \Gamma \gg M = M' \in (\!|\mathrm{A}|\!)^{\Delta}_{\mathcal{K}}(\chi)\end{array}}{\Gamma \gg (\!|\Theta.\mathrm{A}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi; \rho; M) = (\!|\Theta.\mathrm{A}'|\!)^{\Delta'.h.D'}_{\mathcal{K}',\mathcal{E}'}(\chi'; \rho'; M')\ \mathsf{type}}$$

$$\frac{\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta' \blacktriangleright \theta = \theta' \in \Theta \qquad \Gamma \gg \chi = \chi' \in (\!|\Theta'|\!)^{\Delta}_{\mathcal{K}} \qquad \Gamma \gg \rho = \rho' \in (\!|\Theta'|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)}{\Gamma \gg (\!|\Theta'.\theta|\!)^{\mathcal{K},\mathcal{E}}_{\Delta.h.D}(\chi;\rho) = (\!|\Theta'.\theta'|\!)^{\mathcal{K}',\mathcal{E}'}_{\Delta'.h.D'}(\chi';\rho') \in (\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)}$$

$$\frac{\Gamma \gg \Delta \mid \mathcal{K} \mid \Theta \blacktriangleright \mathrm{M} = \mathrm{M}' \in \mathrm{A} \qquad \Gamma \gg \chi = \chi' \in (\!|\Theta|\!)^{\Delta}_{\mathcal{K}} \qquad \Gamma \gg \rho = \rho' \in (\!|\Theta|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi)}{\Gamma \gg (\!|\mathrm{M}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho) = (\!|\mathrm{M}'|\!)^{\Delta'.h.D'}_{\mathcal{K}',\mathcal{E}'}(\chi';\rho') \in (\!|\Theta.\mathrm{A}|\!)^{\Delta.h.D}_{\mathcal{K},\mathcal{E}}(\chi;\rho;(\!|\mathrm{M}|\!)_{\mathcal{K}}(\chi))}$$

Proof. Simultaneously by mutual induction on the argument contexts, types, substitutions, and terms. □

Now we show that the eliminator operator itself is well-typed. For the remainder of this section we fix $\Psi \Vdash \Delta$ tel, $\Psi \Vdash \Delta \blacktriangleright \mathcal{K}$ spec, $\Psi, \Delta, h : \mathsf{Ind}^{\Delta}_{\mathcal{K}}(\overline{v}_{\Delta}) \gg D = D'$ type, and $\Psi \Vdash \Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} = \mathcal{E}' \in [\mathcal{K} \Longrightarrow h.D]$. As with coercion, we define a PER of values that produces well-typed results when supplied to the eliminator, then show that this PER is closed under $\mathit{Step}^{\mathcal{K}}$.

Definition 6.4.5 (Eliminability relation). We define a value (Ψ, Δ)-PER $\mathit{Elim}^{-1} \subseteq \mathit{Ind}_{\mathcal{K}}$ by declaring $V \approx V' \in \mathit{Elim}^{-1}\langle \psi, \delta \rangle$ to hold for $\Psi' \Vdash (\psi, \delta) \in (\Psi, \Delta)$ whenever the following hold.

- $V \approx V' \in \mathit{Ind}_{\mathcal{K}}\langle \psi, \delta \rangle$.
- $\Psi' \Vdash \mathsf{elim}(\overline{v}_{\Delta}.h.D\psi; \delta; W; \mathcal{E}\psi) = \mathsf{elim}(\overline{v}_{\Delta}.h.D'\psi; \delta'; W'; \mathcal{E}'\psi) \in D\psi[\delta, W/h]$ for all pairs $W, W' \in \{V, V'\}$ and δ' with $\Psi' \Vdash \delta = \delta' \in \Delta\psi$.

Lemma 6.4.6 (Extension to terms). For any $\Psi' \Vdash \psi \in \Psi$, $\Psi' \Vdash \delta = \delta' \in \Delta\psi$, and $M \approx M' \in \Downarrow \mathit{Elim}^{-1}\langle \psi, \delta \rangle$, we have the following.

$$\Psi' \Vdash \mathsf{elim}(\overline{v}_{\Delta}.h.D\psi; \delta; M; \mathcal{E}\psi) = \mathsf{elim}(\overline{v}_{\Delta}.h.D'\psi; \delta'; M'; \mathcal{E}'\psi) \in D\psi[\delta, M/h]$$

Proof. By Lemma 3.1.38 and the definition of Elim^{-1}, as the eliminator operator is eager. □

Lemma 6.4.7 (Reduction of elim on fcoe). The following rule is validated for any substitutions $\Psi' \Vdash (\psi, \delta) \in (\Psi, \Delta)$ and $\Psi', x : \mathbb{I} \Vdash \delta' \in \Delta\psi$ with $\Psi' \Vdash \delta'[s/x] = \delta \in \Delta\psi$.

$$\frac{\begin{array}{cc} \Psi' \Vdash r, s \in \mathbb{I} & M \in \Downarrow \mathit{Elim}^{-1}[\psi, \delta'[r/x]] \\ F_x \coloneqq \mathsf{fcoe}^{r \to x}_{x.\delta'}(M) & E \coloneqq \mathsf{elim}(\overline{v}_{\Delta}.h.D\psi; \delta'[r/x]; M; \mathcal{E}\psi) \end{array}}{\Psi' \Vdash \mathsf{elim}(\overline{v}_{\Delta}.h.D\psi; \delta; F_s; \mathcal{E}\psi) = \mathsf{coe}^{r \to s}_{x.D\psi[\delta', F_x/h]}(E) \in D[\delta, F_s/h]}$$

Proof. Note that $\Psi' \Vdash E \in D[\delta'[r/x], M/h]$ holds by Lemma 6.4.6. We proceed by Lemma 3.1.35. For any $\Psi'' \Vdash \psi' \in \Psi'$, we are in one of two cases.

- $r\psi' = s\psi'$. Then $\mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; F_s; \mathcal{E}\psi)\psi' \longmapsto E\psi'$. We know that $\Psi' \Vdash F_r\psi' = M\psi' \in \mathsf{Ind}^\Delta_\mathcal{K}(\delta)\psi'$ by fcoe introduction for the inductive type (Lemma 6.2.14). By the principles required of coercion in D, it follows that $\Psi'' \Vdash E\psi' = \mathsf{coe}^{r\to s}_{x.D\psi[\delta', M/h]}(E)\psi' \in D[\delta, F_r/h]\psi'$.
- $r\psi' \neq s\psi'$. Then $\mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; F_s; \mathcal{E}\psi)\psi' \longmapsto \mathsf{coe}^{r\to s}_{x.D\psi[\delta', F_x/h]}(E)\psi'$, and we know that the reduct is well-typed by coercion in D. □

Corollary 6.4.8. *$Fcoe(Elim^{-1}) \subseteq Elim^{-1}$.*

Proof. As in the proof of Corollary 6.3.10 for coercion: given two applications of fcoe to equal eliminable terms, we can show that the results are equal by applying Lemma 6.4.7.□

Lemma 6.4.9 (Reduction of elim on fhcom). The following rule is validated for any substitutions $\Psi' \Vdash (\psi, \delta) \in (\Psi, \Delta)$.

$$\frac{\begin{array}{c}\Psi' \Vdash r, s \in \mathbb{I} \qquad (\forall i)\ \Psi' \Vdash \xi_i \in \mathbb{F} \\ M \in \Downarrow Elim^{-1}[\psi, \delta] \qquad (\forall i, j)\ \Psi', \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N_j \in \Downarrow Elim^{-1}[\psi, \delta] \\ (\forall i)\ \Psi', \xi_i \gg M \approx N_i[s/x] \in \Downarrow Elim^{-1}[\psi, \delta] \qquad F_x \coloneqq \mathsf{fhcom}^{r\to x}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \\ E \coloneqq \mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; M; \mathcal{E}\psi) \qquad (\forall i)\ E_i \coloneqq \mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; N_i; \mathcal{E}\psi)\end{array}}{\Psi' \Vdash \mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; F_s; \mathcal{E}\psi) = \mathsf{com}^{r\to s}_{x.D\psi[\delta, F_x/h]}(E; \overrightarrow{\xi_i \hookrightarrow x.E_i}) \in D[\delta, F_s/h]}$$

Proof. Straightforward application of coherent head expansion following the pattern of Lemma 6.4.7. □

Corollary 6.4.10. *$Fhcom(Elim^{-1}) \subseteq Elim^{-1}$.*

The case of constructor terms is, as usual, entangled with a property of interpretations, here the well-typedness of the $\overline{\mathsf{act}}$ operator.

Lemma 6.4.11 (Action of argument contexts and types). Let $n \in \mathbb{N}$ and let $R \subseteq Ind_\mathcal{K}$ be a (Ψ, Δ)-relation such that $Fcoe(R) \subseteq R$, $Fhcom(R) \subseteq R$, and $Intro^\mathcal{K}_\ell(R) \subseteq R$ for all ℓ with $|\ell|_\mathcal{K} < n$. Finally, let $\overline{v}_\Delta.h.T, \overline{v}_\Delta.h.T'$ be terms such that $\Psi' \Vdash T[\gamma, M/h] = T'[\gamma', M'/h] \in D[\gamma, M/h]$ for all $\Psi' \Vdash \gamma = \gamma' \in (\Psi, \Delta)$ and $M \approx M' \in \Downarrow R\gamma$. Then the following rule is validated for all $\Psi' \Vdash \psi \in \Psi$.

$$\frac{\Psi' \Vdash \Delta\psi \mid \mathcal{K}\psi \blacktriangleright \Theta = \Theta'\ \mathsf{actx} \qquad |\Theta|_{\mathcal{K}\psi} < n \qquad \chi \approx \chi' \in \{\!|\Theta|\!\}_{\mathcal{K}\psi}(R\psi)}{\Psi' \Vdash \overline{\mathsf{act}}(\Theta; \overline{v}_\Delta.h.T\psi; \chi) = \overline{\mathsf{act}}(\Theta'; \overline{v}_\Delta.h.T'\psi; \chi') \in (\!|\Theta|\!)^{\Delta\psi.h.D\psi}_{\mathcal{K}\psi, \mathcal{E}\psi}(\chi)}$$

Proof. By induction on the derivation of $\Psi' \Vdash \Delta\psi \mid \mathcal{K}\psi \blacktriangleright \Theta = \Theta'$ actx. In the case of a non-empty context, well-typedness of act follows by induction on the argument type, using (trivial) coherent expansion in each case in accordance with the operational semantics shown in Figure 6.7. □

We moreover need to know that the action of argument contexts commutes with argument substitution in an appropriate sense, as captured by the following definition.

Definition 6.4.12. We say that a term $\overline{v}_\Delta.h.T$ *commutes with substitution interpretation below* n when for every $\Psi' \Vdash \psi \in \Psi$, argument substitution $\Psi' \Vdash \Delta\psi \mid \mathcal{K}\psi \mid \Theta' \blacktriangleright \theta \in \Theta$ with $|\Theta'|_{\mathcal{K}}, |\Theta|_{\mathcal{K}}, |\theta|_{\mathcal{K}} < n$, and $\chi \in \{\!|\Theta'|\!\}_{\mathcal{K}}(Elim^{-1})$, we have

$$\Psi' \Vdash \overline{\mathsf{act}}(\Theta; \overline{v}_\Delta.h.T\psi; (\!|\theta|\!)_{\mathcal{K}\psi}(\chi)) = (\!|\theta|\!)^{\mathcal{K}\psi,\mathcal{E}\psi}_{\overline{v}_\Delta.h.D\psi}(\chi; \overline{\mathsf{act}}(\Theta'; \overline{v}_\Delta.h.T\psi; \chi))$$

at the type $(\!|\Theta|\!)^{\overline{v}_\Delta.h.D\psi}_{\mathcal{K}\psi,\mathcal{E}\psi}((\!|\theta|\!)_{\mathcal{K}\psi}(\chi))$.

Lemma 6.4.13 (Naturality). If $Intro^{\mathcal{K}}_\ell(Elim^{-1}) \subseteq Elim^{-1}$ for all ℓ with $|\ell|_{\mathcal{K}} < n$, then $\mathsf{elim}(\overline{v}_\Delta.h.D; -; -; \mathcal{E})$ commutes with substitution interpretation below n.

Proof. By induction on the derivation of $\Psi' \Vdash \Delta\psi \mid \mathcal{K}\psi \mid \Theta' \blacktriangleright \theta \in \Theta$ in the definition of Definition 6.4.12. □

Remark 6.4.14. There are reasonable extensions to the argument term language that would invalidate Lemma 6.4.13. It depends in particular on the fact that the argument type formers are all *negative*, satisfying uniqueness principles up to exact equality. If these were positive—like inductive types—the property would instead hold only up to a path. We expect it would still be possible, although certainly more complicated, to define the eliminator by including "correction" composites in the reduction rules for path constructors as we do in coercion.

Lemma 6.4.15 (Reduction of elim on intro). Let $\ell \in \mathcal{K}$. Suppose that the eliminator $\mathsf{elim}(\overline{v}_\Delta.h.D; -; -; \mathcal{E})$ commutes with substitution interpretation below $|\ell|_{\mathcal{K}}$. Then the following rule is validated for any $\Psi' \Vdash (\psi, \delta) \in (\Psi, \Delta)$.

$$\frac{\begin{array}{c}(\ell : \Phi.\Omega.[\delta'; \Theta.\overrightarrow{\xi_i \hookrightarrow \mathrm{M}_i}]) \in \mathcal{K}\psi \qquad \Psi' \Vdash \Delta\psi \blacktriangleright \mathcal{K}\psi = \mathcal{K}'' \ \mathsf{spec} \\ \Psi' \Vdash \phi \in \Phi \qquad \Psi' \Vdash \omega \in \Omega\phi \qquad \Psi' \Vdash \delta'[\phi, \omega] = \delta \in \Delta\psi \\ \chi \in \Downarrow\{\!|\Theta[\phi, \omega]|\!\}_{\mathcal{K}}(Elim^{-1}\psi) \qquad \rho \coloneqq \overline{\mathsf{act}}(\Theta; \overline{v}_\Delta.h.\mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; h; \mathcal{E}\psi); \chi) \\ \mathrm{H} \coloneqq (\Phi, \Omega, (\!|\Theta|\!)^{\Delta\psi}_{\mathcal{K}\psi}, (\!|\Theta|\!)^{\Delta.h.D\psi}_{\mathcal{K}\psi,\mathcal{E}\psi}(\overline{v}_{(\!|\Theta|\!)^{\Delta\psi}_{\mathcal{K}\psi}})) \qquad (\ell : \overline{v}_{\mathrm{H}}.T) \in \mathcal{E}\psi\end{array}}{\Psi' \Vdash \mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; \mathsf{intro}^{\mathcal{K}''}_\ell(\phi; \omega; \chi); \mathcal{E}\psi) = T[\phi, \omega, \chi, \rho] \in D\psi[\delta, \mathsf{intro}^{\mathcal{K}''}_\ell(\phi; \omega; \chi)/h]}$$

Proof. By coherent head expansion. Let $\Psi'' \Vdash \psi \in \Psi$ be given. We are in one of two cases.

- There is some minimal k such that $\Psi'' \Vdash \xi_k \psi'$ satisfied. Then we have the following reduction.

$$\mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; \mathsf{intro}_\ell^{\mathcal{K}''}(\phi; \omega; \chi); \mathcal{E}\psi)\psi' \longmapsto \mathsf{elim}(\overline{v}_\Delta.h.D\psi; \delta; (\!|\Theta.\mathrm{M}_k[\phi, \omega]|\!)_{\mathcal{K}''}(\chi); \mathcal{E}\psi)\psi'$$

  By Lemma 6.4.13, the latter term is equal to $(\!|\Theta.\mathrm{M}_k[\phi, \omega]|\!)^{\Delta.h.D\psi}_{\mathcal{K}'', \mathcal{E}\psi}(\chi; \rho)\psi'$ as an element of $D\psi[\delta, \mathsf{intro}_\ell^{\mathcal{K}''}(\phi; \omega; \chi)/h]\psi'$, which is in turn equal to $T[\phi, \omega, \chi, \rho]\psi'$ by the requirements on the clause $(\ell : \overline{v}_\mathrm{H}.T) \in \mathcal{E}\psi$ imposed by $\Psi \Vdash \Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} \in [\mathcal{K} \Rightarrow h.D]$.

- There is no k such that $\Psi'' \Vdash \xi_k \psi'$ satisfied. Then the left hand side steps to the right hand side, which is well-typed by $\Psi \Vdash \Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} \in [\mathcal{K} \Rightarrow h.D]$. □

Corollary 6.4.16. $\mathit{Intro}_\ell^{\mathcal{K}}(\mathit{Elim}^{-1}) \subseteq \mathit{Elim}^{-1}$ for all $\ell \in \mathcal{K}$.

Proof. By induction on the height of ℓ, first applying Lemma 6.4.13 and then Lemma 6.4.15.□

Rule 6.4.17 (Elimination).

$$\frac{\Psi \Vdash \Delta\ \mathsf{tel} \qquad \Psi \Vdash \Delta \blacktriangleright \mathcal{K}\ \mathsf{spec} \qquad \Psi, \Delta, h : \mathsf{Ind}^\Delta_\mathcal{K}(\overline{v}_\Delta) \gg D = D'\ \mathsf{type} \qquad \Psi \Vdash \Delta \mid \mathcal{K} \blacktriangleright \mathcal{E} = \mathcal{E}' \in [\mathcal{K} \Rightarrow h.D] \qquad \Psi \Vdash \delta = \delta' \in \Delta \qquad \Psi \Vdash M = M' \in \mathsf{Ind}^\Delta_\mathcal{K}(\delta)}{\Psi \Vdash \mathsf{elim}(\overline{v}_\Delta.h.D; \delta; M; \mathcal{E}) = \mathsf{elim}(\overline{v}_\Delta.h.D'; \delta'; M'; \mathcal{E}') \in D[\delta, M/h]}$$

Proof. By the combination of Lemmas 6.4.7, 6.4.9 and 6.4.15, Lemma 6.4.6, and the definition of $\mathit{Ind}_\mathcal{K}$ as the least fixed-point of $\mathit{Step}^\mathcal{K}$. □

6.5 Strengthening canonicity

By definition of the typing judgment, any well-typed term $\Psi \Vdash M \in \mathsf{Ind}^\Delta_\mathcal{K}(\delta)$ is guaranteed to compute to a value belonging to the inductive relation $\mathit{Ind}_\mathcal{K}$. Such a value is of one of three kinds: it may be a constructor term (intro), but it may also be a formal coercion (fcoe) or composite (fhcom).

This is a broader range of possibilities than one would like, especially in the case that Ψ is empty. For example, we might compute a term $\cdot \Vdash M \in \mathsf{Int}_2$ (an integer modulo 2, as defined in Section 5.1) and get the following unsightly result.

$$\mathsf{fcoe}^{0\to1}_{x.\cdot}(\mathsf{fhcom}^{0\to1}(\mathsf{fhcom}^{0\to1}(\mathsf{fcoe}^{1\to0}_{x.\cdot}(\mathsf{int}(3)); \cdot); 0 \equiv 1 \hookrightarrow \_.\mathsf{int}(8)))$$

We can see an $\mathsf{int}(3)$ somewhere inside, and this term is indeed equal to $\mathsf{int}(3)$ up to a path, but it is buried beneath a pile of "frivolous" formal coercions and composites. Note that there can be no truly significant coercions or composites in a non-indexed type in an empty context. There are no indices to coerce between, and the boundary constraints of a composite will either be true ($0 \equiv 0$ or $0 \equiv 1$), in which case the composite reduces, or false ($0 \equiv 1$ or $1 \equiv 0$), in which case they are irrelevant.

The problem of coercions is easy to solve; we can simply add a rule to the operational semantics so that such frivolous coercions reduce away. (To keep the operational semantics deterministic, we should also add a condition that $\delta \neq \cdot$ on rules that operate on $\mathsf{fcoe}^{r\to s}_{x.\delta}(M)$ values.)

$$\frac{r \neq s}{\mathsf{fcoe}^{r\to s}_{x.\cdot}(M) \longmapsto M}$$

Compositions are less simple, however, because a non-frivolous composite can become a frivolous composite through interval substitution. It takes a few steps to dig up how this becomes a problem; to start, recall the reduction of the inductive type eliminator applied to a formal composite (Figure 6.7).

$$\begin{gathered}\mathsf{elim}(\overline{v}_\delta.h.D; \delta; \mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}); \mathcal{E}) \\ \longmapsto \\ \mathsf{com}^{r\to s}_{x.D[\delta', F_x/h]}(\mathsf{elim}(\overline{v}_\delta.h.D; \delta; M; \mathcal{E}); \overrightarrow{\xi_i \hookrightarrow x.\mathsf{elim}(\overline{v}_\delta.h.D; \delta; N_i; \mathcal{E})})\end{gathered}$$

For this application of the eliminator to be well-typed (*i.e.*, for Lemma 6.4.9 to go through), this reduction must be coherent. If some interval substitution makes the input fhcom frivolous, causing it to reduce to its cap, the right hand side must simplify in a corresponding way. Essentially, if frivolous fhcom terms are made equal to their caps, then, so must *all* frivolous compositions be equal to their caps.[1] In particular, a coercion in a degenerate type line must be equal to its input: $\mathsf{coe}^{r\to s}_{\_.A}(M) = M$. Because of the way composition in path types is defined, ensuring this further requires that any composite with a degenerate type line and tube is equal to its cap: $\mathsf{com}^{r\to s}_{\_.A}(M; \overrightarrow{\xi_i \hookrightarrow \_.N_i}) = M$.

Unfortunately, it is apparently impossible to impose this condition, known variously as *regularity* [CCHM15, Acknowledgements] or *normality* [Awo18, Definition 31], without compromising either univalence or constructivity. The reasons are beyond the scope of this work: the obstruction is composition in the universe, which we have studiously avoided defining. For an illustration of the problems with regularity at the universe type, see [Ang19, §3.4]. More formally, Swan shows that reconciling regularity with univalence requires non-constructivity in a large class of cubical models of type theory [Swa18b].

[1] Alternatively, this reduction rule must be changed in some way—pursuing that option leads to similar conclusions.

Angiuli, Favonia, and Harper therefore solve the original problem by different route, introducing a *validity restriction* that prevents the formation of frivolous composites in the first place [AFH18, Definition 12]. In brief, composite tubes are restricted to certain forms that can never become frivolous by substitution.

Definition 6.5.1 (Validity). A collection $\Psi \Vdash \xi_1, \dots, \xi_n \in \mathbb{F}$ is *valid* when there exist $\Psi \Vdash r \in \mathbb{I}$ and $1 \le i, j \le n$ such that $\Psi, r \equiv 0 \Vdash \xi_i$ satisfied and $\Psi, r \equiv 1 \Vdash \xi_j$ satisfied.

This condition has the following two important properties.

Proposition 6.5.2. If $\Psi \Vdash \overrightarrow{\xi_i} \in \mathbb{F}$ is valid, then $\Psi' \Vdash \overrightarrow{\xi_i \psi} \in \mathbb{F}$ is valid for any $\Psi' \Vdash \psi \in \Psi$.

Proposition 6.5.3. If $\cdot \Vdash \overrightarrow{\xi_i} \in \mathbb{F}$ is valid, then there is some i such that $\cdot \Vdash \xi_i$ satisfied.

The first of these simply checks that validity is stable under interval substitution; this is essential if it is to be a sensible condition to impose. The second implies that any composite with a valid tube in an empty interval context can be simplified.

The solution, then, is to require only that composites exist when the shape of the tube is valid; that is, we add validity as a prerequisite in Definition 3.1.27. Valid composites are sufficient for motivating use of composites, namely coercion in path types. In any case, non-valid composites can be recovered using iterated valid composites [Ang19, Theorem 4.34].

If we add the reduction rule for reducing frivolous formal coercions, impose the validity condition on homogeneous compositions, and add only valid formal composites to inductive types, we can obtain the following improved canonicity theorem for non-indexed HITs in an empty interval context.

Theorem 6.5.4. Assume the above adjustments have been made. Let $\cdot \Vdash \cdot \blacktriangleright \mathcal{K}$ spec and $\cdot \Vdash M \in \mathsf{Ind}^{(\cdot)}_{\mathcal{K}}(\,\cdot\,)$ be given. Then M evaluates to an intro term.

Thus we can run a closed integer modulo 2 and expect to obtain an actual integer as a result. Even simpler, we can define the type of natural numbers as a (particularly degenerate) higher inductive type and have our computations produce actual natural numbers.

Note that we absolutely cannot expect such a strong result for indexed inductive types. In these case, we can still exclude fhcom values with the validity restriction, but there is nothing to be done about formal coercions. The paradigmatic example is the identity type: an element $\cdot \Vdash P \in \mathsf{Id}(A, M, N)$ cannot be guaranteed to evaluate to a refl value, because $\mathsf{Id}(A, M, N)$ is inhabited as soon as there is a *path* from M to N in A.

Chapter 7

Conclusions

7.1 Related work

HITs for ITT Higher inductive types were introduced in the context of univalent intensional type theory at the 2011 Oberwolfach meeting, in discussions between Andrej Bauer, Peter Lumsdaine, Mike Shulman, and Michael Warren (see [Uni13, §6 Notes]). The **HoTT** Book presents many examples of higher inductive types and sketches criteria for a general definition, but definite syntax and semantics for higher inductive types has since taken time to mature. In this and future extensions of **ITT** with higher inductive types, "path" constructors are expressed with identity types. Notably, the reduction rules for eliminators on path constructors are posited only up to identities, not up to exact equality, as such exact equations typically fail to hold in models. In particular, there are many identified-but-not-equal ways to define the action of an eliminator on an identity, and it is not clear that any one deserves to be designated canonical and made to satisfy an exact reduction rule.

There is one obvious and extremely simple schema: include only the quotient type $A /\!\!/ R$ introduced in Section 5.1 (or the inter-derivable pushout). From this minimal base, it is actually possible to build out a number of more sophisticated HITs. Van Doorn [Doo16] and Kraus [Kra16] each give constructions of the propositional truncation using only the quotient, obtaining the truncation as the homotopy colimit of an ω-indexed sequence of types. (Colimits indexed by ω can be defined using quotients and a natural numbers type.) Rijke [Rij17] generalized the latter to construct general n-truncations. While these results are theoretically valuable, the complexity of the definitions makes them unwieldy for computational purposes. Moreover, there are limits to this approach: Lumsdaine and Shulman give an example of a HIT which cannot be constructed from pushouts and the natural numbers [LS20, §9].

Moving up a degree of (a priori) expressivity, Sojakova [Soj14; Soj15; Soj16] intro-

duced the class of W-quotients, also called W-suspensions, and showed these could be characterized as homotopy-initial algebras, building on work on ordinary inductive types in HoTT [AGS12]. These generalize Martin-Löf's W-types [Mar82], which are inductive types of the following form.

$$A : \mathsf{U}, B : A \to \mathsf{U} \gg \textbf{inductive}\ \mathsf{W}(A, B)\ \textbf{where}$$
$$\mid \mathsf{sup}(a : A, f : B\,a \to \mathsf{W}(A, B)) \in \mathsf{W}(A, B)$$

W-types are useful for encoding inductive types with recursive structure; for example, the type of natural numbers may be defined as $\mathsf{W}(\mathsf{Bool}, \lambda b.\, \mathsf{elim}_{\mathsf{Bool}}(\_.\mathsf{U}; b; \mathsf{Void}, \mathsf{U}))$, with $\mathsf{zero} \coloneqq \mathsf{sup}(\mathsf{tt}, \lambda\_.\, \mathsf{abort})$ and $\mathsf{suc}(M) \coloneqq \mathsf{sup}(\mathsf{ff}, \lambda\_.\, M)$.

Expressed in our notation, a W-quotient is an instance of the following parameterized HIT, which enhances the W-type with a path constructor.

$$A, C : \mathsf{U}, B : A \to \mathsf{U}, l, r : C \to A \gg \textbf{inductive}\ \mathsf{W}_{\mathsf{Q}}(B, l, r)\ \textbf{where}$$
$$\mid \mathsf{sup}(a : A, f : B\,a \to \mathsf{W}_{\mathsf{Q}}(B, l, r)) \in \mathsf{W}_{\mathsf{Q}}(B, l, r)$$
$$\mid \mathsf{cell}(c : C, f : B\,(l\,c) \to \mathsf{W}_{\mathsf{Q}}(B, l, r), g : B\,(r\,c) \to \mathsf{W}_{\mathsf{Q}}(B, l, r), x : \mathbb{I}) \in \mathsf{W}_{\mathsf{Q}}(B, l, r)$$
$$[x \equiv 0 \hookrightarrow \mathsf{sup}(l\,c, f) \mid x \equiv 1 \hookrightarrow \mathsf{sup}(r\,c, g)]$$

As an example, Sojakova applies the W-quotient to constructing such types as Int_n, combining the recursive structure of Int and quotient in a single definition. However, because the cell constructor can only connect instances of sup, it is less useful for representing types such as the propositional truncation which have recursive path constructors with non-constructor boundaries. (These can of course be indirectly encoded, as W-quotients subsume ordinary quotients.)

Basold, Geuvers, and van der Weide [BGW17] and Dybjer and Moeneclaey [DM17] present schemata for HITs in ITT that include recursive path constructors, using a grammar of argument types and terms similar to our own. Our work can be seen as a cubical counterpart to these efforts. Kaposi and Kovács [KK18; KK20a] generalize further, defining a schema for *higher inductive-inductive types*, which include indexed higher inductive types as a special case. Their syntax, like ours, also permits path constructors of dimensionality higher than one; unlike in cubical type theory, however, the induction principles for such types rapidly become prohibitively complex, as coherence adjustments must be introduced to reflect the fact that the eliminator does not compute on path constructors up to exact equality.

On the semantic side, Lumsdaine and Shulman [LS20] develop the notion of *cell monad with parameters*, a semantic specification of a higher inductive type, and gave a class of simplicial model categories in which such HITs exist. This class does not obviously correspond to a particular syntactic schema, but includes, in some form, all of the examples we have encountered. As discussed briefly in Section 5.1, however, their universes are not closed under parameterized inductive types: because of the fibrant replacement used

to make HITs Kan, a HIT always lives one universe higher than its type parameters. (By way of contrast, our HITs only depend on their size of their *indices* in this way.) Recent work of Shulman, not yet publicly available, closes this gap.

In almost all cases where a formalism is presented, it fails to be computationally adequate, in part because reduction equations for the eliminator of a HIT on a path constructor are required to hold only up to identity. Of course, any formalism including **HoTT** will already be problematic due to the presence of the univalence axiom. The one exception is Dybjer and Moeneclaey's theory, which targets an interpretation in Hofmann and Streicher's groupoid model [HS98] where these equations hold exactly. However, no adequacy proof is presented in this work, and it is not clear that one should be expected. By contrast, a large part of this work's contribution goes into verifying a coercion algorithm for higher inductive types, essential for canonicity with path-based equality.

From a usability standpoint, cubical HITs are a notable improvement on their ITT equivalents. To begin with, the cubical models justify exact reduction rules for eliminators on path constructors, eliminating a common source of bureaucracy in **HoTT** proofs. More conceptually, cubical type theory scales much more gracefully to higher-dimensional arguments, supporting in its judgmental structure an easily manipulable notion of "n-dimensional term". While Licata and Brunerie [LB15] show that some amount of cubical apparatus can be developed in **HoTT**—defining 2-dimensional "square" types as indexed inductive types and so on—the "native" version is significantly more convenient. As we observe in the introduction of Part III, high-dimensional coherence obligations often arise from iterated induction even when the individual HITs at play are merely one-dimensional. While Part III notes that these are difficult to manage even in cubical type theory, the improvement over **HoTT** is significant.

Cubical HITs Both Cohen, Coquand, Huber, and Mörtberg [CCHM15] and Angiuli, Favonia, and Harper [AFH18], in their respective cubical type theories, include basic examples of higher inductive types. Subsequently, Coquand, Huber, and Mörtberg [CHM18] defined additional examples of higher inductive types in their theory, modeled these in cubical sets, and sketched a (non-indexed) schema. The implementations of the Kan operators for higher inductive types in their setting—De Morgan cubical type theory, which includes additional operations on interval terms—are broadly similar to ours, although they differ in the details. Cavallo, Mörtberg, and Swan [CMS20] show that their cubical type theory, which simultaneously generalizes the cartesian and De Morgan theories, supports at least a circle type, and it is expected that HITs are supported more generally.

Parts of the schema described in this dissertation have been implemented as part of the cartesian proof assistants **RedPRL** [ACFHS18; RedPRL] and `redtt` [redtt], while HITs in the style of [CHM18] have been implemented in a De Morgan cubical mode for **Agda** [Agda; VMA19]. The latter also integrates HITs with its pattern matching machinery,

providing a convenient interface for elimination akin to the case analysis pseudo-code we employ in this thesis.

Quotient inductive types Although we have focused on the combination of higher inductive types with univalence and higher-dimensional equality more generally, higher inductive types are also useful in "zero-dimensional" settings. Higher inductive types truncated at the set level have become known as *quotient inductive types* (QITs) [ACDKN18].

Of particular interest are quotient *inductive-inductive* types (QIITs), which permit the simultaneous definition of several inductive types in which one inductive type may appear as an index to another. A schema for QIITs can serve as a *logical framework*, a schema for defining logics. The syntax of ITT, for example, is higher inductive-inductive type, a collection of interdependent inductively generated judgments (Γ ctx, $\Gamma \vdash A$ type) together with their equality judgments (*e.g.*, $\Gamma \vdash A = A'$ type). This application is explored by Altenkirch and Kaposi [AK16; Kap17]. Dijkstra [Dij17], Altenkirch et al. [ACDKN18], and Kovács and Kaposi [KK20b] define theories and semantics of quotient inductive-inductive types.

Some quotient inductive types can be realized by taking an ordinary inductive type and applying a simple (truncated) quotient in the sense of Section 5.1. In cases with recursive constructors, however, the correctness of this construction may rely on the axiom of choice (AC). Primitive quotient inductive types may therefore be used to avoid relying on AC. This is exploited by Altenkirch, Danielsson, and Kraus to define a partiality monad [ADK17]. Fiore, Pitts, and Steenkamp observe, however, that a combination of (ordinary) inductive-inductive types, quotient types, and size types is sufficient to obtain many quotient inductive types [FPS20].

On the subject of choice, Lumsdaine and Shulman describe a higher inductive type that can be interpreted in Zermelo-Fraenkel set theory with but not without choice [LS20, §9]. As written, this type is not an instance of our schema; it uses definitions by natural number recursion into the type being specified in the boundary of a path constructor, as in the following example.

inductive Ex **where**
| $\mathsf{a} \in \mathsf{Ex}$
| $\mathsf{b}(t : \mathsf{Ex}) \in \mathsf{Ex}$
| $\mathsf{c}(n : \mathsf{Nat}, x : \mathbb{I}) \in \mathsf{Ex} \quad [x \equiv 0 \hookrightarrow \mathsf{a} \mid x \equiv 1 \hookrightarrow \mathsf{elim}_{\mathsf{Nat}}(\_.\mathsf{Ex}; n; \mathsf{a}, \_.t.\mathsf{b}(t))]$

However, this kind of specification can be encoded by taking a function matching the recursive definition as an argument as follows, where $T_{\mathsf{zero}} \coloneqq \mathsf{Path}(\mathsf{Ex}, f\,\mathsf{zero}, \mathsf{a})$ and $T_{\mathsf{suc}} \coloneqq (n : \mathsf{Nat}) \to \mathsf{Path}(\mathsf{Ex}, f\,(\mathsf{suc}(n)), \mathsf{b}(f\,n))$.

inductive Ex **where**
| a ∈ Ex
| b(t : Ex) ∈ Ex
| c(n : Nat, f : Nat → Ex, p : T_{zero}, q : T_{suc}, x : $\mathbb{I}$) ∈ Ex [$x \equiv 0 \hookrightarrow$ a | $x \equiv 1 \hookrightarrow f\ n$]

Our own schema therefore includes a type satisfying the induction principle of Lumsdaine and Shulman's type.

Identity types A primary motivation for constructing indexed inductive types in cubical type theory is to obtain an identity type. As discussed in Section 5.3, while the Path type is a suitable replacement for Id in most regards, it does not satisfy the same J principle—there is a term with the type of J for path types (Lemma 3.2.3), but it does not validate an exact reduction principle on reflexive paths. This failure is explored in detail by Swan [Swa18b].

An alternative construction is therefore necessary to realize identity types in cubical type theory, and in particular to show that cubical type theory interprets **HoTT**. Swan [Swa18a] presents one technique, using the cofibration-trivial fibration factorization of a model structure to obtain identity types from path types; this construction applies in the various structural cubical sets models as well as affine cubical sets. Cohen, Coquand, Huber, and Mörtberg define identity types whose elements are cubical paths paired with constraints on which they are guaranteed reflexive [CCHM15, §9.1]; this construction is also possible in a cartesian setting [ABCFHL19, §2.16], and is analyzed by Swan as a simplified special case of his construction [Swa18a, §6]. Our own construction is distinct from these. In [Cav19], a model-categorical reformulation is presented; it relies instead on a trivial cofibration-fibration factorization and resembles van den Berg and Garner's interpretation of identity types in simplicial sets and related settings [BG12].

7.2 Outlook

We have developed a full-featured schema for higher inductive types in cartesian cubical type theory, complete with a computational interpretation. Our specification grammar accommodates almost all features that appear in the **HoTT** book [Uni13] and subsequent work in homotopy and cubical type theory, including recursive path constructors, recursive arguments of function and path types in the type being constructed, and indices. We expect that similar schemata could now be straightforwardly developed in De Morgan cubical type theory [CCHM15] or Cavallo, Mörtberg, and Swan's minimal cubical type theory [CMS20], taking their examples of higher inductive constructions and generalizing following the pattern developed here. It seems safe to say at this point that the

community's understanding of cubical higher inductive types has reached a state of maturity. Cubical HITs are already being effectively exploited both in synthetic homotopy theory and mathematics more generally [FXG20; MP20; ACMZ21].

The most notable case not handled by our schema is that of the inductive-inductive type [NS10], in which where a type and a family indexed over that type are simultaneously defined by a joint inductive definition. Higher inductive-inductive types are used in the **HoTT** Book to define a type of real numbers [Uni13, §11.3]; as mentioned above, they may also be employed to define type theories within a type theory. We can expect that care is required to give a computational semantics even of *non*-higher inductive-inductive types, given that these include indexed inductive types as a special case. However, we optimistically conjecture that the techniques we use for indexed inductive types—formal coercion values—will generalize gracefully to indexed inductive types, though a proof of correctness would certainly take more work to set up. This view is bolstered by Hugunin's demonstration that inductive-inductive types can be indirectly *derived* from indexed inductive types in cubical type theory [Hug19]. Similarly, we expect that higher *inductive-recursive types* [Dyb00], wherein an inductive type is given simultaneously with a family of types defined *recursively* over it, are now within reach; these have, however, seen less use in homotopy or cubical type theory thus far.

A tempting avenue for future work is to develop a notion of higher *co*inductive type. Coinductive types are the duals of inductive types: where inductive types are least fixed-points generated by constructors, coinductive types are greatest fixed-points supporting destructors [Coq93]. However, while coinductive types have found extensive use in type theory (typically to represent infinite data structures), few applications have been proposed for a higher generalization. Indeed, it is not obvious what the concept dual to higher inductive types should even be.

Finally, the status of higher inductive types in Bezem, Coquand, and Huber's model in affine cubical sets remains unclear. This is not of pressing practical importance, as we can expect that any implementation of HITs in that setting would be more unwieldy than a structural equivalent. We nevertheless believe there is theoretical value in further analysis of the BCH model. For one, affine cubes are indisputably relevant in other settings, as we see in Parts III and IV. Moreover, the BCH model's status as an outlier among the few basic constructive models we know of for homotopy type theory makes it an essential case to understand on the way to any general analysis of such models.

Part III

Internal parametricity

Chapter 8

Introduction

The second part of this thesis is dedicated to realizing a second extension of cubical type theory, this time with *internal parametricity*. Parametricity, introduced by Reynolds [Rey83], is a tool for proving naturality and related properties of type-theoretic constructions. While the technical aspects of this extension are largely orthogonal to those of higher inductive types—shared cubical substrate aside—the strongest motivations for integrating parametricity with cubical type theory come from its potential for reasoning with HITs. To get a sense of that potential, then, let us take an extended look at an example of a higher inductive type, the *smash product*, and the problems it presents for practical theorem proving.

The smash product The smash product originates in homotopy theory as a binary operator on pointed spaces. In synthetic homotopy theory, where types play the role of spaces, a *pointed type* is a type paired with a single "basepoint" element: the universe of pointed types is $\mathsf{U}_* \coloneqq (A : \mathsf{U}) \times A$, its elements thus pairs $\langle A, a_0 \rangle$ with $A \in \mathsf{U}$ and $a_0 \in A$. For the sake of readability, let us adopt a few notational conventions for pointed types. We write pointed types with a subscript $*$, as in $A_*, B_*, \ldots \in \mathsf{U}_*$, then write $A, B, \ldots \in \mathsf{U}$ and $a_0 \in A$, $b_0 \in B$, $\ldots$ for their first and second projections respectively. Given two pointed types $A_*, B_* \in \mathsf{U}_*$, the type of *pointed functions* between them is the type $(A_* \to B_*) \coloneqq (f : A \to B) \times \mathsf{Path}(B, f\, a_0, b_0) \in \mathsf{U}$ of functions that send the basepoint of A to that of B, up to a path. This type is itself pointed, as we always have a unique pointed constant function $\langle \lambda\_.\, b_0, \lambda^{\mathbb{I}}\_.\, b_0 \rangle \in A_* \to B_*$; we write $(A_* \to_* B_*) \coloneqq \langle A_* \to B_*, \langle \lambda\_.\, b_0, \lambda^{\mathbb{I}}\_.\, b_0 \rangle \rangle \in \mathsf{U}_*$ for that pointed type. An isomorphism of pointed types, $A_* \simeq B_*$, is an isomorphism whose underlying function is pointed.

The smash product, written $\wedge_*$, is the natural notion of (monoidal) product for the category of pointed types. In particular, it interacts with the pointed function type in the same way that the ordinary function and product types interact. In categorical terms, $\wedge_*$

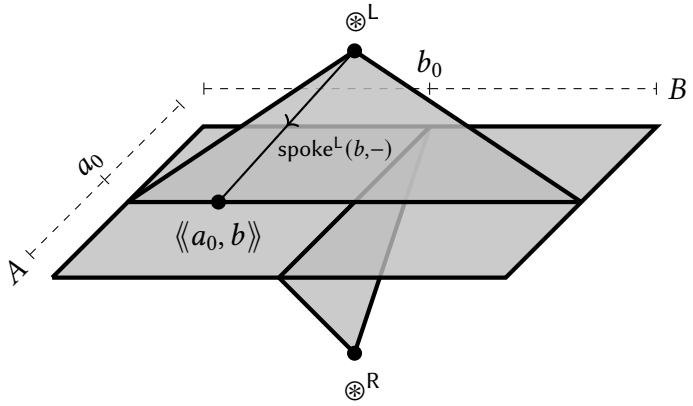
is left adjoint to $\to_*$.

$$(A \times B) \to C \simeq A \to (B \to C)$$
$$(A_* \wedge_* B_*) \to_* C_* \simeq A_* \to_* (B_* \to_* C_*)$$

In cubical type theory, we can define the smash product as the following higher inductive type [Doo18, Definition 4.3.6].

$$\begin{aligned}
&A_* : \mathsf{U}_*, B_* : \mathsf{U}_* \gg \textbf{inductive } A_* \wedge B_* \textbf{ where}\\
&\mid \langle\!\langle a : A, b : B \rangle\!\rangle \in A_* \wedge B_*\\
&\mid \circledast^{\mathsf{L}} \in A_* \wedge B_*\\
&\mid \mathsf{spoke}^{\mathsf{L}}(b : B, x : \mathbb{I}) \in A_* \wedge B_* \quad [x \equiv 0 \hookrightarrow \circledast^{\mathsf{L}} \mid x \equiv 1 \hookrightarrow \langle\!\langle a_0, b \rangle\!\rangle]\\
&\mid \circledast^{\mathsf{R}} \in A_* \wedge B_*\\
&\mid \mathsf{spoke}^{\mathsf{R}}(a : A, x : \mathbb{I}) \in A_* \wedge B_* \quad [x \equiv 0 \hookrightarrow \circledast^{\mathsf{R}} \mid x \equiv 1 \hookrightarrow \langle\!\langle a, b_0 \rangle\!\rangle]
\end{aligned}$$

The smash product of A_* and B_* is a quotient of the product $A \times B$; we start with elements $\langle\!\langle a, b \rangle\!\rangle \in A_* \wedge B_*$ for every $a \in A$ and $b \in B$, then identify all pairs of the form $\langle\!\langle a_0, b \rangle\!\rangle$ or $\langle\!\langle a, b_0 \rangle\!\rangle$. The latter is accomplished by first adding two "hub" points $\circledast^{\mathsf{L}}$ and $\circledast^{\mathsf{R}}$, then equating all terms of the form $\langle\!\langle a_0, b \rangle\!\rangle$ and $\langle\!\langle a, b_0 \rangle\!\rangle$ with $\circledast^{\mathsf{L}}$ and $\circledast^{\mathsf{R}}$ respectively using "spoke" path constructors. We can picture the smash product as in the following image, with the two axes of the product $A \times B$ connected to their respective hub points.



We write $A_* \wedge_* B_*$ for the pointed type $\langle A_* \wedge B_*, \langle\!\langle a_0, b_0 \rangle\!\rangle \rangle$.

The precise definition of the smash product, the intuition behind it, and its use in algebraic topology are not our focus here. Rather, we want to make some generic points about the difficulty of proving results that involve higher inductive types. The smash product appears repeatedly in work on synthetic homotopy, for example in the theses of Brunerie [Bru16, Chapter 4] and Van Doorn [Doo18, §4.3]. In both these cases, a major

pain point is proving that the smash product satisfies some apparently innocuous properties: commutativity, associativity, unit laws, and higher-dimensional coherence laws relating these.

Why are these so difficult to prove? We can start to get a sense by looking at commutativity and associativity. It is simple to define a commutator, a map $A_* \wedge B_* \to B_* \wedge A_*$, by case analysis on the input.

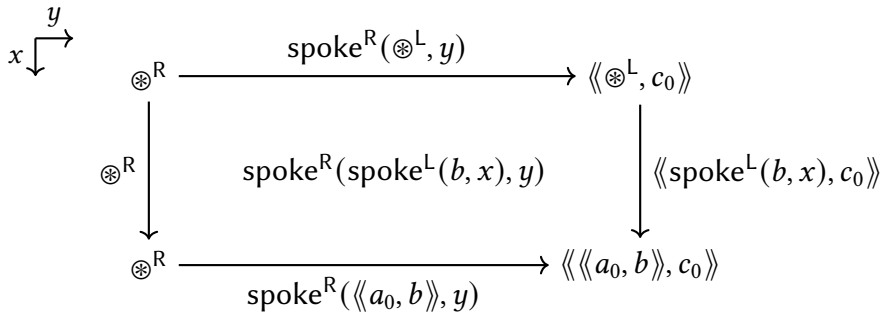
$$\mathsf{commute}\ s \coloneqq \left[\begin{array}{l} \textbf{case}\ s\ \textbf{of} \\ \mid \langle\!\langle a, b \rangle\!\rangle \mapsto \langle\!\langle b, a \rangle\!\rangle \\ \mid \circledast^{\mathsf{L}} \mapsto \circledast^{\mathsf{R}} \\ \mid \mathsf{spoke}^{\mathsf{L}}(b, x) \mapsto \mathsf{spoke}^{\mathsf{R}}(b, x) \\ \mid \circledast^{\mathsf{R}} \mapsto \circledast^{\mathsf{L}} \\ \mid \mathsf{spoke}^{\mathsf{R}}(a, x) \mapsto \mathsf{spoke}^{\mathsf{L}}(a, x) \end{array}\right]$$

Defining an associator, $(A_* \wedge_* B_*) \wedge C_* \to A_* \wedge (B_* \wedge_* C_*)$, is more tedious. Notably, we must go through two layers of case analysis, because the domain $(A_* \wedge_* B_*) \wedge C_*$ of our function contains a twice-iterated smash product.

$$\mathsf{assoc}\ s \coloneqq \left[\begin{array}{l} \textbf{case}\ s\ \textbf{of} \\ \mid \langle\!\langle \langle\!\langle a, b \rangle\!\rangle, c \rangle\!\rangle \mapsto \langle\!\langle a, \langle\!\langle b, c \rangle\!\rangle \rangle\!\rangle \\ \mid \langle\!\langle \circledast^{\mathsf{L}}, c \rangle\!\rangle \mapsto \cdots \\ \mid \langle\!\langle \mathsf{spoke}^{\mathsf{L}}(b, x), c \rangle\!\rangle \mapsto \cdots \\ \mid \langle\!\langle \circledast^{\mathsf{R}}, c \rangle\!\rangle \mapsto \cdots \\ \mid \langle\!\langle \mathsf{spoke}^{\mathsf{R}}(a, x), c \rangle\!\rangle \mapsto \cdots \\ \mid \circledast^{\mathsf{L}} \mapsto \cdots \\ \mid \mathsf{spoke}^{\mathsf{L}}(c, y) \mapsto \cdots \\ \mid \circledast^{\mathsf{R}} \mapsto \cdots \\ \mid \mathsf{spoke}^{\mathsf{R}}(\langle\!\langle a, b \rangle\!\rangle, y) \mapsto \cdots \\ \mid \mathsf{spoke}^{\mathsf{R}}(\circledast^{\mathsf{L}}, y) \mapsto \cdots \\ \mid \mathsf{spoke}^{\mathsf{R}}(\mathsf{spoke}^{\mathsf{L}}(b, x), y) \mapsto \cdots \\ \mid \mathsf{spoke}^{\mathsf{R}}(\circledast^{\mathsf{R}}, y) \mapsto \cdots \\ \mid \mathsf{spoke}^{\mathsf{R}}(\mathsf{spoke}^{\mathsf{R}}(a, x), y) \mapsto \cdots \end{array}\right]$$

The proliferation of cases is daunting, but this would not be a serious problem if each individual branch were straightforward to fill. Unfortunately, this is not the case; the real killer is the *increased dimensionality* that comes from iterated case analysis on higher inductive types. Consider the $\mathsf{spoke}^{\mathsf{R}}(\mathsf{spoke}^{\mathsf{L}}(b, x), y)$ case in the definition of assoc. This is a two-dimensional case, that is, depends on two interval variables x and y; in order for assoc to be well-defined, the boundary of this case's output must agree with the output of the appropriate lower-dimensional cases. Here, $\mathsf{spoke}^{\mathsf{R}}(\mathsf{spoke}^{\mathsf{L}}(b, x), y)$ has the following

boundary.

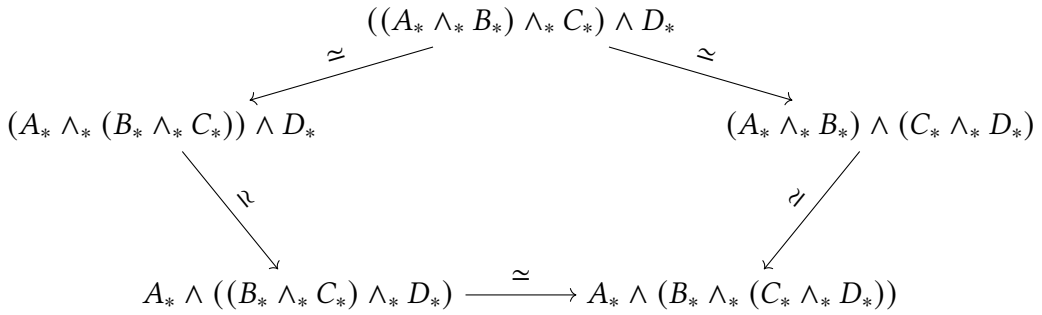


Our goal for the two-dimensional $\mathrm{spoke}^{\mathsf{R}}(\mathrm{spoke}^{\mathsf{L}}(b, x), y)$ case thus depends on what we have written in the zero- and one-dimensional $\circledast^{\mathsf{R}}$, $\mathrm{spoke}^{\mathsf{R}}(\circledast^{\mathsf{L}}, y)$, $\langle\!\langle \mathrm{spoke}^{\mathsf{L}}(b, x), c \rangle\!\rangle$, and $\mathrm{spoke}^{\mathsf{R}}(\langle\!\langle a, b \rangle\!\rangle, y)$ cases. In particular, the complexity of higher-dimensional cases is very sensitive to the complexity of lower-dimensional cases. If, for example, each one-dimensional case involves some non-trivial composition of constructors, then it falls to the two-dimensional constructors to untangle and relate these. Worse, the complexity often depends on essentially arbitrary choices. For example, we can equally well send $\circledast^{\mathsf{L}} \mapsto \circledast^{\mathsf{L}}$, $\circledast^{\mathsf{L}} \mapsto \circledast^{\mathsf{R}}$, $\circledast^{\mathsf{L}} \mapsto \langle\!\langle \circledast^{\mathsf{L}}, c_0 \rangle\!\rangle$, $\circledast^{\mathsf{L}} \mapsto \langle\!\langle \circledast^{\mathsf{R}}, c_0 \rangle\!\rangle$, or $\circledast^{\mathsf{L}} \mapsto \langle\!\langle \langle\!\langle a_0, b_0 \rangle\!\rangle, c_0 \rangle\!\rangle$, given that all of these options are equal up to a path. If we send $\circledast^{\mathsf{L}} \mapsto \circledast^{\mathsf{L}}$, then the $\mathrm{spoke}^{\mathsf{L}}(c, x)$ case requires a path $\circledast^{\mathsf{L}} \rightsquigarrow \langle\!\langle a_0, \langle\!\langle b_0, c \rangle\!\rangle \rangle\!\rangle$, which is easily satisfied by $\mathrm{spoke}^{\mathsf{L}}(\langle\!\langle b_0, c \rangle\!\rangle, x)$; if we send $\circledast^{\mathsf{L}} \mapsto \circledast^{\mathsf{R}}$, the necesssary path $\circledast^{\mathsf{R}} \rightsquigarrow \langle\!\langle a_0, \langle\!\langle b_0, c \rangle\!\rangle \rangle\!\rangle$ cannot be satisfied with a single constructor, instead requiring some composition. Sometimes it is clear which choice produces the simplest higher-dimensional goals, but it is often not.

These problems are further exacerbated if we want to prove any properties of these definitions. For example, we would certainly like to know that the associator is an isomorphism. After defining a candidate inverse assoc^{-1}, we would then have to construct some $\mathsf{inv} \in (s : (A_* \wedge_* B_*) \wedge C_*) \to \mathsf{Path}((A_* \wedge_* B_*) \wedge C_*, \mathrm{assoc}^{-1}(\mathrm{assoc}\, s), s)$. Like the definition of assoc itself, this requires twice-iterated case analysis on the smash product. This time, however, the codomain is a path type, so the dimensionality of each case is bumped up by one. And again, our solution for each case of inv depends in a delicate way on how we have defined assoc and assoc^{-1} as well as the lower-dimensional cases of inv.

Worse still, there are actually infinitely many coherence conditions of increasing dimensionality that one might like the commutator and associator (and unitors) to satisfy. One step up from associativity, we have *Mac Lane's pentagon identity*, which states that the following diagram commutes. In words, the pentagon asserts that the two ways of re-associating from $((A_* \wedge_* B_*) \wedge_* C_*) \wedge D_*$ to $A_* \wedge (B_* \wedge_* (C_* \wedge_* D_*))$—beginning either

by re-associating the inner or the outer triple—produce the same results.



To prove this would require a case analysis on the elements of a *thrice*-iterated smash product. As the codomain is again a path type, this means dealing with four-dimensional terms. Then we might require a further coherence operator relating the different ways of iteratively applying the pentagon identity, and so on without end. (These operators are collectively known as *associahedra*.) To add another wrinkle of complexity, each of these operators should also satisfy a *naturality* condition that lives one dimension higher.

Admittedly, there is (as yet) no pressing need in synthetic homotopy theory to climb very far up this tower of results. However, both Brunerie and Van Doorn's results do rely on the pentagon identity. Brunerie does not give a proof, only sketches an argument that one should be possible. Van Doorn adapts a result of Eilenberg and Kelly [EK66, Chapter 2, Theorem 5.3] to derive the pentagon identity from the *pointed natural isomorphism* $\mathsf{transpose} \in (A_* \wedge_* B_*) \to_* C_* \simeq_* A_* \to_* (B_* \to_* C_*)$. It is plausible that the higher coherences would also follow from this result without further case analysis, which would be a substantial reduction of complexity from the direct approach. However, even the construction of transpose is non-trivial, and Van Doorn leaves one component of the proof unchecked.[1] Brunerie has also attempted to generate proofs of these coherences automatically, using an algorithm that looks for opportunities to apply Martin-Löf's identity elimination rule, but this too reaches the limits of practicality around the pentagon level [Bru18].

Despite the difficulty of verifying these results, the proofs are not at all conceptually interesting, and it is hard to imagine how they could fail to hold. Is it even possible to write down an associator that does not satisfy the pentagon identity? In fact, under sufficient restrictions on the language, *it is not*. We will arrive at this realization by using a classic technique from programming language theory: *parametricity*.

[1]The missing piece is described in [Doo18, Remark 4.3.29]. Briefly, naturality requires an operation relating the isomorphisms $(A_* \wedge_* B_*) \to_* C_* \simeq_* A_* \to_* (B_* \to_* C_*)$ and $(A'_* \wedge_* B'_*) \to_* C'_* \simeq_* A'_* \to_* (B'_* \to_* C'_*)$ whenever there are pointed functions $A_* \to A'_*$, $B_* \to B'_*$, and $C_* \to C'_*$. *Pointed* naturality requires that this operation satisfies a further condition when one of f, g, h is a constant function. Van Doorn relies on pointed naturality in C to obtain the pentagon identity, but does not show that it holds.

Reynolds' parametricity Parametricity, introduced in the seminal work of Reynolds [Rey83], is a property that constrains the behavior of *polymorphic functions*, functions that depend on type variables. Strachey [Str67] distinguishes two varieties of polymorphism: *parametric* and *ad-hoc*. As retold by Reynolds, a parametrically polymorphic function is intuitively one whose behavior is uniform in its type variables, which "does the same thing" no matter how those variables are instantiated, such as the following commutator for the sum/coproduct type.

$$\lambda A.\, \lambda B.\, \lambda c. \left[\begin{array}{l} \textbf{case}\ c\ \textbf{of} \\ \mid \mathsf{inl}(a) \mapsto \mathsf{inr}(a) \\ \mid \mathsf{inr}(b) \mapsto \mathsf{inr}(b) \end{array}\right] \in (A, B : \mathsf{U}) \to A + B \to B + A$$

An ad-hoc polymorphic function, on the other hand, is one whose behavior *does* depend on how its type variables are instantiated, such as the following bizarre function that behaves differently when its type argument is Int.

$$\lambda A.\, \lambda a. \left[\begin{array}{l} \textbf{case}\ A\ \textbf{of} \\ \mid \mathsf{Int} \mapsto 2 \\ \mid \_ \mapsto a \end{array}\right] \in (A : \mathsf{U}) \to A \to A$$

In this telling, the property of being parametric is a syntactic condition: a function is parametric when its definition does not use any case analysis on its type variables. Reynolds' realization was that this syntactic condition implies a powerful semantic property: the existence of an *action on relations*.

Reynolds' original results apply to a formal simple type theory with type variables: the theory with non-dependent functions ($A \to B$), non-dependent products ($A \times B$), and booleans (Bool). Terms are built from function definition and application, pairing and projections, boolean constructors tt and ff, and boolean case analysis. (Reynolds also allows for some additional fixed collection of type and term constants.) Note that no facility for case analysis on types is provided. This type theory has a canonical interpretation in set theory: given an assignment $E = \{X_1 \mapsto S_1, \ldots, X_n \mapsto S_n\}$ of sets to each type variable in a type A, we have an induced set $[\![A]\!]_E$, with function types translated into sets of set-theoretic functions and so on. Likewise, any term $t : A$ has an interpretation as an element $[\![t]\!]_E \in [\![A]\!]_E$. Reynolds' semantic definition of parametric polymorphism is given in terms of this interpretation.

To understand Reynolds' result, let us focus our attention on the simplest case: the type of functions $X \to X$ polymorphic in the type variable X. Given an interpretation $X \mapsto S$, the interpretation of this type is naturally the set of functions $S \to S$.

Definition. A family of set-theoretic functions $(f_S \in S \to S \mid S \in \mathit{Set})$ is *parametric* when it preserves all binary relations: for every pair of sets $S, T \in \mathit{Set}$ and binary relation $R \subseteq S \times T$, if $(s, t) \in R$, then $(f_S(s), f_T(t)) \in R$.

In this vein, Reynolds defines systematically what it means for a set-theoretic family $(a_E \in [\![A]\!]_E \mid E \in \mathit{TypeVars}(A) \to \mathit{Set})$ to be parametric for each type A of the formal theory. The capstone result is then the *abstraction theorem*, which states that the interpretation of any term is parametric.

Definition (Reynolds' abstraction theorem). For any term $t : A$, the induced family $([\![t]\!]_E \in [\![A]\!]_E \mid E \in \mathit{TypeVars}(A) \to \mathit{Set})$ is parametric.

To be parametric is a powerful property. Returning to our example $X \to X$, suppose that $(f_S \in S \to S \mid S \in \mathit{Set})$ is a parametric family. For any set S and element $s \in S$, we have a relation $R \coloneqq \{(s, \star)\} \subseteq S \times \{\star\}$. As $(s, \star) \in R$, we must have $(f_S(s), f_{\{\star\}}(\star)) \in R$ as well—but this means that $f_S(s) = s$. In other words, the *only* parametric family of functions $S \to S$ is the family of identity functions. And by the abstraction theorem, this means that any term $t : X \to X$ is semantically identical to the identity term $\lambda a.\, a : X \to X$.

Now we get an inkling of how we might apply parametricity to the problem of smash products. Certainly we can expect to obtain naturality properties from parametricity: naturality is merely the restriction of parametricity to relations that are graphs of functions. (Wadler memorably dubbed the naturality results that fall out of parametricity "Theorems for Free!" [Wad89].) In fact, we can go even farther. For example, we can show that any parametric candidate associator $\mathsf{assoc} \in (A_* \wedge_* B_*) \wedge_* C_* \to_* A_* \wedge_* (B_* \wedge_* C_*)$ is an isomorphism so long as it is not the constant function $\lambda\_.\, \langle\!\langle a_0, \langle\!\langle b_0, c_0 \rangle\!\rangle \rangle\!\rangle$; it is then a simple matter to exclude the latter case by testing assoc on small inputs. We can even show that any parametric assoc satisfies the pentagon identity.

Internalizing parametricity We deviate from Reynolds' original program, which is based in a set-theoretic denotational semantics, by instead adapting the more recent system of *internal parametricity* developed by Bernardy and Moulin [BM12; BM13; BCM15]. In an internally parametric type theory, the consequences of parametricity are available *within* the theory, rather than holding of an external interpretation (in set theory or otherwise). In particular, in a computational type theory, the operators that implement the abstraction theorem are themselves computational; thus we are able to obtain results without departing from our computational conception of type theory.

On another level, internal parametricity is attractive to us because its realization shares many features with cubical type theory, right down to the crucial use of an interval object. Recall that, as captured by univalence, a line of types $x : \mathbb{I} \gg A \in \mathsf{U}$ corresponds to an isomorphism $e \in A[0/x] \simeq A[1/x]$; moreover, terms $x : \mathbb{I} \gg a \in A$ in that line correspond to paths $e\,(a[0/x]) \rightsquigarrow a[1/x]$ relating their endpoints across the isomorphism. Recall also that all constructions have an action on paths: given a function $f \in A \to B$ and a path $p \in \mathsf{Path}(A, a_0, a_1)$, we can apply f to p pointwise to obtain a path $\lambda^{\mathbb{I}} x.\, f\,(p\,x) \in \mathsf{Path}(B, f\,a_0, f\,a_1)$. The combined consequence of these facts is

that every polymorphic function in cubical type theory has an action on isomorphisms: given some $f \in (X : \mathsf{U}) \to X \to X$ and an isomorphism $e : A \simeq B$, for example, we have $e\,(f\,a) \rightsquigarrow f\,b$ whenever $e\,a \rightsquigarrow b$. Internal parametricity operates on the same principle: now, lines $\boldsymbol{x} : \mathbf{I} \gg A \in \mathsf{U}$ corresponding to relations $R \in A[0/x] \times A[1/x] \to \mathsf{U}$ and terms $\boldsymbol{x} : \mathbf{I} \gg a \in A$ to proofs of $R\,\langle a[0/x], a[1/x]\rangle$. By exploiting the action of terms on the parametric equivalent of paths—which we call *bridges*, following Nuyts et al. [NVD17]—we obtain a theory in which all terms are guaranteed to act on relations.

Outline In this part, we extend the cubical framework of Part I to incorporate *bridge interval variables*, using many of the same strategies as in Part I, followed by the type formers of internal parametricity employed by Bernardy, Coquand, and Moulin [BCM15]. The definition of the extended framework and construction of an instance occurs in Chapter 9; we also remark there on the similarities and distinctions between the behavior of the two kinds of interval. There is little explicit interaction between the "path" and "bridge" elements of the combined type theory, and so this chapter is largely a retelling of [BCM15]. However, cubical equality notably improves the theory around the parametricity primitives—much as we have seen it do with functions, universes, and quotients in previous parts.

In Chapter 10, we apply parametric cubical type theory to prove a number of results, showing how classical consequences of parametricity are validated, establishing some methodology of internal parametricity, and proving the promised results concerning the smash product. In Chapter 11, we explore a formalism for internally parametric type theory and a presheaf model thereof, developing a novel treatment of affine interval variables and simplifying some aspects of Bernardy, Coquand, and Moulin's model by relying on cubical equality. We discuss related and future work in Chapter 12.

This part depends heavily on Part I, but is largely independent of Part II; we do need an intuitive understanding of higher inductive types to prove results involving the smash product, of course, but our intent is that an intuitive understanding is sufficient.

Chapter 9

Parametric cubical type theory

We now enrich the cubical type theory defined in Part I with a *bridge interval*—an interval for internal parametricity—and its attendant type formers and operators, as developed by Bernardy, Coquand, and Moulin [BCM15]. At each stage, we will be revisiting a cubical element from a new angle. Unsurprisingly, the bridge interval parallels the path interval, as do bridge types path types. In addition, the function extensionality principle is paralleled by a new extent operator, while V types are paralleled by Gel types.

Of course, none of these parallels are exact. The differences between cubical and parametric type theory have two sources. The first is mundane: we do not expect to able to coerce along relations as we can along equivalences, so the bridge interval comes with no notion of coercion or composition. Happily, this means the parametric extension is rather less technically involved than the cubical. The second is more interesting: the bridge interval does not support *contraction*. This means we are prohibited from performing substitutions, like the one shown below, which substitute the same bridge variable for two different variables.

$$z : \mathbb{I} \Vdash (z/x, z/y) \in (x : \mathbb{I}, y : \mathbb{I}) \quad \text{✘}$$

In the traditional parlance of substructural logic, bridge interval hypotheses are *affine*. The differences between the cubical constructs and their cubical equivalents nearly all flow from this single modification to the interval theory.

The original cubical model of identity types, the BCH model of Bezem, Coquand, and Huber [BCH13], also used an affine interval. Bernardy, Coquand, and Moulin's internal parametricity therefore naturally adopted the same structure [BCM15]. Cubical type theory later drifted to a structural approach; affinity is more problematic for higher inductive types, and is simply unnecessarily complex when a structural interval will do. We will see here that it is, on the other hand, indispensable for internal parametricity.

9.1 The bridge interval

The first step is to extend the theory of interval contexts and substitutions from Section 3.1.1 with the new bridge interval, which exists in parallel with the cubical path interval. For the most part, we will not repeat the cubical elements here; instead we present only the new components, which are typically either definitions of new judgments or extensions of existing inductively defined judgments by new rules.

9.1.1 Interval contexts and substitutions

Definition 9.1.1 (Interval contexts). We extend the interval context judgment Ψ ictx, specified in Definition 3.1.2, by adding extension by a bridge interval as a context former.

$$\frac{\Psi \text{ ictx}}{(\Gamma, \boldsymbol{x} : \mathbf{I}) \text{ ictx}}$$

Definition 9.1.2 (Bridge interval elements). $\Psi \Vdash \boldsymbol{r} \in \mathbf{I}$ holds when $\boldsymbol{r} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{1}$, or $\boldsymbol{r} = \boldsymbol{x}$ for some $(\boldsymbol{x} : \mathbf{I}) \in \Psi$.

We see our first difference between the two intervals in the definition of substitution. Recall that we define substitutions into a context with a path interval hypothesis as shown below.

$$\frac{\Psi' \Vdash \psi \in \Psi \qquad \Psi' \Vdash r \in \mathbb{I}}{\Psi' \Vdash (\psi, r/x) \in (\Psi, x : \mathbb{I})}$$

As described above, we intend the bridge interval to be affine, so we cannot define substitutions into contexts with a bridge hypothesis in the same way; it is easy to construct a contraction substitution from this rule. Intuitively, a substitution $\Psi' \Vdash \psi \in (\Psi, \boldsymbol{x} : \mathbf{I})$ should still consist of two components: a substitution $\Psi' \Vdash \psi' \in \Psi$ and a bridge term $\Psi' \Vdash \boldsymbol{r} \in \mathbf{I}$. In this case, however, we want to also ensure that the same variable is not used twice between ψ' and $\boldsymbol{r}$: in other words, if $\boldsymbol{r}$ is a variable in Ψ', then ψ' should not use that variable.

To express this condition, we define an *interval restriction* operation that removes an interval variable from its context. Here we adapt the nominal restriction operation from Cheney's *nominal type theory* [Che12], which likewise extends type theory with a new kind of affine hypothesis; we only adjust the definition to accommodate the constants $\mathbf{0}$ and $\mathbf{1}$. Restriction by these has no effect: while we cannot duplicate variables, we can use constants freely.

Definition 9.1.3 (Restriction for interval contexts). We define the restriction of an interval context Ψ by a bridge interval term $\Psi \Vdash \boldsymbol{r} \in \mathbf{I}$, written $\Psi \setminus \boldsymbol{r}$, as follows.

$$\begin{aligned}
\Psi \setminus \mathbf{0} &\coloneqq \Psi \\
\Psi \setminus \mathbf{1} &\coloneqq \Psi \\
(\Psi, y : \mathbb{I}) \setminus \boldsymbol{x} &\coloneqq (\Psi \setminus \boldsymbol{x}), y : \mathbb{I} \\
(\Psi, \boldsymbol{y} : \mathbf{I}) \setminus \boldsymbol{x} &\coloneqq \begin{cases} \Psi & \text{if } \boldsymbol{x} = \boldsymbol{y} \\ (\Psi \setminus \boldsymbol{x}), \boldsymbol{y} : \mathbf{I} & \text{otherwise} \end{cases}
\end{aligned}$$

Definition 9.1.4 (Interval substitutions). We extend the interval substitution judgment $\Psi' \Vdash \psi \in \Psi$, specified in Definition 3.1.4, by the following rule.

$$\frac{\Psi' \Vdash \boldsymbol{r} \in \mathbf{I} \qquad \Psi' \setminus \boldsymbol{r} \Vdash \psi \in \Psi}{\Psi' \Vdash (\psi, \boldsymbol{r}/\boldsymbol{x}) \in (\Psi, \boldsymbol{x} : \mathbf{I})}$$

To construct the identity substitution $\boldsymbol{x} : \mathbf{I}, \boldsymbol{y} : \mathbf{I} \Vdash (\boldsymbol{x}/\boldsymbol{x}, \boldsymbol{y}/\boldsymbol{y}) \in (\boldsymbol{x} : \mathbf{I}, \boldsymbol{y} : \mathbf{I})$, we must show that $\boldsymbol{x} : \mathbf{I}, \boldsymbol{y} : \mathbf{I} \setminus \boldsymbol{y} \Vdash (\boldsymbol{x}/\boldsymbol{x}) \in (\boldsymbol{x} : \mathbf{I})$, which is to say that $\boldsymbol{x} : \mathbf{I} \Vdash (\boldsymbol{x}/\boldsymbol{x}) \in (\boldsymbol{x} : \mathbf{I})$. Here we have no problem. If we try to type the forbidden "$\boldsymbol{z} : \mathbf{I} \Vdash (\boldsymbol{z}/\boldsymbol{x}, \boldsymbol{z}/\boldsymbol{y}) \in (\boldsymbol{x} : \mathbf{I}, \boldsymbol{y} : \mathbf{I})$", on the other hand, we find we need the evidently nonsensical "$\cdot \Vdash (\boldsymbol{z}/\boldsymbol{x}) \in (\boldsymbol{x} : \mathbf{I})$".

Finally, we add equations on bridge interval terms to the language of constraints. While a path constraint may identify any pair of terms, $r \equiv s$, we only allow the identification of a bridge interval term with a constant. This reflects the affine nature of these terms: the only way two bridge variables can become equal is if they both become the same constant. More practically, while the general path constraints are apparently necessary to implement coercion in V types in this theory—see the discussion of *diagonal cofibrations* in [CMS20]—such a need does not arise in the bridge theory.

Definition 9.1.5 (Closed constraint judgments). We extend the constraint and constraint satisfaction judgments, specified in Definition 3.1.21, by the following.

$$\frac{\Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad \boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}}{\Psi \Vdash (\boldsymbol{r} \equiv \boldsymbol{\varepsilon}) \in \mathbb{F}} \qquad\qquad \frac{\boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}}{\Psi \Vdash \boldsymbol{\varepsilon} \equiv \boldsymbol{\varepsilon} \text{ satisfied}}$$

Much as composition with path constraints is necessary to implement coercion in path types (Figure 3.2), we will need bridge constraints to do the same for bridge types.

In theory, these additions to the interval theory could invalidate theorems we already have proven for cubical type theory; for example, some Kan operation might rely on analyzing the shape of constraints. In practice, however, it is easy to check that this is not the case.

9.1.2 Type systems and open judgments

Next, we introduce the type theory proper. There is no change in the underlying definitions of operational semantics, Ψ-relation, and type system; we simply repeat Definitions 3.1.5, 3.1.6 and 3.1.16 with the adjusted theory of interval contexts and substitutions. We do, however, need to modify the induced judgments, in particular the term context judgment and the closing and general substitution judgments.

The well-formed contexts and closing substitutions are simple enough to extend, following exactly the pattern of interval contexts and substitutions.

Definition 9.1.6 (Contexts). We extend the context judgment $\Gamma = \Gamma'\ \mathsf{ctx}$, specified in Definition 3.1.23, as follows.

$$\frac{\Gamma = \Gamma'\ \mathsf{ctx}}{(\Gamma, \boldsymbol{x} : \mathbf{I}) = (\Gamma', \boldsymbol{x} : \mathbf{I})\ \mathsf{ctx}}$$

Definition 9.1.7 (Closing substitutions). We extend the closing substitution judgment $\Psi \Vdash \gamma = \gamma' \in \Gamma$, specified in Definition 3.1.23, as follows.

$$\frac{\Psi \setminus \boldsymbol{r} \Vdash \gamma = \gamma' \in \Gamma \qquad \Psi \Vdash \boldsymbol{r} \in \mathbf{I}}{\Psi \Vdash (\gamma, \boldsymbol{r}/\boldsymbol{x}) = (\gamma', \boldsymbol{r}/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{I})}$$

In a substitution $\Psi, \boldsymbol{y} : \mathbf{I} \Vdash (\gamma, \boldsymbol{y}/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{I})$, we are guaranteed that the terms in γ do not use $\boldsymbol{y}$. To put it another way, any instantiation of the context $(\Gamma, \boldsymbol{x} : \mathbf{I})$ will have the property that the terms supplied for Γ will not intersect with any variable substituted for $\boldsymbol{x}$. Note that the same does not apply to terms that come *after* a bridge interval hypothesis: in an instantiation of $(\Gamma, \boldsymbol{x} : \mathbf{I}, \Delta)$, the terms supplied for Δ *can* reference a variable substituted for $\boldsymbol{x}$.

The open bridge interval judgments are defined as in the path case.

Definition 9.1.8 (Open interval judgments). The judgment $\Gamma \gg \boldsymbol{r} \in \mathbf{I}$ is defined to hold when $\boldsymbol{r} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{1}$, or $\boldsymbol{r} = \boldsymbol{x}$ for some $(\boldsymbol{x} : \mathbf{I}) \in \Gamma$. The equality judgment $\Gamma \gg \boldsymbol{r} = \boldsymbol{s} \in \mathbf{I}$ is defined to hold when $\Gamma \gg \boldsymbol{r}, \boldsymbol{s} \in \mathbf{I}$ and their equality is in the equivalence relation generated by the equational constraints occurring in Γ.

To define general substitutions, we must first extend the definition of interval restriction from interval to arbitrary contexts.

Definition 9.1.9 (Restriction for term contexts). If a bridge term $\boldsymbol{r}$ is equal to some constant, then restriction has no effect.

$$\Gamma \setminus \boldsymbol{r} \coloneqq \Gamma \quad \text{if } \Gamma \gg \boldsymbol{r} = \boldsymbol{\varepsilon} \in \mathbf{I} \text{ for some } \boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}$$

Otherwise, restriction is defined as follows.

$$\begin{aligned}(\Gamma, y : \mathbb{I}) \setminus \boldsymbol{x} &\coloneqq (\Gamma \setminus \boldsymbol{x}), y : \mathbb{I} \\ (\Gamma, \boldsymbol{y} : \mathbf{I}) \setminus \boldsymbol{x} &\coloneqq \begin{cases} \Gamma & \text{if } \boldsymbol{x} = \boldsymbol{y} \\ (\Gamma \setminus \boldsymbol{x}), \boldsymbol{y} : \mathbf{I} & \text{otherwise} \end{cases} \\ (\Gamma, \xi) \setminus \boldsymbol{x} &\coloneqq (\Gamma \setminus \boldsymbol{x}), \xi \\ (\Gamma, a : A) \setminus \boldsymbol{x} &\coloneqq \Gamma \setminus \boldsymbol{x}\end{aligned}$$

Restriction by a bridge variable $\boldsymbol{x}$ removes any term hypotheses that succeed $\boldsymbol{x}$, but not those that precede it. For example, we have $(a : A, \boldsymbol{x} : \mathbf{I}, b : B) \setminus \boldsymbol{x} = (a : A)$. It would not make sense to preserve the term hypotheses following $\boldsymbol{x}$, as their types may only make sense in the presence of $\boldsymbol{x}$; even putting dependency aside, they can be instantiated with terms that use $\boldsymbol{x}$, so they could be used to indirectly "smuggle in" an $\boldsymbol{x}$ if left alone. The hypotheses that precede $\boldsymbol{x}$, on the other hand, can never be instantiated with terms that use $\boldsymbol{x}$; this is ensured by the use of restriction in the definition of closing substitutions.

Definition 9.1.10 (Open substitutions). $\Gamma' \gg \gamma = \gamma' \in \Gamma$ is inductively generated by the the following rules.

$$\frac{}{\Gamma' \gg \cdot = \cdot \in \cdot} \qquad \frac{\Gamma' \setminus \boldsymbol{r} \gg \gamma = \gamma' \in \Gamma \qquad \Gamma' \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I}}{\Gamma' \gg (\gamma, \boldsymbol{r}/\boldsymbol{x}) = (\gamma', \boldsymbol{r}'/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{I})}$$

$$\frac{\Gamma' \gg r \in \mathbb{I} \qquad \Gamma' \gg \gamma = \gamma' \in \Gamma}{\Gamma' \gg (\gamma, r/x) = (\gamma', r/x) \in (\Gamma, x : \mathbb{I})} \qquad \frac{\Gamma' \gg \gamma = \gamma' \in \Gamma \qquad \Gamma' \gg \xi\gamma \text{ satisfied}}{\Gamma' \gg \gamma = \gamma' \in (\Gamma, \xi)}$$

$$\frac{\Gamma' \gg \gamma = \gamma' \in \Gamma \qquad \Gamma' \gg M = M' \in A\gamma}{\Gamma' \gg (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma, a : A)}$$

It is important that restriction have an action on substitutions, particularly closing substitutions. This is essential for our usual method of proving rules, where we show a given rule holds relative to an arbitrary *interval* context and infer that it must hold for arbitrary term contexts by instantiating pointwise. If a hypothesis involves a restriction, then we must apply a restricted instantiation at that hypothesis.

Lemma 9.1.11 (Action of restriction). Given a context Γ, substitution γ into Γ, and interval term $\boldsymbol{r}$ over Γ, we define *the action of restriction by* $\boldsymbol{r}$ *on* γ, written $(\gamma : \Gamma) \setminus \boldsymbol{r}$, as follows. If $\boldsymbol{r}$ is a constant, then restriction is the identity.

$$(\gamma : \Gamma) \setminus \boldsymbol{r} \coloneqq \gamma \quad \text{if } \Gamma \gg \boldsymbol{r} = \boldsymbol{\varepsilon} \in \mathbf{I} \text{ for some } \boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}$$

Otherwise, restriction is defined as follows.

$$
\begin{aligned}
(\Gamma, y : \mathbb{I}) \setminus \boldsymbol{x} &\coloneqq (\Gamma \setminus \boldsymbol{x}), y : \mathbb{I} \\
(\Gamma, \boldsymbol{y} : \mathbf{I}) \setminus \boldsymbol{x} &\coloneqq \begin{cases} \Gamma & \text{if } \boldsymbol{x} = \boldsymbol{y} \\ (\Gamma \setminus \boldsymbol{x}), \boldsymbol{y} : \mathbf{I} & \text{otherwise} \end{cases} \\
(\Gamma, \xi) \setminus \boldsymbol{x} &\coloneqq \begin{cases} \Gamma \setminus \boldsymbol{x} & \text{if } \boldsymbol{x} \text{ occurs in } \xi \\ (\Gamma \setminus \boldsymbol{x}), \xi & \text{otherwise} \end{cases} \\
(\Gamma, a : A) \setminus \boldsymbol{x} &\coloneqq \Gamma \setminus \boldsymbol{x}
\end{aligned}
$$

Given contexts $\Gamma = \Gamma'$ ctx, substitutions $\Gamma'' \gg \gamma = \gamma' \in \Gamma$, and terms $\Gamma \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I}$, we have that $\Gamma'' \setminus \boldsymbol{r}\gamma \gg (\gamma : \Gamma) \setminus \boldsymbol{r} = (\gamma' : \Gamma') \setminus \boldsymbol{r}' \in \Gamma \setminus \boldsymbol{r}$.

Proof. If $\boldsymbol{r}$ is equal to a constant, then this is immediate. Otherwise, we go by induction on the derivation of $\Gamma'' \gg \gamma = \gamma' \in \Gamma$.

- Case: $\Gamma' \gg \cdot = \cdot \in \cdot$. Immediate.
- Case: $\Gamma'' \gg (\gamma, \boldsymbol{s}/\boldsymbol{y}) = (\gamma', \boldsymbol{s}'/\boldsymbol{y}) \in (\Gamma, \boldsymbol{y} : \mathbf{I})$. If $\boldsymbol{r} = \boldsymbol{y}$, then we have $\Gamma'' \setminus \boldsymbol{s} \gg \gamma = \gamma' \in \Gamma$ by assumptions of this rule, which is exactly what we need. If not, then we instead have the substitutions $\Gamma'' \setminus \boldsymbol{s} \setminus \boldsymbol{r}\gamma \gg (\gamma : \Gamma) \setminus \boldsymbol{r} = (\gamma' : \Gamma') \setminus \boldsymbol{r}' \in \Gamma \setminus \boldsymbol{r}$ by induction hypothesis, to which we append $\Gamma'' \setminus \boldsymbol{r}\gamma \gg \boldsymbol{s} \in \mathbf{I}$ using the fact that $\Gamma'' \setminus \boldsymbol{s} \setminus \boldsymbol{r}\gamma = \Gamma'' \setminus \boldsymbol{r}\gamma \setminus \boldsymbol{s}$.
- Case: $\Gamma' \gg (\gamma, r/x) = (\gamma', r/x) \in (\Gamma, x : \mathbb{I})$. By induction hypothesis and the substitution formation rule for path dimensions.
- Case: $\Gamma' \gg \gamma = \gamma' \in (\Gamma, \xi)$. By induction hypothesis and the substitution formation rule for constraints.
- Case: $\Gamma' \gg (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma, a : A)$. Immediate by induction hypothesis. □

Remark 9.1.12. On the level of syntax, the effect of a restricted substitution is the same as that of the original substitution. That is, if M is a term depending only on the variables in $\Gamma \setminus r$, then $M[(\gamma : \Gamma) \setminus \boldsymbol{r}] = M\gamma$.

Now we get down to specifics. The new operational semantics rules we use for parametric type theory are shown in Figure 9.1. We construct our type systems in the usual way, taking the least fixed-point of an operator that introduces one layer of each type former. Below we get a sneak peak at the key type formers of parametric type theory, the bridge and Gel types, which we introduce in more detail below.

Example 9.1.13 (Small type system). We define an operator IP on candidate type systems as follows: given τ, $IP(\tau)$ is the union of the following clauses.

- $IP(\tau) \vDash \Psi \Vdash \mathsf{Bridge}(\boldsymbol{x}.A, M_0, M_1) \approx \mathsf{Bridge}(\boldsymbol{x}.A, M_0', M_1') \downarrow R$ whenever
  - $A \approx A' \in \Downarrow\tau[S]$ for some $(\Psi, \boldsymbol{x} : \mathbf{I})$-PER S,
  - $M_\varepsilon \approx M_\varepsilon' \in \Downarrow S[\varepsilon/x]$ for $\boldsymbol{\varepsilon} \in \{0, 1\}$,
  - $V \approx V' \in R\langle\psi\rangle$ holds for $\Psi' \Vdash \psi \in \Psi$ exactly when $V = \lambda^{\mathbf{I}} x.\, M$ and $V' = \lambda^{\mathbf{I}} x.\, M'$ for some M, M' with $M \approx M' \in \Downarrow S\psi$ and $M[\boldsymbol{\varepsilon}/\boldsymbol{x}] \approx M_\varepsilon \psi \in \Downarrow S\psi[\boldsymbol{\varepsilon}/\boldsymbol{x}]$ for $\varepsilon \in \{0, 1\}$.
- $IP(\tau) \vDash \Psi \Vdash \mathsf{Gel}_{\boldsymbol{r}}(A, B, a.b.R) \approx \mathsf{Gel}_{\boldsymbol{r}}(A', B', a.b.R') \downarrow S$ whenever
  - $\tau \vDash \Psi \Vdash \boldsymbol{r} \in \mathbf{I}$,
  - $A \approx A' \in \Downarrow\tau[S]$ for some $(\Psi \setminus \boldsymbol{r})$-PER S,
  - $B \approx B' \in \Downarrow\tau[T]$ for some $(\Psi \setminus \boldsymbol{r})$-PER T,
  - $R\psi[M/a, N/b] \approx R'\psi[M'/a, N'/b] \in \Downarrow\tau[U_{M,N}]$ for all $\Psi' \Vdash \psi \in (\Psi \setminus \boldsymbol{r})$, $M \approx M' \in \Downarrow S\psi$, and $N \approx N' \in \Downarrow T\psi$, for some family of PERs $U_{M,N}$ respecting equality in $\Downarrow S$ and $\Downarrow T$,
  - $V \approx V' \in S\langle\psi\rangle$ holds for $\Psi' \Vdash \psi \in \Psi$ exactly when one of the following holds:
    - $\boldsymbol{r}\psi = \mathbf{0}$, and $V \approx V' \in S\psi$,
    - $\boldsymbol{r}\psi = \mathbf{1}$, and $V \approx V' \in T\psi$,
    - $\boldsymbol{r}\psi = \boldsymbol{x}$, and $V = \mathsf{gel}_{\boldsymbol{x}}(M, N, P)$ and $V' = \mathsf{gel}_{\boldsymbol{x}}(M', N', P')$ with $M \approx M' \in \Downarrow S\psi$, $N \approx N' \in \Downarrow T\psi$, and $P \approx P' \in \Downarrow U_{M,N}\psi$.

We define the candidate type system τ_0^{IP} to be the least fixed point of $F \cup H \cup IP$, where F is as defined in Example 3.1.32 and H is as defined in Example 6.2.22; we may omit H if we have no interest in interpreting higher inductive types.

As in Examples 3.1.33 and 6.2.23, we may construct a type system for internally parametric type theory with a universe by taking the least fixed point of $F \cup H \cup IP \cup U(\tau_0^{IP})$, where U is as defined in Example 3.1.33.

9.2 Bridge types

The first type former we need for parametric type theory is the internalization of bridge interval abstraction: the bridge type. We think of an element of $\mathsf{Bridge}(\boldsymbol{x}.A, M_0, M_1)$ as a proof that M_0 and M_1 are related across the relation $\boldsymbol{x}.A$.

We display the standard collection of rules for bridge types in Figure 9.2. Like paths, bridges are formed by abstraction and used by application, and they satisfy familiar reduction, boundary, and uniqueness equations. The only distinction is the addition of the interval restriction $\setminus \boldsymbol{r}$ in the application rule, which forbids us from instantiating a

Bridges

$$\frac{}{\mathsf{Bridge}(\boldsymbol{x}.A, M, N)\ \mathsf{val}} \qquad \frac{}{\lambda^{\mathbf{I}}\boldsymbol{x}.\, M\ \mathsf{val}} \qquad \frac{P \longmapsto P'}{P\,\boldsymbol{r} \longmapsto P'\,\boldsymbol{r}} \qquad \frac{}{(\lambda^{\mathbf{I}}\boldsymbol{x}.\, M)\,\boldsymbol{r} \longmapsto P[\boldsymbol{r}/\boldsymbol{x}]}$$

$$\frac{}{\mathsf{coe}^{r\to s}_{x.\mathsf{Bridge}(\boldsymbol{y}.A,M_0,M_1)}(P) \longmapsto \lambda^{\mathbf{I}}\boldsymbol{y}.\, \mathsf{com}^{r\to s}_{x.A}(P; \boldsymbol{y} \equiv \mathbf{0} \hookrightarrow x.M_0, \boldsymbol{y} \equiv \mathbf{1} \hookrightarrow x.M_1)}$$

$$\frac{}{\begin{array}{c}\mathsf{hcom}^{r\to s}_{\mathsf{Bridge}(\boldsymbol{y}.A,M_0,M_1)}(P; \overrightarrow{\xi_i \hookrightarrow x.Q_i}) \\ \longmapsto \\ \lambda^{\mathbf{I}}\boldsymbol{y}.\, \mathsf{hcom}^{r\to s}_{A}(P\,\boldsymbol{y}; \overrightarrow{\xi_i \hookrightarrow x.Q_i\,\boldsymbol{y}}, \boldsymbol{y} \equiv \mathbf{0} \hookrightarrow \_.M_0, \boldsymbol{y} \equiv \mathbf{1} \hookrightarrow \_.M_1)\end{array}}$$

Gel ***types***

$$\frac{}{\mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, a.b.R)\ \mathsf{val}} \qquad \frac{}{\mathsf{Gel}_{\boldsymbol{\varepsilon}}(A_0, A_1, a.b.R) \longmapsto A_\varepsilon}$$

$$\frac{}{\mathsf{gel}_{\boldsymbol{x}}(M_0, M_1, P)\ \mathsf{val}} \qquad \frac{}{\mathsf{gel}_{\boldsymbol{\varepsilon}}(M_0, M_1, P) \longmapsto M_\varepsilon}$$

$$\frac{Q \longmapsto Q'}{\mathsf{ungel}(\boldsymbol{x}.Q) \longmapsto \mathsf{ungel}(\boldsymbol{x}.Q')} \qquad \frac{\boldsymbol{x} \notin P}{\mathsf{ungel}(\boldsymbol{x}.\mathsf{gel}_{\boldsymbol{x}}(M_0, M_1, P)) \longmapsto P}$$

$$\frac{\begin{array}{c}M^y_\varepsilon \coloneqq \mathsf{hcom}^{r\to y}_{A_\varepsilon}(Q[\boldsymbol{\varepsilon}/\boldsymbol{x}]; \overrightarrow{\xi_i[\boldsymbol{\varepsilon}/\boldsymbol{x}] \hookrightarrow y.Q_i[\boldsymbol{\varepsilon}/\boldsymbol{x}]}) \\ P \coloneqq \mathsf{com}^{r\to s}_{y.R[M^y_0/a_0, M^y_1/a_1]}(\mathsf{ungel}(\boldsymbol{x}.Q); (\overrightarrow{\xi_i \hookrightarrow y.\mathsf{ungel}(\boldsymbol{x}.Q_i)})_{\boldsymbol{x}\notin\xi_i})\end{array}}{\mathsf{hcom}^{r\to s}_{\mathsf{Gel}_{\boldsymbol{x}}(A_0,A_1,a_0.a_1.R)}(Q; \overrightarrow{\xi_i \hookrightarrow y.Q_i}) \longmapsto \mathsf{gel}_{\boldsymbol{x}}(M^s_0, M^s_1, P)}$$

$$\frac{M^y_\varepsilon \coloneqq \mathsf{coe}^{r\to y}_{y.A_\varepsilon}(Q[\boldsymbol{\varepsilon}/\boldsymbol{x}]) \qquad P \coloneqq \mathsf{coe}^{r\to s}_{y.R[M^y_0/a_0, M^y_1/a_1]}(\mathsf{ungel}(\boldsymbol{x}.Q))}{\mathsf{coe}^{r\to s}_{y.\mathsf{Gel}_{\boldsymbol{x}}(A_0,A_1,a_0.a_1.R)}(Q) \longmapsto \mathsf{gel}_{\boldsymbol{x}}(M^s_0, M^s_1, P)}$$

The extent ***operator***

$$\frac{}{\mathsf{extent}_{\boldsymbol{\varepsilon}}(M; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N}) \longmapsto N_\varepsilon[M/a]}$$

$$\frac{}{\mathsf{extent}_{\boldsymbol{x}}(M; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N}) \longmapsto \overline{N}[M[\mathbf{0}/\boldsymbol{x}]/a_0, M[\mathbf{1}/\boldsymbol{x}]/a_1, \lambda^{\mathbf{I}}\boldsymbol{x}.\, M/\overline{a}]\,\boldsymbol{x}}$$

Figure 9.1: Additional operational semantics for parametric type theory

$$\frac{\Psi, \boldsymbol{x} : \mathbf{I} \Vdash A = A' \text{ type} \qquad \Psi \Vdash M_0 = M_0' \in A[\mathbf{0}/\boldsymbol{x}] \qquad \Psi \Vdash M_1 = M_1' \in A[\mathbf{1}/\boldsymbol{x}]}{\Psi \Vdash \mathsf{Bridge}(\boldsymbol{x}.A, M_0, M_1) = \mathsf{Bridge}(\boldsymbol{x}.A', M_0', M_1') \text{ type}}$$

$$\frac{\Psi, \boldsymbol{x} : \mathbf{I} \Vdash A \text{ type} \qquad \Psi, \boldsymbol{x} : \mathbf{I} \Vdash M = M' \in A}{\Psi \Vdash \lambda^{\mathbf{I}} a.\, M = \lambda^{\mathbf{I}} a.\, M' \in \mathsf{Bridge}(\boldsymbol{x}.A, M[\mathbf{0}/\boldsymbol{x}], M[\mathbf{1}/\boldsymbol{x}])}$$

$$\frac{\begin{array}{c}\Psi, \boldsymbol{x} : \mathbf{I} \Vdash A \text{ type} \qquad (\forall \varepsilon)\ \Psi \Vdash M_\varepsilon \in A[\boldsymbol{\varepsilon}/\boldsymbol{x}] \\ \Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad \Psi \setminus \boldsymbol{r} \Vdash P = P' \in \mathsf{Bridge}(\boldsymbol{x}.A, M_0, M_1)\end{array}}{\Psi \Vdash P\, \boldsymbol{r} = P'\, \boldsymbol{r} \in A[\boldsymbol{r}/\boldsymbol{x}]}$$

$$\frac{\Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash A \text{ type} \qquad \Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash M \in A}{\Psi \Vdash (\lambda^{\mathbf{I}} \boldsymbol{x}.\, M)\, \boldsymbol{r} = M[\boldsymbol{r}/\boldsymbol{x}] \in A[\boldsymbol{r}/\boldsymbol{x}]}$$

$$\frac{\begin{array}{c}\Psi, \boldsymbol{x} : \mathbf{I} \Vdash A \text{ type} \qquad (\forall \varepsilon)\ \Psi \Vdash M_\varepsilon \in A[\boldsymbol{\varepsilon}/\boldsymbol{x}] \\ \Psi \Vdash P \in \mathsf{Bridge}(\boldsymbol{x}.A, M_0, M_1) \qquad \varepsilon \in \{0, 1\}\end{array}}{\Psi \Vdash P\, \boldsymbol{\varepsilon} = M_\varepsilon \in A[\boldsymbol{\varepsilon}/\boldsymbol{x}]}$$

$$\frac{\Psi, \boldsymbol{x} : \mathbf{I} \Vdash A \text{ type} \qquad (\forall \varepsilon)\ \Psi \Vdash M_\varepsilon \in A[\boldsymbol{\varepsilon}/\boldsymbol{x}] \qquad \Psi \Vdash P \in \mathsf{Bridge}(\boldsymbol{x}.A, M_0, M_1)}{\Psi \Vdash P = \lambda^{\mathbf{I}} \boldsymbol{x}.\, P\, \boldsymbol{x} \in \mathsf{Bridge}(\boldsymbol{x}.A, M_0, M_1)}$$

Figure 9.2: Rules for bridge types

bridge P with a term $\boldsymbol{r}$ that already occurs in P. This matches the situation for judgmental bridges: given $\Psi, \boldsymbol{x} : \mathbf{I} \Vdash M \in A$, we can only instantiate $\boldsymbol{x}$ with some $\Psi \Vdash \boldsymbol{r} \in \mathbf{I}$ if M and A are actually well-typed in the sub-context $(\Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I}) \subseteq (\Psi, \boldsymbol{x} : \mathbf{I})$, in which case we can apply the substitution $\Psi \Vdash (\mathsf{id}_{\Psi \setminus r}, \boldsymbol{r}/\boldsymbol{x}) \in (\Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I})$ to get $\Psi \Vdash M[\boldsymbol{r}/\boldsymbol{x}] \in A[\boldsymbol{r}/\boldsymbol{x}]$.

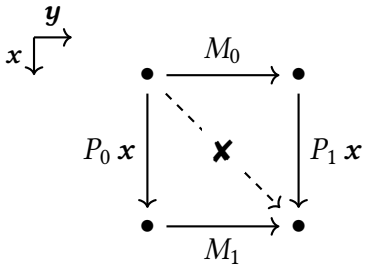
As the proofs of these rules do not deviate noticeably from those for path types (Section 3.1.6.1), we leave them as an exercise to the reader; the Kan operations, too, are the same as for paths, though now relying on the presence of $\boldsymbol{r} \equiv \boldsymbol{\varepsilon}$ constraints. (Full proofs may be found in [CH19b, §5].) However, it is worth observing explicitly that, even with the complication of restriction in hypotheses, we can still derive open rules from their closed form, as in the example below. As mentioned above, the key fact is that interval restriction has an action on closing substitutions.

Rule 9.2.1 (Open bridge reduction). Let Γ ctx.

$$\frac{\Gamma, \boldsymbol{x} : \mathbf{I} \gg A \text{ type} \qquad \Gamma \gg \boldsymbol{r} \in \mathbf{I} \qquad \Gamma \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \gg M \in A}{\Gamma \gg (\lambda^{\mathbf{I}} \boldsymbol{x}.\, M)\, \boldsymbol{r} = M[\boldsymbol{r}/\boldsymbol{x}] \in A[\boldsymbol{r}/\boldsymbol{x}]}$$

Proof. By definition of the open typing judgment, we must show for every $\Psi \Vdash \gamma = \gamma' \in \Gamma$ that $\Psi \Vdash ((\lambda^{\mathbf{I}} \boldsymbol{x}.\, M)\, \boldsymbol{r})\gamma = M[\boldsymbol{r}/\boldsymbol{x}]\gamma' \in A[\boldsymbol{r}/\boldsymbol{x}]\gamma$. By instantiating $\Gamma, \boldsymbol{x} : \mathbf{I} \gg A$ type with $(\gamma, \boldsymbol{x}/\boldsymbol{x})$, we have $\Psi, \boldsymbol{x} : \mathbf{I} \Vdash A\gamma$ type; by instantiating $\Gamma \gg \boldsymbol{r} \in \mathbf{I}$ with γ, we have $\Psi \Vdash \boldsymbol{r}\gamma \in \mathbf{I}$; by instantiating $\Gamma \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \gg M \in A$ with $\Psi \setminus \boldsymbol{r}\psi \Vdash (\gamma : \Gamma) \setminus \boldsymbol{r}$, we have $\Psi \setminus \boldsymbol{r}\gamma, \boldsymbol{x} : \mathbf{I} \Vdash M\gamma \in A\gamma$. We thus obtain $\Psi \Vdash ((\lambda^{\mathbf{I}} \boldsymbol{x}.\, M)\, \boldsymbol{r})\gamma = M[\boldsymbol{r}/\boldsymbol{x}]\gamma' \in A[\boldsymbol{r}/\boldsymbol{x}]\gamma$ by applying the closed rule. □

To give a concrete consequence of affinity, we cannot take the diagonal of a two-dimensional bridge. That is, given a term $Q \in \mathsf{Bridge}(\boldsymbol{y}.\mathsf{Bridge}(A, M_0, M_1), P_0, P_1)$, we cannot write the term "$\lambda^{\mathbf{I}} \boldsymbol{x}.\, Q\, \boldsymbol{x}\, \boldsymbol{x} \in \mathsf{Bridge}(A, P_0\, \mathbf{0}, P_1\, \mathbf{1})$", the diagonal of the square shown below.



Indeed, the term $Q\, \boldsymbol{x}$ already mentions $\boldsymbol{x}$, so cannot be applied to $\boldsymbol{x}$ a second time.

Note, however, that nothing prevents a bridge variable from occurring multiple times in a term in general. We see an example in the proof of the following lemma, which is a carbon copy of Lemma 3.2.4.

Lemma 9.2.2 (Bridges in products). Let $\boldsymbol{x} : \mathbf{I} \gg A$ type and $\boldsymbol{x} : \mathbf{I}, a : A \gg B$ type be given together with $T_0 \in ((a : A) \times B)[\mathbf{0}/\boldsymbol{x}]$ and $T_1 \in ((a : A) \times B)[\mathbf{1}/\boldsymbol{x}]$. Then we have an isomorphism of the following type.

$$\mathsf{Bridge}(\boldsymbol{x}.(a : A) \times B, T_0, T_1)$$
$$\simeq$$
$$(p : \mathsf{Bridge}(\boldsymbol{x}.A, \mathsf{fst}(T_0), \mathsf{fst}(T_1))) \times \mathsf{Bridge}(\boldsymbol{x}.B[p\, \boldsymbol{x}/a], \mathsf{snd}(T_0), \mathsf{snd}(T_1))$$

Proof. In the forward direction, given $t : \mathsf{Bridge}(\boldsymbol{x}.(a : A) \times B, T_0, T_1)$, we have the pair of bridges $\langle \lambda^{\mathbf{I}} \boldsymbol{x}.\, \mathsf{fst}(t\, \boldsymbol{x}), \lambda^{\mathbf{I}} \boldsymbol{x}.\, \mathsf{snd}(t\, \boldsymbol{x}) \rangle$. In the reverse, given a pair of bridges across the two types, $p : \mathsf{Bridge}(\boldsymbol{x}.A, \mathsf{fst}(T_0), \mathsf{fst}(T_1))$ and $q : \mathsf{Bridge}(\boldsymbol{x}.B[p\, \boldsymbol{x}/a], \mathsf{snd}(T_0), \mathsf{snd}(T_1))$, we have a bridge in the product type $\lambda^{\mathbf{I}} \boldsymbol{x}.\, \langle p\, \boldsymbol{x}, q\, \boldsymbol{x} \rangle$. These constructions are inverse up to exact equality. □

In the term $\lambda^{\mathbf{I}} \boldsymbol{x}.\, \langle p\, \boldsymbol{x}, q\, \boldsymbol{x} \rangle$ above, we have used $\boldsymbol{x}$ in two places, but this is not a problem: the only requirement is that $\boldsymbol{x}$ be fresh for p and q individually. This is the case here

because p and q were introduced prior to $\boldsymbol{x}$: they precede $\boldsymbol{x}$ in the context, so are not deleted by $\setminus \boldsymbol{x}$.

As was the case for cubical equality, Lemma 9.2.2 is one of a laundry list of results we will be able to prove relating bridges in compound types to bridges in their component types. In particular, we can characterize bridges in path types: a path between bridges is the same as a bridge between paths.

Lemma 9.2.3 (Bridges in path types). Let $y : \mathbb{I}, \boldsymbol{x}{:}\mathbf{I} \gg A$ type, $\boldsymbol{x}{:}\mathbf{I} \gg M_0 \in A[0/y]$, and $\boldsymbol{x} : \mathbf{I} \gg M_0 \in A[0/y]$ be given together with $P_0 \in \mathsf{Path}(y.A[\mathbf{0}/\boldsymbol{x}], M_0[\mathbf{0}/\boldsymbol{x}], M_1[\mathbf{0}/\boldsymbol{x}])$ and $P_1 \in \mathsf{Path}(y.A[\mathbf{1}/\boldsymbol{x}], M_0[\mathbf{1}/\boldsymbol{x}], M_1[\mathbf{1}/\boldsymbol{x}])$. Then we have an isomorphism of the following type.

$$\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Path}(y.A, M_0, M_1), P_0, P_1)$$
$$\simeq$$
$$\mathsf{Path}(y.\mathsf{Bridge}(\boldsymbol{x}.A, P_0\, y, P_1\, y), \lambda^{\mathbf{I}}\boldsymbol{x}.\, M_0, \lambda^{\mathbf{I}}\boldsymbol{x}.\, M_1)$$

Proof. Like the function extensionality isomorphism for paths, this isomorphism simply swaps the order of binders. Given p of the former type, we have $\lambda^{\mathbb{I}} y.\, \lambda^{\mathbf{I}}\boldsymbol{x}.\, p\, \boldsymbol{x}\, y$ in the latter; given q in the former, we have $\lambda^{\mathbf{I}}\boldsymbol{x}.\, \lambda^{\mathbb{I}} y.\, q\, y\, \boldsymbol{x}$ in the latter. (We use here the fact that restriction ignores path interval variables.) These are evidently inverses up to exact equality. □

The above, read in reverse, doubles as a characterization of paths in bridge types. The type of bridges across many compound types can be characterized in the same way and with the same proof as the type of paths, as in the case of products. There are, however, key differences. As our next step, we consider function types, a case where the stories diverge.

9.3 Function types and the extent operator

In Section 3.2, we proved two results characterizing the behavior of paths at function type. First, we had *function extensionality* (Lemma 3.2.5), a practically trivial result characterizing the type $\mathsf{Path}(x.(a : A) \to B, F_0, F_1)$ when A does not depend on x. Second, we gave a more general characterization for the case where A *does* depend on x, the proof of which relied on the existence of coercion for paths (Lemma 3.2.6).

For bridges, we again want the more general characterization, in accordance with the standard definition of relation at function type used in classical parametricity and logical relations more generally: two functions are related when they take related arguments to

related results.

$$\mathsf{Bridge}(\boldsymbol{x}.(a : A) \to B, F_0, F_1)$$
$$\stackrel{?}{\simeq}$$
$$(a_0 : A[\mathbf{0}/\boldsymbol{x}])\ (a_1 : A[\mathbf{1}/\boldsymbol{x}])\ (p : \mathsf{Bridge}(\boldsymbol{x}.A, a_0, a_1)) \to \mathsf{Bridge}(\boldsymbol{x}.B[p\,\boldsymbol{x}/a], F_0\,a_0, F_1\,a_1)$$

We cannot, however, simply repeat our proof of paths: we have no coercion across bridges. Instead, we will rely here for the first time on the affinity of bridge variables.

We can easily implement the forward direction as in the proof of Lemma 3.2.6: given $q : \mathsf{Bridge}(\boldsymbol{x}.(a : A) \to B, F_0, F_1)$, we define the function $\lambda a_0.\, \lambda a_1.\, \lambda p.\, \lambda^{\mathbf{I}}\boldsymbol{x}.\, (q\,\boldsymbol{x})\ (p\,\boldsymbol{x})$ from bridges in the domain to bridges in the codomain. The difficulty, then, is in the converse. Suppose we are given a function h of the right hand type above. We need to transform this into a path of functions: given $\boldsymbol{x} : \mathbf{I}$ and then $a : A$, we must produce an element of B that is $F_0\,a$ when $\boldsymbol{x} = \mathbf{0}$ and $F_1\,a$ when $\boldsymbol{x} = \mathbf{1}$. In the proof of Lemma 3.2.6, we used coercion to create a path from a and applied h, but we cannot do this now.

Consider the situation where $a : A$ has been instantiated with some closed term M. Because M is introduced after $\boldsymbol{x}$, it might use $\boldsymbol{x}$; indeed, we can think of it as a *function of* $\boldsymbol{x}$. If we could *abstract* the variable $\boldsymbol{x}$ in M, writing $\lambda^{\mathbf{I}}\boldsymbol{x}.\, M$, we would have a bridge over A, and could take $h\,(M[\mathbf{0}/\boldsymbol{x}])\ (M[\mathbf{1}/\boldsymbol{x}])\ (\lambda^{\mathbf{I}}\boldsymbol{x}.\, M)\,\boldsymbol{x}$ as our result. Of course, we cannot literally do so with a: the term $\lambda^{\mathbf{I}}\boldsymbol{x}.\, a$ is a constant bridge, as a does not mention $\boldsymbol{x}$. As our operational semantics is defined on closed terms, however, we can instead define an auxiliary operator that performs interval abstraction on such terms.

For this purpose, we introduce the extent operator to the operational semantics, as shown in Figure 9.1 and replicated below. We call this operator "extent" because it reveals *the extent of a term* (M below) in a given *direction*, a bridge interval term $\boldsymbol{r}$. If $\boldsymbol{r}$ is a constant, then M is simply a point; if $\boldsymbol{r}$ is a variable $\boldsymbol{x}$, then M is *one point on a bridge*, namely the point at $\boldsymbol{x}$ of the bridge $\lambda^{\mathbf{I}}x.\, M$.

$$\frac{}{\mathsf{extent}_{\boldsymbol{\varepsilon}}(M; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N}) \longmapsto N_\varepsilon[M/a]}$$

$$\frac{}{\mathsf{extent}_{\boldsymbol{x}}(M; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N}) \longmapsto \overline{N}[M[\mathbf{0}/\boldsymbol{x}]/a_0, M[\mathbf{1}/\boldsymbol{x}]/a_1, \lambda^{\mathbf{I}}\boldsymbol{x}.\, M/\overline{a}]\,\boldsymbol{x}}$$

Like an eliminator for an inductive type, extent takes a case branch term for each possible value of $\boldsymbol{r}$, the terms N_0, N_1, and $\overline{N}$ above. If $\boldsymbol{r}$ is an endpoint constant, we pass M—which is just a point—to the corresponding case, per the first reduction rule above. If $\boldsymbol{r}$ is a variable $\boldsymbol{x}$, then we pass the bridge $\lambda^{\mathbf{I}}x.\, M$ (and its two endpoints) to the variable case.

The extent operator satisfies the following principles. We have a typing rule as well as reductions for the constant and variable cases.

Rules 9.3.1 (Extent). We present the first rule in unary form for lack of space, but extent does preserve exact equality in each argument.

(1)
$$\frac{\begin{array}{c}\Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash A \text{ type} \qquad \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I}, a : A \gg B \text{ type} \\ \Psi \Vdash M \in A[\boldsymbol{r}/\boldsymbol{x}] \qquad (\forall \varepsilon)\ \Psi \setminus \boldsymbol{r}, a_\varepsilon : A[\boldsymbol{\varepsilon}/\boldsymbol{x}] \gg N_\varepsilon \in B[\boldsymbol{\varepsilon}/\boldsymbol{x}, a_\varepsilon/a] \\ \Psi \setminus \boldsymbol{r}, a_0 : A[\mathbf{0}/\boldsymbol{x}], a_1 : A[\mathbf{1}/\boldsymbol{x}], \overline{a} : \mathsf{Bridge}(\boldsymbol{x}.A, a_0, a_1) \gg \overline{N} \in \mathsf{Bridge}(\boldsymbol{x}.B[\overline{a}\,\boldsymbol{x}/a], N_0, N_1)\end{array}}{\Psi \Vdash \mathsf{extent}_{\boldsymbol{r}}(M; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N}) \in B[\boldsymbol{r}/\boldsymbol{x}, M/a]}$$

(2)
$$\frac{\begin{array}{c}\varepsilon \in \{0,1\} \qquad \Psi, \boldsymbol{x} : \mathbf{I} \Vdash A \text{ type} \qquad \Psi, \boldsymbol{x} : \mathbf{I}, a : A \gg B \text{ type} \\ \Psi \Vdash M \in A[\boldsymbol{\varepsilon}/\boldsymbol{x}] \qquad (\forall \varepsilon)\ \Psi \setminus \boldsymbol{r}, a_\varepsilon : A[\boldsymbol{\varepsilon}/\boldsymbol{x}] \gg N_\varepsilon \in B[\boldsymbol{\varepsilon}/\boldsymbol{x}, a_\varepsilon/a]\end{array}}{\Psi \Vdash \mathsf{extent}_{\boldsymbol{\varepsilon}}(M; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N}) = N_\varepsilon[M/a_\varepsilon] \in B[\boldsymbol{\varepsilon}/\boldsymbol{x}, M/a]}$$

(3)
$$\frac{\begin{array}{c}\Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash A \text{ type} \qquad \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I}, a : A \gg B \text{ type} \\ \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash M \in A \qquad (\forall \varepsilon)\ \Psi \setminus \boldsymbol{r}, a_\varepsilon : A[\boldsymbol{\varepsilon}/\boldsymbol{x}] \gg N_\varepsilon \in B[\boldsymbol{\varepsilon}/\boldsymbol{x}, a_\varepsilon/a] \\ \Psi \setminus \boldsymbol{r}, a_0 : A[\mathbf{0}/\boldsymbol{x}], a_1 : A[\mathbf{1}/\boldsymbol{x}], \overline{a} : \mathsf{Bridge}(\boldsymbol{x}.A, a_0, a_1) \gg \overline{N} \in \mathsf{Bridge}(\boldsymbol{x}.B[\overline{a}\,\boldsymbol{x}/a], N_0, N_1) \\ O \coloneqq \overline{N}[M[\mathbf{0}/\boldsymbol{x}]/a_0, M[\mathbf{1}/\boldsymbol{x}]/a_1, \lambda^{\mathbf{I}}\boldsymbol{x}.\, M/\overline{a}]\, \boldsymbol{r}\end{array}}{\Psi \Vdash \mathsf{extent}_{\boldsymbol{r}}(M[\boldsymbol{r}/\boldsymbol{x}]; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N}) = O \in B[M/a]\,[\boldsymbol{r}/\boldsymbol{x}]}$$

We will prove these momentarily; first, though, we see that we can use extent to define the bridge of functions induced by h as follows. Indeed, extent is precisely what we need.

$$\lambda^{\mathbf{I}}\boldsymbol{x}.\, \lambda a.\, \mathsf{extent}_{\boldsymbol{x}}(a; a_0.(F_0\, a_0), a_1.(F_1\, a_1), a_0.a_1.p.(h\, a_0\, a_1\, p))$$

Proof (of Rules 9.3.1). As usual, we prove the reduction rules first.

(2) Immediate by coherent head expansion.

(3) By coherent head expansion. It is easy to check that O is well-typed in $B[M/a]\,[\boldsymbol{r}/\boldsymbol{x}]$. Let $\Psi' \Vdash \psi \in \Psi$ be given; we are in one of two cases.

- $\boldsymbol{y}\psi = \boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}$. Then $\mathsf{extent}_{\boldsymbol{y}}(M[\boldsymbol{y}/\boldsymbol{x}]; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N})\psi$ reduces to the term $N_\varepsilon[M[\boldsymbol{y}/\boldsymbol{x}]/a_\varepsilon]\psi$. By the boundary rule for bridges, the latter is equal to $O\psi$ in $B[M/a]\,[\boldsymbol{y}/\boldsymbol{x}]\psi$.
- $\boldsymbol{r}\psi = \boldsymbol{y}$ for some variable $\boldsymbol{y}$. Then $\mathsf{extent}_{\boldsymbol{r}}(M[\boldsymbol{r}/\boldsymbol{x}]; a_0.N_0, a_1.N_1, a_0.a_1.\overline{a}.\overline{N})\psi$ reduces to the following term.

$$\overline{N}\psi[M[\boldsymbol{r}/\boldsymbol{x}]\psi[\mathbf{0}/\boldsymbol{y}]/a_0, M[\boldsymbol{r}/\boldsymbol{x}]\psi[\mathbf{1}/\boldsymbol{y}]/a_1, \lambda^{\mathbf{I}}\boldsymbol{y}.\, M[\boldsymbol{r}/\boldsymbol{x}]\psi/\overline{a}]\, \boldsymbol{y}$$

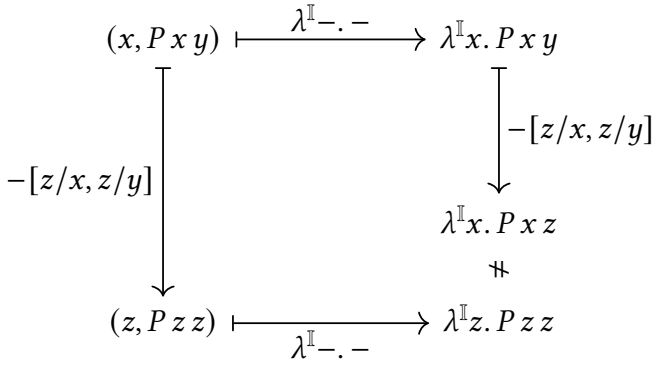
Note that $\boldsymbol{r}$ must itself be a variable in this case. We have $M[\boldsymbol{r}/\boldsymbol{x}]\psi = M\psi[\boldsymbol{y}/\boldsymbol{x}]$, so the above term is equal to the following.

$$\overline{N}\psi[M\psi[\mathbf{0}/\boldsymbol{x}]/a_0, M\psi[\mathbf{1}/\boldsymbol{x}]/a_1, \lambda^{\mathbf{I}}\boldsymbol{y}.\, M\psi[\boldsymbol{y}/\boldsymbol{x}]/\overline{a}]\ \boldsymbol{y}$$

Finally, we know that $\boldsymbol{r}$ does not occur in M by typing assumption. As ψ is affine, it can send no variables but $\boldsymbol{r}$ to $\boldsymbol{y}$, so $\boldsymbol{y}$ does not occur in $M\psi$. It follows that $\lambda^{\mathbf{I}}\boldsymbol{y}.\, M\psi[\boldsymbol{y}/\boldsymbol{x}] = \lambda^{\mathbf{I}}\boldsymbol{x}.\, M\psi$. The term above is therefore syntactically equal to $O\psi$.

(1) By cases on $\Psi \Vdash \boldsymbol{r} \in \mathbf{I}$. If $\boldsymbol{r}$ is a constant, then this follows from the constant reduction rule. If $\boldsymbol{r}$ is a variable, then it follows from the variable reduction rule. □

In the proof above we see the crucial role of affinity in the well-behavedness of extent: it delivers the equation $\lambda^{\mathbf{I}}\boldsymbol{y}.\, M\psi[\boldsymbol{y}/\boldsymbol{x}] = \lambda^{\mathbf{I}}\boldsymbol{x}.\, M\psi$. Intuitively, interval abstraction is stable under affine substitution, but not all structural substitutions. Say, for example, that we have some two-dimensional path term P applied to path interval variables x and y. By interleaving abstraction by x with the diagonal substitution $z : \mathbb{I} \Vdash (z/x, z/y) \in (x : \mathbb{I}, y : \mathbb{I})$ in different orders, we get different results.



This instability is familiar to any programmer who has had to implement capture-avoiding substitution. With affine variables, on the other hand, this situation cannot occur.

Theorem 9.3.2 (Bridges in function types). Let $\boldsymbol{x} : \mathbf{I} \gg A$ type and $\boldsymbol{x} : \mathbf{I}, a : A \gg B$ type be given together with $F_0 \in ((a : A) \to B)[\mathbf{0}/\boldsymbol{x}]$ and $F_1 \in ((a : A) \to B)[\mathbf{1}/\boldsymbol{x}]$. Then we have an isomorphism of the following type.

$$\mathsf{Bridge}(\boldsymbol{x}.(a : A) \to B, F_0, F_1)$$
$$\simeq$$
$$(a_0 : A[\mathbf{0}/\boldsymbol{x}])\ (a_1 : A[\mathbf{1}/\boldsymbol{x}])\ (p : \mathsf{Bridge}(\boldsymbol{x}.A, a_0, a_1)) \to \mathsf{Bridge}(\boldsymbol{x}.B[p\,\boldsymbol{x}/a], F_0\, a_0, F_1\, a_1)$$

That is, a bridge in a function type is a function from bridges in the domain to bridges in the codomain.

Proof. We have the functions in either direction defined above. One of inverse condition is the reduction rule for extent; the other may proven by applying extent. □

It is worth noting that, while affine variables allow us to prove this characterization without using coercion, they also *prevent* us from proving a function extensionality principle for bridges à la Lemma 3.2.5. That is, even when A does not depend on $\boldsymbol{x}$, the following does not hold in general.

$$\mathsf{Bridge}(\boldsymbol{x}.(a : A) \to B, F_0, F_1) \simeq (a : A) \to \mathsf{Bridge}(\boldsymbol{x}.B, F_0\, a, F_1\, a) \quad \text{✘}$$

To see why the proof of Lemma 3.2.5 does not apply, suppose we are given a function $h : (a : A) \to \mathsf{Bridge}(\boldsymbol{x}.B, F_0\, a, F_1\, a)$. We would like to write the following.

$$\lambda^{\mathbf{I}}\boldsymbol{x}.\, \lambda a.\, h\, a\, \boldsymbol{x} \in \mathsf{Bridge}(\boldsymbol{x}.(a : A) \to B, F_0, F_1) \quad \text{✘}$$

But this term is not in fact well-typed. We cannot apply $h\, a$ to $\boldsymbol{x}$, because a is not apart from $\boldsymbol{x}$: it was introduced after $\boldsymbol{x}$, so can be instantiated with terms that contain $\boldsymbol{x}$. Put another way, we have $(\boldsymbol{x} : \mathbf{I}, a : A) \setminus \boldsymbol{x} = \cdot$, so $h\, a$ is not well-typed in $(\boldsymbol{x} : \mathbf{I}, a : A) \setminus \boldsymbol{x}$.

To note one more point in the space of possibility, the BCH cubical sets model of homotopy type theory is based on affine cubical sets, like parametric type theory, but includes a coercion operation, like our structural cubical type theory. In this setting, we can obtain the equivalent of Theorem 9.3.2 by the extent argument. There, however, coercion can then be used to show derive function extensionality as well. (Indeed, function extensionality is a formal consequence of univalence [Uni13, §4.9], so it must hold in the BCH model.)

9.4 Gel types and relativity

As in cubical type theory, the coup de grâce of parametric type theory is a characterization of bridges in the universe. For cubical type theory, this is the univalence axiom (Theorem 3.2.9), which identifies paths in the universe with isomorphisms. More precisely, univalence states that the canonical function from paths to isomorphisms, implemented by coercion as shown below, is invertible.

$$p : \mathsf{Path}(\mathsf{U}, A, B) \quad \mapsto \quad \left[\begin{array}{c} \mathsf{coe}^{0\to 1}_{x.p\,x}(-) \\ A \quad \simeq \quad B \\ \mathsf{coe}^{1\to 0}_{x.p\,x}(-) \end{array}\right] \in A \simeq B$$

Recall that the inverse of this map is provided by a new type former, the V type (Section 3.1.6.2), that composes paths with isomorphisms.

The characterization of bridges in the universe follows the same blueprint. This time, however, the aim is to identify bridges with relations. In one direction, the bridge type former provides our map from bridges to relations.

$$p : \mathsf{Bridge}(\mathsf{U}, A, B) \quad \mapsto \quad \lambda\langle a, b\rangle.\, \mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, a, b) \in A \times B \to \mathsf{U}$$

That is, the relation induced by $p : \mathsf{Path}(\mathsf{U}, A, B)$ relates a with b when there is a bridge from a to b over p. We dub the equivalent of univalence *relativity*.

Definition 9.4.1 (Relativity). We say a universe U closed under bridge types is *relativistic* if the canonical map $\mathsf{Bridge}(\mathsf{U}, A, B) \to (A \times B \to \mathsf{U})$ defined above is an isomorphism.

To make our universes relativistic, we again introduce a new type former, the Gel *type*, which converts relations to bridges between types. The operational semantics for Gel types is shown in Figure 9.1. We call them "Gel" types because they share a basic structure with the G operation of the BCH cubical set model [BCH19, §3] but apply to r*el*ations rather than isomorphisms.

In comparison to the V type, the Gel type is refreshingly simple: given a relation $\Psi \setminus \boldsymbol{r}, a_0 : A_0, a_1 : A_1 \gg R$ type, it directly produces a bridge between A_0 and A_1, which is to say a type dependent on a fresh bridge variable $\boldsymbol{r}$. The proofs of the formation, introduction, and elimination rules for Gel are similar in complexity to those we gave for V types in Section 3.1.6.2: there are non-trivial coherence obligations to check, but they are of a fairly simple character.

Rules 9.4.2 (Gel pretype formation).

$$\frac{\Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad (\forall \varepsilon)\ \Psi \setminus \boldsymbol{r} \Vdash A_\varepsilon = A'_\varepsilon\ \mathsf{type} \qquad \Psi \setminus \boldsymbol{r}, a_0 : A_0, a_1 : A_1 \gg R = R'\ \mathsf{type}}{\Psi \Vdash \mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R) = \mathsf{Gel}_{\boldsymbol{r}}(A'_0, A'_1, a_0.a_1.R')\ \mathsf{pretype}}$$

$$\frac{\varepsilon \in \{0, 1\} \qquad \Psi \Vdash A_\varepsilon\ \mathsf{type}}{\Psi \Vdash \mathsf{Gel}_{\boldsymbol{\varepsilon}}(A_0, A_1, a_0.a_1.R) = A_\varepsilon\ \mathsf{pretype}}$$

Proof. Straightforward by coherent value introduction and expansion respectively. □

Note that the term-level arguments of Gel (the types A_0, A_1 and the relation R) are all precluded from using the interval term $\boldsymbol{r}$. The introduction form for Gel types takes a similar form: an element of $\mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R)$ consists of a pair of terms and a proof they are related, and draws a bridge across the Gel type between those terms. This is one direction of an isomorphism $\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, a_0.a_1.R), M_0, M_1) \simeq R[M_0/a_0, M_1/a_1]$ we intend to set up.

Rules 9.4.3 (Gel introduction).

$$\frac{(\forall \varepsilon)\ \Psi \setminus \boldsymbol{r} \Vdash M_\varepsilon = M'_\varepsilon \in A_\varepsilon \qquad \Psi \setminus \boldsymbol{r} \Vdash P = P' \in R[M_0/a_0, M_1/a_1]}{\Psi \Vdash \mathsf{gel}_{\boldsymbol{r}}(M_0, M_1, P) = \mathsf{gel}_{\boldsymbol{r}}(M'_0, M'_1, P') \in \mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R)}$$

$$\frac{\varepsilon \in \{0,1\} \qquad \Psi \Vdash M_\varepsilon \in A_\varepsilon}{\Psi \Vdash \mathsf{gel}_{\boldsymbol{\varepsilon}}(M_0, M_1, P) = M_\varepsilon \in A_\varepsilon}$$

Proof. Straightforward by coherent value introduction and expansion respectively. □

Finally, the projection operator ungel provides the other direction of the isomorphism: given a bridge over the Gel type, it produces a witness to the relation.

Rules 9.4.4 (Gel elimination).

$$\frac{\Psi, \boldsymbol{x} : \mathbf{I} \Vdash Q = Q' \in \mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, R)}{\Psi \Vdash \mathsf{ungel}(\boldsymbol{x}.Q) = \mathsf{ungel}(\boldsymbol{x}.Q') \in R[Q[\mathbf{0}/\boldsymbol{x}]/a_0, Q[\mathbf{1}/\boldsymbol{x}]/a_1]}$$

$$\frac{\Psi \Vdash P \in R[M_0/a_0, M_1/a_1]}{\Psi \Vdash \mathsf{ungel}(\boldsymbol{x}.\mathsf{gel}_{\boldsymbol{x}}(M_0, M_1, P)) = P \in R[M_0/a_0, M_1/a_1]}$$

$$\frac{\begin{array}{c}\Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad \Psi \setminus \boldsymbol{r} \Vdash A_0\ \mathsf{type} \qquad \Psi \setminus \boldsymbol{r} \Vdash A_1\ \mathsf{type} \\ \Psi \setminus \boldsymbol{r}, a_0 : A_0, a_1 : A_1 \Vdash R\ \mathsf{type} \qquad \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash Q \in \mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, a_0.a_1.R)\end{array}}{\Psi \Vdash Q[\boldsymbol{r}/\boldsymbol{x}] = \mathsf{gel}_{\boldsymbol{r}}(Q[\mathbf{0}/\boldsymbol{x}], Q[\mathbf{1}/\boldsymbol{x}], \mathsf{ungel}(\boldsymbol{x}.Q)) \in \mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R)}$$

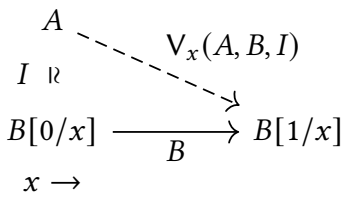
Proof. For the first, we cannot directly apply Lemma 3.1.38, because the argument Q to ungel appears under a binder (of $\boldsymbol{x}$). We instead give a hand-rolled argument by coherent head expansion. (We could have instead proven a more general form of Lemma 3.1.38, but this is the only place we would use it.) By Lemma 3.1.36, we have for every $\Psi' \Vdash \psi \in \Psi$ that $Q\psi \Downarrow \mathsf{gel}_{\boldsymbol{x}}(M_\psi, N_\psi, P_\psi)$ for some terms M_ψ, N_ψ, and P_ψ with $\Psi', \boldsymbol{x} : \mathbf{I} \Vdash Q\psi = \mathsf{gel}_{\boldsymbol{x}}(M_\psi, N_\psi, P_\psi) \in \mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, R)\psi$. By stability of typing under substitution, it follows that $P_{\mathsf{id}_\Psi}\psi = P_\psi \in R[M_{\mathsf{id}_\Psi}/a_0, N_{\mathsf{id}_\Psi}/a_1]$ for all ψ. By the same and the boundary rules for gel, we also have $\Psi \gg Q[\mathbf{0}/\boldsymbol{x}] = M_{\mathsf{id}_\Psi} \in A_0$ and likewise for $Q[\mathbf{1}/\boldsymbol{x}]$ and N_{id_Ψ}. Combining these, we have $\mathsf{ungel}(\boldsymbol{x}.Q)\psi \longmapsto^* P_\psi$ and $\Psi' \Vdash P_\psi = P\psi \in R[Q[\mathbf{0}/\boldsymbol{x}]/a_0, Q[\mathbf{1}/\boldsymbol{x}]/a_1]$ for all ψ, whence $\Psi \Vdash \mathsf{ungel}(\boldsymbol{x}.Q) = P_{\mathsf{id}_\Psi} \in R[Q[\mathbf{0}/\boldsymbol{x}]/a_0, Q[\mathbf{1}/\boldsymbol{x}]/a_1]$ by coherent head expansion.

We apply the same reasoning to the right hand side, finding that $\Psi \Vdash \mathsf{ungel}(\boldsymbol{x}.Q') = P'_{\mathsf{id}_\Psi} \in R[Q[\mathbf{0}/\boldsymbol{x}]/a_0, Q[\mathbf{1}/\boldsymbol{x}]/a_1]$ for some P'_{id_Ψ} that satisfies the equation $\Psi, \boldsymbol{x} : \mathbf{I} \Vdash Q' =$

$\mathsf{gel}_{\boldsymbol{x}}(M'_{\mathsf{id}_\Psi}, N'_{\mathsf{id}_\Psi}, P'_{\mathsf{id}_\Psi}) \in \mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, R)$. We combine this with the above to get that $\mathsf{gel}_{\boldsymbol{x}}(M_{\mathsf{id}_\Psi}, N_{\mathsf{id}_\Psi}, P_{\mathsf{id}_\Psi})$ is equal to $\mathsf{gel}_{\boldsymbol{x}}(M'_{\mathsf{id}_\Psi}, N'_{\mathsf{id}_\Psi}, P'_{\mathsf{id}_\Psi})$ at the Gel type, which implies that P_{id_Ψ} is equal to P'_{id_Ψ}; the result follows by transitivity.

The second rule is immediate by coherent expansion. For the third rule, we apply Lemma 3.1.36 to see that Q is equal to some gel value. The equation then follows by previously proven rules for gel. □

It is worth interrogating the difference in form between the V types of Section 3.1.6.2 and the new Gel types. In contrast to Gel, which simply converts a relation into a bridge of types, V extends an existing path of types by an isomorphism, as shown below.



The formulation of Gel is unavailable for V because path dimensions are structural: we cannot forbid the path interval variable from occurring in the other arguments, except in a sense by hypothesizing $x \equiv 0$ or $x \equiv 1$. We cannot put *all* of the inputs under one of these assumptions, as then we would have no type at the other endpoint! So we allow the argument B to depend on the variable. This is not a problem for univalence, because we always have the *option* of supplying a degenerate B—we are merely unable to *enforce* degeneracy. Note however that, conversely, a V-like type former would be insufficient for relativity. In the world of paths, we know that a degenerate path of types corresponds to the identity isomorphism, so composing with a degenerate path is a way of converting an isomorphism into a path. But *a degenerate bridge of types does not necessarily correspond to the identity relation.* Indeed, according to the formulation of relativity, a degenerate bridge of types B corresponds instead to the bridge relation $\mathsf{Bridge}(B, -, -)$, which can be distinct from the identity relation $\mathsf{Path}(B, -, -)$. The canonical example, of course, is $B := \mathsf{U}$.

Remark 9.4.5. Our Gel types are weaker than the equivalent introduced in [BCM15], our sole departure from their blueprint. They require that the equation

$$\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, a_0.a_1.R), M_0, M_1) = R[M_0/a_0, M_1/a_1]\ \mathsf{type}$$

hold up to *exact equality*, while for us this only holds up to an isomorphism. Note that we need this equation up to a path in order to show that Bridge and Gel are inverse, thus for relativity. In a cubical type theory, we can rely on univalence to turn the isomorphism into the necessary path. Without univalence, one must instead posit an exact

equation. Such an equation is rather onerous to satisfy in semantics. In particular, it requires Bernardy et al. to divert from a standard presheaf model to a model in what they call *refined* presheaves. By contrast, our own presheaf model (Section 11.1) does not require any such refinement.

Finally, we must check that Gel supports coercion and composition. In typical fashion, we prove reduction rules for the two operations first, then conclude they are well-typed and preserve equality. The reader familiar with the intricacies of cubical type theory will notice that the reduction for coercion is much simpler than its equivalent for V types (see, *e.g.*, [Ang19, §4.4.9]). This is a reflection of the fact that the principal direction of a coercion—the interval variable abstracted in the type line—is always a path variable. For V types, one must consider both the cases $\mathrm{coe}_{x.\mathsf{V}_x(A,B,I)}$ and $\mathrm{coe}_{x.\mathsf{V}_r(A,B,I)}$ where $r \neq x$, the former of which is the more involved. With Gel, on the other hand, the directions of the coercion is always orthogonal to the direction of the type itself.

Lemma 9.4.6 (Coercion reduction in Gel types).

$$\frac{\begin{array}{c}\Psi \Vdash s,t \in \mathbb{I} \qquad \Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad (\forall \varepsilon)\ \Psi \setminus \boldsymbol{r}, y : \mathbb{I} \Vdash A_\varepsilon\ \mathrm{type} \\ \Psi \setminus \boldsymbol{r}, y : \mathbb{I}, a_0 : A_0, a_1 : A_1 \gg R\ \mathrm{type} \qquad \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash Q \in \mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, a_0.a_1.R)[s/y] \\ (\forall \varepsilon)\ M^y_\varepsilon \coloneqq \mathrm{coe}^{s\to y}_{y.A_\varepsilon}(Q[\boldsymbol{\varepsilon}/\boldsymbol{x}]) \qquad P \coloneqq \mathrm{coe}^{s\to t}_{y.R[M_0^y/a_0, M_1^y/a_1]}(\mathsf{ungel}(\boldsymbol{x}.Q))\end{array}}{\Psi \Vdash \mathrm{coe}^{s\to t}_{y.\mathsf{Gel}_{\boldsymbol{r}}(A_0,A_1,a_0.a_1.R)}(Q[\boldsymbol{r}/\boldsymbol{x}]) = \mathsf{gel}_{\boldsymbol{x}}(M_0^t, M_1^t, P) \in \mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R)[t/y]}$$

Proof. By coherent expansion. Let $\Psi' \Vdash \psi \in \Psi$ be given. We are in one of two cases.

- $\boldsymbol{r}\psi = \boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}$. Then we have $\mathrm{coe}^{s\to t}_{y.\mathsf{Gel}_{\boldsymbol{r}}(A_0,A_1,a_0.a_1.R)}(Q[\boldsymbol{r}/\boldsymbol{x}])\psi \longmapsto \mathrm{coe}^{s\to t}_{y.A_\varepsilon}(Q[\boldsymbol{r}/\boldsymbol{x}])\psi$, and we know $\Psi' \Vdash \mathrm{coe}^{s\to t}_{y.A_\varepsilon}(Q[\boldsymbol{r}/\boldsymbol{x}])\psi = \mathsf{gel}_{\boldsymbol{x}}(M_0^t, M_1^t, P)\psi \in A_\varepsilon[t/y]\psi$ by the boundary rule for gel and definition of M^t_ε.
- $\boldsymbol{r}\psi = \boldsymbol{z}$ for some variable $\boldsymbol{z}$. Then the left hand side steps to the right hand side, which is well-typed by coercion for A_0, A_1, and R. As with extent, this relies on the affinity of bridge substitutions, in this case the fact that $\mathsf{ungel}(\boldsymbol{z}.Q\psi[\boldsymbol{z}/\boldsymbol{x}]) = \mathsf{ungel}(\boldsymbol{x}.Q)\psi$. □

Corollary 9.4.7 (Trivial coercion in Gel types).

$$\frac{\begin{array}{c}\Psi \Vdash s \in \mathbb{I} \qquad \Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad (\forall \varepsilon)\ \Psi \setminus \boldsymbol{r}, y : \mathbb{I} \Vdash A_\varepsilon\ \mathrm{type} \\ \Psi \setminus \boldsymbol{r}, y : \mathbb{I}, a_0 : A_0, a_1 : A_1 \gg R\ \mathrm{type} \qquad \Psi \Vdash Q \in \mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, a_0.a_1.R)[s/y]\end{array}}{\Psi \Vdash \mathrm{coe}^{s\to s}_{x.\mathsf{Gel}_{\boldsymbol{r}}(A_0,A_1,a_0.a_1.R)}(Q) = Q \in \mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R)[s/y]}$$

Proof. We are in one of two cases.

If $\boldsymbol{r}$ is a constant $\boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}$, then we have by coherent head expansion that $\Psi \Vdash \mathsf{coe}^{s \to s}_{x.\mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R)}(Q) = \mathsf{coe}^{s \to s}_{x.A_\varepsilon}(Q) \in \mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R)[s/y]$, in which case the result follows from the conditions on coercion in A_ε.

If $\boldsymbol{r}$ is a variable $\boldsymbol{x}$, then we apply Lemma 9.4.6 to find that the coercion is equal to $\mathsf{gel}_{\boldsymbol{x}}(M_0^s, M_1^s, P)$ as defined in that rule. The reduction of trivial coercions in A_0, A_1, and R shows that this term is equal to $\mathsf{gel}_{\boldsymbol{x}}(Q[\mathbf{0}/\boldsymbol{x}], Q[\mathbf{1}/\boldsymbol{x}], \mathsf{ungel}(\boldsymbol{x}.Q))$, thus to Q by uniqueness for Gel types. □

Lemma 9.4.8 (Composition reduction in Gel types).

$$\frac{\begin{array}{c}\Psi \Vdash s, t \in \mathbb{I} \qquad \Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad (\forall \varepsilon)\ \Psi \setminus \boldsymbol{r} \Vdash A_\varepsilon\ \mathsf{type} \qquad \Psi \setminus \boldsymbol{r}, a_0 : A_0, a_1 : A_1 \gg R\ \mathsf{type} \\ G \coloneqq \mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, a_0.a_1.R) \qquad \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash Q \in G \qquad (\forall i)\ \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I} \Vdash \xi_i \in \mathbb{F} \\ (\forall i, j)\ \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I}, x : \mathbb{I}, \xi_i, \xi_j \Vdash Q_i = Q_j \in G \qquad (\forall i)\ \Psi \setminus \boldsymbol{r}, \boldsymbol{x} : \mathbf{I}, \xi_i \Vdash Q = Q_i[s/x] \in G \\ M_\varepsilon^y \coloneqq \mathsf{hcom}^{s \to y}_{A_\varepsilon}(Q[\boldsymbol{\varepsilon}/\boldsymbol{x}]; \overrightarrow{\xi_i[\boldsymbol{\varepsilon}/\boldsymbol{x}] \hookrightarrow y.Q_i[\boldsymbol{\varepsilon}/\boldsymbol{x}]}) \\ P \coloneqq \mathsf{com}^{s \to t}_{y.R[M_0^y/a_0, M_1^y/a_1]}(\mathsf{ungel}(\boldsymbol{x}.Q); \overrightarrow{(\xi_i \hookrightarrow y.\mathsf{ungel}(\boldsymbol{x}.Q_i))_{\boldsymbol{x} \notin \xi_i}})\end{array}}{\Psi \Vdash \mathsf{hcom}^{s \to t}_G(Q[\boldsymbol{r}/\boldsymbol{x}]; \overrightarrow{\xi_i[\boldsymbol{r}/\boldsymbol{x}] \hookrightarrow y.Q_i[\boldsymbol{r}/\boldsymbol{x}]}) = \mathsf{gel}_{\boldsymbol{x}}(M_0^t, M_1^t, P) \in G[\boldsymbol{r}/\boldsymbol{x}]}$$

Proof. By coherent expansion. Let $\Psi' \Vdash \psi \in \Psi$ be given. We are in one of two cases.

- $\boldsymbol{r}\psi = \boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}$. Then the composition at $G\psi$ steps to the same composition $A_\varepsilon\psi$, which is $M_\varepsilon^t\psi$. We know $\Psi' \Vdash M_\varepsilon^t\psi = \mathsf{gel}_{\boldsymbol{x}}(M_0^t, M_1^t, P) \in A_\varepsilon[t/y]\psi$ by the boundary rule for gel.
- $\boldsymbol{r}\psi = \boldsymbol{z}$ for some variable $\boldsymbol{z}$. Then the left hand side steps to the right hand side, which is well-typed by composition for A_0, A_1, and R. We use affinity to ensure that $\mathsf{ungel}(\boldsymbol{z}.Q\psi[\boldsymbol{z}/\boldsymbol{x}]) = \mathsf{ungel}(\boldsymbol{x}.Q)\psi$ and $\mathsf{ungel}(\boldsymbol{z}.Q_i\psi[\boldsymbol{z}/\boldsymbol{x}]) = \mathsf{ungel}(\boldsymbol{x}.Q_i)\psi$. □

Corollary 9.4.9 (Boundary of composition in Gel types). Let $\Psi \Vdash \boldsymbol{r} \in \mathbf{I}$, $\Psi \setminus \boldsymbol{r} \Vdash A_\varepsilon$ type for each $\varepsilon \in \{0, 1\}$, and $\Psi \setminus \boldsymbol{r}, a_0 : A_0, a_1 : A_1 \gg R$ type be given, and set $G \coloneqq \mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R)$. Then the following rules are validated.

$$\frac{\begin{array}{c}\Psi \Vdash s = t \in \mathbb{I} \\ (\forall i)\ \Psi \Vdash \xi_i \in \mathbb{F} \qquad (\forall i, j)\ \Psi, x : \mathbb{I}, \xi_i, \xi_j \Vdash Q_i = Q_j \in G \qquad (\forall i)\ \Psi, \xi_i \Vdash Q = Q_i[s/x] \in G\end{array}}{\Psi \Vdash \mathsf{hcom}^{s \to t}_G(Q; \overrightarrow{\xi_i \hookrightarrow y.Q_i}) = Q \in G}$$

$$\frac{\begin{array}{c}\Psi \Vdash s, t \in \mathbb{I} \qquad (\forall i)\ \Psi \Vdash \xi_i \in \mathbb{F} \qquad (\forall i, j)\ \Psi, x : \mathbb{I}, \xi_i, \xi_j \Vdash Q_i = Q_j \in G \\ (\forall i)\ \Psi, \xi_i \Vdash Q = Q_i[s/x] \in G \qquad \Psi \Vdash \xi_j\ \mathsf{satisfied}\end{array}}{\Psi \Vdash \mathsf{hcom}^{s \to s}_G(Q; \overrightarrow{\xi_i \hookrightarrow y.Q_i}) = Q_j[t/x] \in G}$$

Proof. We are in one of two cases. If $\boldsymbol{r}$ is a constant $\boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}$, then these equations follow from the corresponding conditions in A_ε by coherent head expansion. If $\boldsymbol{r}$ is a variable $\boldsymbol{x}$, then we apply Lemma 9.4.8 to see that the composition is equal to $\mathsf{gel}_{\boldsymbol{x}}(M_0^t, M_1^t, P)$ as defined in that lemma. We then prove the two rules as follows.

For the first rule, if $s = t$, then the rule for trivial compositions in A_0, A_1, and R provide that $\Psi \Vdash M_\varepsilon^t = Q[\boldsymbol{\varepsilon}/\boldsymbol{x}] \in A_\varepsilon$ for $\varepsilon \in \{0, 1\}$ and $\Psi \Vdash P = \mathsf{ungel}(\boldsymbol{x}.Q) \in R[M_0^t/a_0, M_1^t/a_1]$, so that the combined term is equal to Q by uniqueness for Gel types.

For the second rule, suppose $\Psi \Vdash \xi_j$ satisfied for some j. By the rule for the boundary of composition in A_0 and A_1, we have that $\Psi \Vdash M_\varepsilon^t = Q_j[\boldsymbol{\varepsilon}/\boldsymbol{x}] \in A_\varepsilon$ for $\varepsilon \in \{0, 1\}$. Moreover, we know that ξ_j does not refer to $\boldsymbol{x}$. If it did, it would have to be of the form $\boldsymbol{x} \equiv \mathbf{0}$ or $\boldsymbol{x} \equiv \mathbf{1}$, which would contradict our assumption that $\Psi \Vdash \xi_j$ satisfied. Thus we have an entry in the composition defining P corresponding to ξ_j, and the composition boundary rule for R then implies that $\Psi \Vdash P \in \mathsf{ungel}(\boldsymbol{x}.Q_j)R[M_0^t/a_0, M_1^t/a_1]$. Again, we conclude by uniqueness that the combined term is equal to Q_i. □

Theorem 9.4.10 (Type formation).

$$\frac{\Psi \Vdash \boldsymbol{r} \in \mathbf{I} \qquad (\forall \varepsilon)\ \Psi \setminus \boldsymbol{r} \Vdash A_\varepsilon = A'_\varepsilon\ \mathsf{type} \qquad \Psi \setminus \boldsymbol{r}, a_0 : A_0, a_1 : A_1 \gg R = R'\ \mathsf{type}}{\Psi \Vdash \mathsf{Gel}_{\boldsymbol{r}}(A_0, A_1, a_0.a_1.R) = \mathsf{Gel}_{\boldsymbol{r}}(A'_0, A'_1, a_0.a_1.R')\ \mathsf{type}}$$

$$\frac{\varepsilon \in \{0, 1\} \qquad \Psi \Vdash A_\varepsilon\ \mathsf{type}}{\Psi \Vdash \mathsf{Gel}_{\boldsymbol{\varepsilon}}(A_0, A_1, a_0.a_1.R) = A_\varepsilon\ \mathsf{type}}$$

Proof. For the first, we must show the coercion and composition are well-typed and equal in the two Gel types and that the satisfy the necessary coherence conditions. The former follows from Lemmas 9.4.6 and 9.4.8—we reduce the operations and see that the reducts are well-typed and equal—and the coherence conditions hold by Corollaries 9.4.7 and 9.4.9.

The second equation requires, beyond the equality of pretypes, that coe and hcom are defined in equal ways at $\mathsf{Gel}_{\boldsymbol{\varepsilon}}(A_0, A_1, a_0.a_1.R)$ and A_ε. This is immediate, as coercion and composition in the former reduce to their equivalents in the latter. □

Chapter 10

Programming with parametricity

We now put our tools to the test and explore the consequences of internal parametricity. We start with a simple example to warm up in Section 10.1: a proof that the Church encoding of the booleans is isomorphic to the "primitive" boolean inductive type, which is a classical consequence of external parametricity [Wad07].

Next, in Section 10.2, we tackle a more theoretical result: the relativity principle, the equivalent of univalence for bridges, which states that bridges in U are isomorphic to relations. This result is novel: the principle is also true in Bernardy et al.'s calculus, but is ensured by imposing stricter equations on the equivalents of Gel types, while we do without them by leaning on function extensionality and univalence.

In Section 10.3, we identify the class of *bridge-discrete types*, those whose bridge types are isomorphic to their path types. We find that assumptions of bridge-discreteness play the role of classical parametricity's *identity extension lemma*, which Nuyts et al. [NVD17] note is conspicuously absent from parametric type theory in the style of Bernardy, Coquand, and Moulin. We also show that the type of booleans is bridge-discrete, suggesting more generally how the bridge types of (higher) inductive types can be characterized. In Section 10.4, we note that this implies the refutation of a form of the excluded middle as a corollary.

Finally, we fulfill in Section 10.5 our promise of using internal parametricity to prove coherence results for the smash product.

10.1 Characterizing Church booleans

Church encodings are a method of obtaining inductive-like types through impredicative quantification, essentially defining an inductive type as the type of elements to which its recursion principle can be applied. As an example, consider the type of booleans, which we have as a primitive type as a particularly trivial consequence of Part II.

inductive Bool **where**
| tt ∈ Bool
| ff ∈ Bool

A *Church boolean* is a function that takes a type and two elements of that type and returns a third element of that type. There are two canonical such booleans: the function that always returns the first element, and the function that always returns the second.

Definition 10.1.1. We define the type of Church booleans, $\mathbb{B}$ type, as follows.

$$\mathbb{B} \coloneqq (A : \mathsf{U}) \to A \to A \to A$$

We define terms $\mathbb{t}, \mathbb{f} \in \mathbb{B}$ by $\mathbb{t} \coloneqq \lambda A.\, \lambda t.\, \lambda f.\, t$ and $\mathbb{f} \coloneqq \lambda A.\, \lambda t.\, \lambda f.\, f$.

The Church booleans enjoy a recursion principle: given $c : \mathbb{B}$, $a_0 : A$, and $a_1 : A$, we have $c\, A\, a_0\, a_1 \in A$, and moreover we have reduction equations $\mathbb{t}\, A\, a_0\, a_1 = a_0 \in A$ and $\mathbb{f}\, A\, a_0\, a_1 = a_1 \in A$. However, they do not necessarily satisfy the *induction* (that is, dependent elimination) principle for the booleans—unless we assume parametricity [Wad90; Has94].

In the presence of parametricity and impredicative quantification, one can show that the *only* elements of $\mathbb{B}$ are $\mathbb{t}$ and $\mathbb{f}$, thus obtaining a type of booleans without relying on a primitive inductive type mechanism. Because our universes are predicative, this is not quite possible, but we *can* show that $\mathbb{B}$ is isomorphic to the primitive Bool when the latter type already exists.

Theorem 10.1.2. $\mathbb{B} \simeq \mathsf{Bool}$.

Proof. We can easily define functions in either direction. Starting with a Church boolean $c : \mathbb{B}$, we apply it to Bool, tt, and ff to obtain a primitive boolean; starting from a primitive boolean $b : \mathsf{Bool}$, we behave either as $\mathbb{t}$ or as $\mathbb{f}$ by case analysis on b.

$$H \coloneqq \lambda c.\, c\, \mathsf{Bool}\, \mathsf{tt}\, \mathsf{ff} \in \mathbb{B} \to \mathsf{Bool}$$
$$K \coloneqq \lambda b.\, \lambda A.\, \lambda t.\, \lambda f.\, \mathsf{elim}_{\mathsf{Bool}}(\_.A; b; t, f) \in \mathsf{Bool} \to \mathbb{B}$$

One inverse condition is easy to check. Given b:Bool, we construct a path $H\,(K\, b) \rightsquigarrow b$ by case analysis (*i.e.*, by $\mathsf{elim}_{\mathsf{Bool}}$). In the case that b is tt, we have $H\,(K\,\mathsf{tt}) = H\,\mathbb{t} \in \mathsf{Bool}$ and then $H\,\mathbb{t} = (\lambda A.\, \lambda t.\, \lambda f.\, t)\,\mathsf{Bool}\,\mathsf{tt}\,\mathsf{ff} = \mathsf{tt} \in \mathsf{Bool}$. By the same token, we have $H\,(K\,\mathsf{ff}) = \mathsf{ff} \in \mathsf{Bool}$.

It is the second condition that requires the use of parametricity. Let $c : \mathbb{B}$ be given. By function extensionality (Lemma 3.2.5), it suffices to show that, for all $A : \mathsf{U}$, $t : A$, and $f : A$, we have a path $(K\,(H\, c))\, A\, t\, f \rightsquigarrow c\, A\, t\, f$. Expanding the definition of H and simplifying,

the left hand side is equal to $\mathsf{elim}_{\mathsf{Bool}}(\_.A; c\ \mathsf{Bool}\ \mathsf{tt}\ \mathsf{ff}; t, f)$. Our goal, then, is to relate the behavior of c at Bool to its behavior at an arbitrary A.

To do so, we first single out a relation between Bool and A.

$$R \coloneqq \lambda\langle b, a\rangle.\, \mathsf{Path}(A, \mathsf{elim}_{\mathsf{Bool}}(\_.A; b; t, f), a) \in \mathsf{Bool} \times A \to \mathsf{U}$$

Notice that our goal is to show $R\,\langle c\ \mathsf{Bool}\ \mathsf{tt}\ \mathsf{ff}, c\ A\ t\ f\rangle$. To do so, we invoke parametricity: c takes related arguments to related results. In parametric type theory, that slogan cashes out in our ability to form Gel types. In particular, given a fresh bridge interval variable $\boldsymbol{x}$, we have the type $G_{\boldsymbol{x}} \coloneqq \mathsf{Gel}_{\boldsymbol{x}}(\mathsf{Bool}, A, R)$ corresponding to R with two canonical inhabitants.

$$t_{\boldsymbol{x}} \coloneqq \mathsf{gel}_{\boldsymbol{x}}(\mathsf{tt}, t, \lambda^{\mathbb{I}}\_.\, t) \in G_{\boldsymbol{x}}$$
$$f_{\boldsymbol{x}} \coloneqq \mathsf{gel}_{\boldsymbol{x}}(\mathsf{ff}, f, \lambda^{\mathbb{I}}\_.\, f) \in G_{\boldsymbol{x}}$$

The first element expresses that tt is related to t in R, as witnessed by the term $\lambda^{\mathbb{I}}\_.\, t \in R\,\langle \mathsf{tt}, t\rangle$; the second does the same for ff and f.

By applying c at this Gel type and its elements, we obtain a bridge relating $c\ \mathsf{Bool}\ \mathsf{tt}\ \mathsf{ff}$ and $c\ A\ t\ f$ over $\boldsymbol{x}.G_{\boldsymbol{x}}$; in effect, we have applied c at the relation R.

$$\lambda^{\mathbf{I}}\boldsymbol{x}.\, c\ G_{\boldsymbol{x}}\ t_{\boldsymbol{x}}\ f_{\boldsymbol{x}} \in \mathsf{Bridge}(\boldsymbol{x}.G_{\boldsymbol{x}}, c\ \mathsf{Bool}\ \mathsf{tt}\ \mathsf{ff}, c\ A\ t\ f)$$

Note the endpoints of this bridge: by definition, we have $G_{\mathbf{0}} = \mathsf{Bool} \in \mathsf{U}$, $t_{\mathbf{0}} = \mathsf{tt} \in \mathsf{Bool}$, and $f_{\mathbf{0}} = \mathsf{ff} \in \mathsf{Bool}$, likewise $G_{\mathbf{1}} = A \in \mathsf{U}$, $t_{\mathbf{1}} = t \in A$, and $f_{\mathbf{1}} = f \in A$. At $\mathbf{0}$, every relational term reduces to its $\mathbf{0}$ endpoint; at $\mathbf{1}$, to its $\mathbf{1}$ endpoint. Finally, the ungel eliminator takes a bridge over a Gel type to a witness of the underlying relation.

$$\mathsf{ungel}(\boldsymbol{x}.c\ G_{\boldsymbol{x}}\ t_{\boldsymbol{x}}\ f_{\boldsymbol{x}}) \in R\langle c\ \mathsf{Bool}\ \mathsf{tt}\ \mathsf{ff}, c\ A\ t\ f\rangle$$

This is exactly the type we needed to inhabit, and so we are satisfied.

It is perhaps instructive to consider how the inverse, constructed with parametricity primitives, evaluates when instantiated with a concrete $c : \mathbb{B}$. Say, for example, we take $\mathbb{t}$. Then the term $\mathsf{ungel}(\boldsymbol{x}.\mathbb{t}\ G_{\boldsymbol{x}}\ t_{\boldsymbol{x}}\ f_{\boldsymbol{x}})$ steps first to $\mathsf{ungel}(\boldsymbol{x}.t_{\boldsymbol{x}})$, then extracts the witness inside $t_{\boldsymbol{x}}$ to produce the reflexive proof $\lambda^{\mathbb{I}}\_.\, t \in R\langle \mathbb{t}\ \mathsf{Bool}\ \mathsf{tt}\ \mathsf{ff}, \mathbb{t}\ A\ t\ f\rangle$. Likewise, instantiating c with $\mathbb{f}$ produces the reflexive path $\lambda^{\mathbb{I}}\_.\, f$ packaged in $f_{\boldsymbol{x}}$. □

Note that there are actually closed elements of $\mathbb{B}$ that are exactly equal neither to $\mathbb{t}$ or to $\mathbb{f}$. For example, we have the term $\lambda A.\, \lambda t.\, \lambda f.\, \mathsf{coe}^{0\to 1}_{\_.A}(t) \in \mathbb{B}$; a coercion in a degenerate type line is only guaranteed to be equal to its input up to a path in general, not exactly, so this term is not exactly equal to $\mathbb{t}$. Nevertheless, the result above shows it is equal to $\mathbb{t}$ up to a path. Notice also that we obtain parametricity results despite the fact that

the cubical language *does* contain terms that evaluate by case analysis on types, namely the Kan operators coe and hcom. Indeed, nothing prevents us from including a general type-case operator in the language; it will simply fail to be well-typed if used in a non-parametric way. In short, our parametricity is not a syntactic condition, but a semantic one. Indeed, the fact that $\lambda A.\, \lambda t.\, \lambda f.\, \mathrm{coe}^{0\to 1}_{\_.A}(t)$ can be given the type $\mathbb{B}$ reflects the fact that coe is defined on all elements of the universe, in particular Gel types, and so must behave parametrically.

10.2 The relativity principle

For our next trick, we prove the relativity principle (Definition 9.4.1): the isomorphism between bridges in the universe and relations, the equivalent of the univalence principle for bridges. Like univalence, it is rare that we need the principle in all its strength: as in the previous sections, we usually only use the ability to turn a relation into a bridge, which is to say the Gel type former. Nevertheless, it forms the conceptual backbone of the system.

Notation 10.2.1. Given a relation $R \in A \times B \to \mathsf{U}$ valued in some universe, we abbreviate the type $\mathsf{Gel}_{\boldsymbol{r}}(A, B, a.b.R\,\langle a, b\rangle) \in \mathsf{U}$ as $\mathsf{Gel}_{\boldsymbol{r}}(A, B, R)$.

One thing to notice in the following proof is its reliance on function extensionality and univalence. To prove an isomorphism between $\mathsf{Bridge}(\mathsf{U}, A, B)$ and $A \times B \to \mathsf{U}$, we necessarily must prove path equations in function types and the universe. In particular, it is essential that we can turn the isomorphism $\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, R), a, b) \simeq R\,\langle a, b\rangle$ into a path; this is one of the inverse conditions for the main relativity isomorphism. The argument to come would not, therefore, go through in a type theory built on ITT rather than cubical type theory. To avoid the issue in their own formalism, Bernardy, Coquand, and Moulin therefore impose this as an exact equation, $\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, R), a, b) = R\,\langle a, b\rangle$ type, as discussed in Remark 9.4.5.

Lemma 10.2.2 (Bridges in an isomorphism type). Let $\boldsymbol{x} : \mathbf{I} \gg A, B$ type be given together with isomorphisms $i_0 : A[\mathbf{0}/\boldsymbol{x}] \simeq B[\mathbf{0}/\boldsymbol{x}]$ and $i_1 : A[\mathbf{1}/\boldsymbol{x}] \simeq B[\mathbf{1}/\boldsymbol{x}]$. Then we have an isomorphism of the following type.

$$\mathsf{Bridge}(\boldsymbol{x}.A \simeq B, i_0, i_1)$$
$$\simeq$$
$$((a_0 : A[\mathbf{0}/\boldsymbol{x}])\ (a_1 : A[\mathbf{1}/\boldsymbol{x}]) \to \mathsf{Bridge}(\boldsymbol{x}.A, a_0, a_1) \simeq \mathsf{Bridge}(\boldsymbol{x}.B, i_0\, a_0, i_1\, a_1))$$

Proof. The type of isomorphisms (Definition 1.2.1) is defined using product, function, and path types. We already have characterizations of bridges in each of these types

(Lemma 9.2.2, Theorem 9.3.2, and Lemma 9.2.3). Applying each of these characterizations in turn beginning with $\mathsf{Bridge}(\boldsymbol{x}.A \simeq B, i_0, i_1)$, we arrive at a type readily seen to be isomorphic to the right hand side above. Specifically, the characterizations deliver us to a type of tuples consisting of a family of functions $\mathsf{Bridge}(\boldsymbol{x}.A, a_0, a_1) \to \mathsf{Bridge}(\boldsymbol{x}.B, i_0\, a_0, i_1\, a_1)$ indexed over a_0, a_1, two families of functions in the opposite direction, and proofs that these are left and right inverses respectively for every a_0, a_1. It then remains only to pull the indices a_0, a_1 out to the top level. □

Theorem 10.2.3 (Relativity). Let A, B type be given. Then the following function is an isomorphism.

$$\lambda p.\, \lambda\langle a, b\rangle.\, \mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, a, b) \in \mathsf{Bridge}(\mathsf{U}, A, B) \to (A \times B \to \mathsf{U})$$

Proof. We use the Gel types to build our candidate inverse.

$$\lambda R.\, \lambda^{\mathbb{I}}\boldsymbol{x}.\, \mathsf{Gel}_{\boldsymbol{x}}(A, B, R) \in (A \times B \to \mathsf{U}) \to \mathsf{Bridge}(\mathsf{U}, A, B)$$

We have two inverse conditions to show.

1. $(R : A \times B \to \mathsf{U}) \to (\lambda\langle a, b\rangle.\, \mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, R), a, b)) \rightsquigarrow R$.

   By function extensionality (Lemma 3.2.5), it suffices to construct a path in U from $\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, R), a, b)$ to $R\,\langle a, b\rangle$ for all $a : A$ and $b : B$. In turn, by univalence (Theorem 3.2.9), it is enough to give an *isomorphism* from $\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, R), a, b)$ to $R\,\langle a, b\rangle$ for all a, b.

   Such an isomorphism is provided up to exact equality by the constructor and eliminator for the Gel type, with functions in either direction defined as follows.

$$\lambda t.\, \lambda^{\mathbb{I}}\boldsymbol{x}.\, \mathsf{gel}_{\boldsymbol{x}}(a, b, t) \in R\,\langle a, b\rangle \to \mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, R), a, b)$$
$$\lambda q.\, \mathsf{ungel}(\boldsymbol{x}.q\,\boldsymbol{x}) \in \mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, R), a, b) \to R\,\langle a, b\rangle$$

   By the reduction and uniqueness rules for Gel types, these functions cancel each other up to exact equality.

2. $(p : \mathsf{Bridge}(\mathsf{U}, A, B)) \to (\lambda^{\mathbb{I}}\boldsymbol{x}.\, \mathsf{Gel}_{\boldsymbol{x}}(A, B, \mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, -, -))) \rightsquigarrow p$.

   Let p:$\mathsf{Bridge}(\mathsf{U}, A, B)$ be given. By the characterization of paths in bridges (Lemma 9.2.3), it is equivalent to give a bridge between paths of the following type.

$$\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Path}(\mathsf{U}, \mathsf{Gel}_{\boldsymbol{x}}(A, B, \mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, -, -)), p\,\boldsymbol{x}), \lambda^{\mathbb{I}}\_.\, A, \lambda^{\mathbb{I}}\_.\, B)$$

   Now we take advantage of univalence to replace the inner type of paths in U above with a type of isomorphisms, finding that the above type is isomorphic to the following.

$$\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, \mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, -, -)) \simeq p\,\boldsymbol{x}, \mathsf{coe}^{0\simeq 1}_{\_.A}, \mathsf{coe}^{0\simeq 1}_{\_.B})$$

Finally, we apply Lemma 10.2.2, reducing this bridge in the isomorphism type to an isomorphism of bridge types; this is where we rely on extent. We are left to show, for every $a : A$ and $b : B$, the following isomorphism.

$$\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, \mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, -, -)), a, b)$$
$$\simeq$$
$$\mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, \mathrm{coe}^{0\to 1}_{-.A}(a), \mathrm{coe}^{0\to 1}_{-.B}(b))$$

We have a pair of paths $\lambda^{\mathbb{I}} y.\, \mathrm{coe}^{y\to 1}_{-.A}(a) \in \mathsf{Path}(A, \mathrm{coe}^{0\to 1}_{-.A}(a), a)$ and $\lambda^{\mathbb{I}} y.\, \mathrm{coe}^{y\to 1}_{-.B}(b) \in \mathsf{Path}(B, \mathrm{coe}^{0\to 1}_{-.A}(b), b)$, so we can delete the coercions in the above. This leaves us to show $\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Gel}_{\boldsymbol{x}}(A, B, \mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, -, -)), a, b)$ is isomorphic to $\mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, a, b)$. This is an instance of the inverse condition we have already proven, instantiated at the relation $\mathsf{Bridge}(\boldsymbol{x}.p\,\boldsymbol{x}, -, -)$. □

10.3 Bridge-discrete types

In classical parametricity, the *identity extension lemma* is a key basic result: it says that the relational interpretation of an operator on types takes identity relations to identity relations. In particular, the interpretation of any closed type is the identity relation. In internal parametricity, the corresponding statement would be that any "homogeneous" bridge type $\mathsf{Bridge}(A, M_0, M_1)$—where A is a type rather than a line $\boldsymbol{x} : \mathbf{I} \gg A$ type—is isomorphic to $\mathsf{Path}(A, M_0, M_1)$, the relation $\mathsf{Bridge}(A, -, -)$ being the analogue of the relational interpretation of A. We have just seen that this is false: we have $\mathsf{Bridge}(\mathsf{U}, A, B) \simeq (A \times B \to \mathsf{U}) \not\simeq (A \simeq B) \simeq \mathsf{Path}(\mathsf{U}, A, B)$. We can, however, identify the types that *do* satisfy this property, which we call the *bridge-discrete types*. We will see that the class of bridge-discrete types is closed under every type former we have introduced *except* the universe, and that assumptions of bridge-discreteness can effectively play the role of the identity extension lemma.

A bit more precisely, we define the bridge-discrete types to be those for which the canonical map from paths to bridges is an isomorphism.

Definition 10.3.1. Let A type be given. We define a map loosen_A as follows, so that $\mathsf{loosen}_A \in \mathsf{Path}(A, a_0, a_1) \to \mathsf{Bridge}(A, a_0, a_1)$ for any $a_0, a_1 : A$.

$$\mathsf{loosen}_A \coloneqq \lambda p.\, \mathrm{coe}^{0\to 1}_{x.\mathsf{Bridge}(A, p\,0, p\,x)}(\lambda^{\mathbf{I}}\_.\, p\,0)$$

We say A is *bridge-discrete* if loosen_A is an isomorphism for every pair of endpoints, *i.e.*, if the following type is inhabited.

$$\mathsf{IsBDisc}(A) \coloneqq (a_0, a_1 : A) \to \mathsf{IsIso}(\mathsf{Path}(A, a_0, a_1), \mathsf{Bridge}(A, a_0, a_1), \mathsf{loosen}_A)$$

We define the *universe of bridge-discrete types* as $\mathsf{U}_{\mathsf{bdisc}} \coloneqq (A : \mathsf{U}) \times \mathsf{IsBDisc}(A)$.

Remark 10.3.2. The map loosen_A takes reflexive paths to reflexive bridges, up to a path: for any $a : A$, we have $\lambda^{\mathbb{I}} y.\, \mathsf{coe}^{y \to 1}_{x.\mathsf{Bridge}(A,a,a)}(\lambda^{\mathbf{I}}\_.\, a) \in \mathsf{loosen}_A\,(\lambda^{\mathbb{I}}\_.\, a) \rightsquigarrow (\lambda^{\mathbf{I}}\_.\, a)$.

By requiring loosen_A in particular to be an isomorphism, we ensure that the type $\mathsf{IsBDisc}(A)$ is a proposition (Definition 3.2.1), as the type $\mathsf{IsIso}(A, B, f)$ is always a proposition [Uni13, Theorem 4.3.2]. To show that a type is bridge-discrete, however, it suffices to show that *any* map is an isomorphism (indeed, a retraction); this is a special case of a result stated in Section 3.2.

Lemma 10.3.3 (Bridge-discreteness by retract). Let A type and suppose we have two functions as follows.

$$f : (a_0, a_1 : A) \to \mathsf{Bridge}(A, a_0, a_1) \to \mathsf{Path}(A, a_0, a_1)$$
$$g : (a_0, a_1 : A) \to \mathsf{Path}(A, a_0, a_1) \to \mathsf{Bridge}(A, a_0, a_1)$$

If $g\, a_0\, a_1\, (f\, a_0\, a_1\, q) \rightsquigarrow q$ for all $a_0, a_1 : A$ and $q : \mathsf{Bridge}(A, a_0, a_1)$, then A is bridge-discrete.

Proof. By Lemma 3.2.8. □

Before we show that any types are bridge-discrete, the following demonstrates why this collection of types is worth identifying. Whenever we want to prove a parametricity result about a type that involves an "external" type parameter, we likely need to assume that parameter is bridge-discrete.

Theorem 10.3.4. For any bridge-discrete A type, we have an isomorphism of the following type.

$$((B : \mathsf{U}) \to (A \to B) \to B) \simeq A$$

Proof. Set $\mathbb{A} \coloneqq ((B : \mathsf{U}) \to (A \to B) \to B)$. We follow the pattern established in Theorem 10.1.2. The functions in either direction are simple to define.

$$H \coloneqq \lambda c.\, c\, A\, (\lambda a.\, a) \in \mathbb{A} \to A$$
$$K \coloneqq \lambda a.\, \lambda A.\, \lambda f.\, f\, a \in A \to \mathbb{A}$$

Moreover, calculation shows immediately that $H\,(K\,a) = a \in A$ for any $a : A$. For the other inverse, we work by parametricity. We must show that for every $c : \mathbb{A}$, $B : \mathsf{U}$, and $f : A \to B$, we have a path from $f\,(c\,A\,(\lambda a.\, a))$ to $c\,B\,f$. We define a relation from A to B, the *graph of* f, by $R \coloneqq \lambda\langle a, b\rangle.\, \mathsf{Path}(B, f\,a, b)$. We aim to apply c at the Gel type for R in a fresh direction $\boldsymbol{x}$.

To do so, we will need a term $f_{\boldsymbol{x}} \in A \to \mathsf{Gel}_{\boldsymbol{x}}(A, B, R)$ to supply as the function argument. It is here that we use bridge-discreteness of A, which gives us some function $t \in (a_0, a_1 : A) \to \mathsf{Bridge}(A, a_0, a_1) \to \mathsf{Path}(A, a_0, a_1)$. Using t, we can define $f_{\boldsymbol{x}}$ by way of extent.

$$f_{\boldsymbol{x}} \coloneqq \lambda a.\, \mathsf{extent}_{\boldsymbol{x}}(a; a_0.a_0, a_1.f\ a_1, a_0.a_1.q.\lambda^{\mathbf{I}}\boldsymbol{x}.\, \mathsf{gel}_{\boldsymbol{x}}(a_0, f\ a_1, \lambda^{\mathbb{I}} y.\, f\ (t\ a_0\ a_1\ q\ y)))$$

We have $f_{\mathbf{0}} = \lambda a.\, a \in A \to A$ and $f_{\mathbf{1}} = f \in A \to B$. In words, to construct $f_{\boldsymbol{x}}$, we need to know that any $a_0, a_1 : A$ related by $\mathsf{Bridge}(A, -, -)$ satisfy $f\ a_0 \rightsquigarrow f\ a_1$. This is only guaranteed if bridges in the constant A give rise to paths in the same; thus the necessity of bridge-discreteness.

The remainder of the argument proceeds as in Theorem 10.1.2. By applying c at $f_{\boldsymbol{x}}$, we obtain an element of $\mathsf{Gel}_{\boldsymbol{x}}(A, B, R)$, which we can ungel to get the witness to R we require.

$$\mathsf{ungel}(\boldsymbol{x}.c\ (\mathsf{Gel}_{\boldsymbol{x}}(A, B, R))\ f_{\boldsymbol{x}}) \in \mathsf{Path}(B, f\ (c\ A\ (\lambda a.\, a)), c\ B\ f) \qquad \square$$

Now we take a look at types formed from bridge-discrete arguments. When we have a bridge-discrete family of types, we have the following result, which will help analyze dependent function and product types.

Lemma 10.3.5. Let A type and $a : A \gg B$ type. Suppose that B is bridge-discrete for every $a : A$. Then for any path $a_0, a_1 : A$, $p : \mathsf{Path}(A, a_0, a_1)$, and $b_0 : B[a_0/a]$ and $b_1 : B[a_1/a]$, we have the following isomorphism.

$$\mathsf{Path}(x.B[p\ x/a], b_0, b_1) \simeq \mathsf{Bridge}(\boldsymbol{x}.B[\mathsf{loosen}_A\ p\ \boldsymbol{x}/a], b_0, b_1)$$

Proof. By Lemma 3.2.3, it suffices to show this when $a_0 = a_1$ and p is the degenerate path $\lambda\_.\, a_0$. In that case, we have $\mathsf{loosen}_A\ (\lambda^{\mathbb{I}}\_.\, a_0) \rightsquigarrow (\lambda^{\mathbf{I}}\_.\, a_0)$, and so the isomorphism we must construct follows directly from bridge-discreteness of $B[a_0/a]$. $\square$

Theorem 10.3.6. The universe of bridge-discrete types is closed under product, function, path, and bridge types.

Proof. For product types, $(a : A) \times B$, this follows from Lemmas 3.2.4 and 9.2.2, using Lemma 10.3.5 to get a correspondence between dependent bridges and paths in B over the correspondence in A. For functions, it follows from Lemma 3.2.6 and Theorem 9.3.2. For paths and bridges, it follows from Lemma 9.2.3. $\square$

We may also show that inductive types preserve bridge-discreteness. Here, we show as an example that Bool is bridge-discrete. The argument is more involved than for the preceding types, employing in particular relativity (that is, Gel types). This presents an interesting parallel to the use of univalence to characterize the path types of higher inductive types, sketched in our discussion of descent in Chapter 4.

Theorem 10.3.7 (Bridges in Bool). Bool is bridge-discrete.

Proof. We define a right inverse to $\mathsf{loosen}_{\mathsf{Bool}} \in \mathsf{Path}(\mathsf{Bool}, b_0, b_1) \to \mathsf{Bridge}(\mathsf{Bool}, b_0, b_1)$ given $b_0, b_1 : \mathsf{Bool}$. We make use of the Gel type for the path relation in Bool.

$$\boldsymbol{x} : \mathbf{I} \gg G_{\boldsymbol{x}} \coloneqq \mathsf{Gel}_{\boldsymbol{x}}(\mathsf{Bool}, \mathsf{Bool}, \mathsf{Path}(\mathsf{Bool}, -, -))\ \mathsf{type}$$

This type has two canonical elements corresponding to the reflexive proofs of equality in Bool, $t_{\boldsymbol{x}} \coloneqq \mathsf{gel}_{\boldsymbol{x}}(\mathsf{tt}, \mathsf{tt}, \lambda^{\mathbb{I}}\_.\,\mathsf{tt})$ and $f_{\boldsymbol{x}} \coloneqq \mathsf{gel}_{\boldsymbol{x}}(\mathsf{ff}, \mathsf{ff}, \lambda^{\mathbb{I}}\_.\,\mathsf{ff})$. We first define an auxiliary map $F_{\boldsymbol{x}} \in \mathsf{Bool} \to G_{\boldsymbol{x}}$ by case analysis, returning the corresponding reflexivity path in each case.

$$F_{\boldsymbol{x}} \coloneqq \lambda b.\, \mathsf{elim}_{\mathsf{Bool}}(\_.G_{\boldsymbol{x}}; b; t_{\boldsymbol{x}}, f_{\boldsymbol{x}}) \in \mathsf{Bool} \to G_{\boldsymbol{x}}$$

Note we can transform this bridge in a function type into a function from bridges to bridges, then use ungel to extract a path from the resulting bridge over $\boldsymbol{x}.G_{\boldsymbol{x}}$.

$$F \coloneqq \lambda p.\, \mathsf{ungel}(\boldsymbol{x}.F_{\boldsymbol{x}}\ (p\ \boldsymbol{x})) \in \mathsf{Bridge}(\mathsf{Bool}, b_0, b_1) \to \mathsf{Path}(\mathsf{Bool}, F_{\mathbf{0}}\ b_0, F_{\mathbf{1}}\ b_1)$$

Modulo the not-quite-correct endpoints of the output, this will be our candidate right inverse. Conversely, we can extract a map $G_{\boldsymbol{x}} \to \mathsf{Bool}$ from $\mathsf{loosen}_{\mathsf{Bool}}$ using extent.

$$L_{\boldsymbol{x}} \coloneqq \lambda g.\, \mathsf{extent}_{\boldsymbol{x}}(g; b_0.b_0, b_1.b_1, \_.\_.q.\mathsf{loosen}_{\mathsf{Bool}}\ (\mathsf{ungel}(\boldsymbol{x}.q\ \boldsymbol{x}))) \in G_{\boldsymbol{x}} \to \mathsf{Bool}$$

To check the inverse condition, we start by checking that $F_{\boldsymbol{x}}$ is right inverse to $L_{\boldsymbol{x}}$, constructing a term of the following type.

$$P_{\boldsymbol{x}} \in (b : \mathsf{Bool}) \to \mathsf{Path}(\mathsf{Bool}, L_{\boldsymbol{x}}\ (F_{\boldsymbol{x}}\ b), b)$$

Examining $L_{\boldsymbol{x}}\ (F_{\boldsymbol{x}}\ \mathsf{tt})$, we have the following sequence of equations and paths in Bool.

$$\begin{aligned} L_{\boldsymbol{x}}\ (F_{\boldsymbol{x}}\ \mathsf{tt}) &= \mathsf{extent}_{\boldsymbol{x}}(t_{\boldsymbol{x}}; b_0.b_0, b_1.b_1, \_.\_.q.\mathsf{loosen}_{\mathsf{Bool}}\ (\mathsf{ungel}(\boldsymbol{x}.q\ \boldsymbol{x}))) \\ &= \mathsf{loosen}_{\mathsf{Bool}}(\mathsf{ungel}(\boldsymbol{x}.t_{\boldsymbol{x}})\ \boldsymbol{x} \\ &= \mathsf{loosen}_{\mathsf{Bool}}(\lambda^{\mathbb{I}}\_.\,\mathsf{tt})\ \boldsymbol{x} \\ &\rightsquigarrow (\lambda^{\mathbf{I}}\_.\,\mathsf{tt})\ \boldsymbol{x} \\ &= \mathsf{tt} \end{aligned}$$

We likewise have a path $L_{\boldsymbol{x}}\ (F_{\boldsymbol{x}}\ \mathsf{ff}) \rightsquigarrow \mathsf{ff}$, so we can define $P_{\boldsymbol{x}}$ by case analysis. Finally, we move from bridges of functions to functions of bridges once more, defining the term $\lambda q.\, \lambda^{\mathbb{I}}\boldsymbol{x}.\, P_{\boldsymbol{x}}\ (q\ \boldsymbol{x})$ of the following type.

$$(q : \mathsf{Bridge}(\mathsf{Bool}, b_0, b_1)) \to \mathsf{Bridge}(\boldsymbol{x}.\mathsf{Path}(\mathsf{Bool}, L_{\boldsymbol{x}}\ (F_{\boldsymbol{x}}\ q\ \boldsymbol{x}), q\ \boldsymbol{x}), P_{\mathbf{0}}\ b_0, P_{\mathbf{1}}\ b_1)$$

Exploiting the isomorphism between bridges in path types and paths in bridge types (Lemma 9.2.3), this induces a family of dependent paths as follows.

$$(q : \mathsf{Bridge}(\mathsf{Bool}, b_0, b_1)) \to \mathsf{Path}(y.\mathsf{Bridge}(\mathsf{Bool}, P_{\mathbf{0}}\, b_0\, y, P_{\mathbf{1}}\, b_1\, y), \lambda^{\mathbf{I}}\boldsymbol{x}.\, L_{\boldsymbol{x}}\, (F_{\boldsymbol{x}}\, (q\, \boldsymbol{x})), q)$$

A quick calculation, reducing the extent term in $L_{\boldsymbol{x}}$, reveals that $\lambda^{\mathbf{I}}\boldsymbol{x}.\, L_{\boldsymbol{x}}\, (F_{\boldsymbol{x}}\, q\, \boldsymbol{x})$ is equal to $\mathsf{loosen}_{\mathsf{Bool}}(F\, q)$ in $\mathsf{Path}(\mathsf{Bool}, F_{\mathbf{0}}\, b_0, F_{\mathbf{1}}\, b_1)$.

We are almost at the end; we just have to deal with some endpoints. Abstracting slightly, we have shown that the following is inhabited for some pair of singletons $\langle b_0', \eta_0\rangle$: $(b_0' : \mathsf{Bool}) \times \mathsf{Path}(\mathsf{Bool}, b_0', b_0)$ and $\langle b_1', \eta_1\rangle : (b_1' : \mathsf{Bool}) \times \mathsf{Path}(\mathsf{Bool}, b_1', b_1)$.

$$(f : \mathsf{Bridge}(\mathsf{Bool}, b_0, b_1) \to \mathsf{Path}(\mathsf{Bool}, b_0', b_1')) \times$$
$$(q : \mathsf{Bridge}(\mathsf{Bool}, b_0, b_1)) \to \mathsf{Path}(y.\mathsf{Bridge}(\mathsf{Bool}, \eta_0\, y, \eta_1\, y), \mathsf{loosen}_{\mathsf{Bool}}(f\, q), q)$$

Namely, we have a witness at $\eta_0 \coloneqq P_{\mathbf{0}}\, b_0$ and $\eta_1 \coloneqq P_{\mathbf{1}}\, b_1$. By singleton contractibility (Lemma 3.2.2), this choice of singletons is equal, up to a path, to the pair of reflexive singletons $\langle b_0, \lambda^{\mathbb{I}}\_.\, b_0\rangle$ and $\langle b_1, \lambda^{\mathbb{I}}\_.\, b_1\rangle$. By coercion, we thus obtain an element of the type above instantiated with that choice of singletons, which is exactly a right inverse for $\mathsf{loosen}_{\mathsf{Bool}}$. □

To give an idea of how this argument would proceed for inductive types more generally, we sketch the proof for Nat, showing how to define the map from bridges to paths.

Lemma 10.3.8. For any n_0, n_1:Nat, we have a map $\mathsf{Bridge}(\mathsf{Nat}, n_0, n_1) \to \mathsf{Path}(\mathsf{Nat}, n_0, n_1)$.

Proof. We begin again with Gel type for the path relation in Nat.

$$\boldsymbol{x} : \mathbf{I} \gg G_{\boldsymbol{x}} \coloneqq \mathsf{Gel}_{\boldsymbol{x}}(\mathsf{Nat}, \mathsf{Nat}, \mathsf{Path}(\mathsf{Nat}, -, -))\ \mathsf{type}$$

We have canonical terms $z_{\boldsymbol{x}} \in G_{\boldsymbol{x}}$ and $s_{\boldsymbol{x}} \in G_{\boldsymbol{x}} \to G_{\boldsymbol{x}}$ defined as follows.

$$\begin{aligned} z_{\boldsymbol{x}} &\coloneqq \mathsf{gel}_{\boldsymbol{x}}(\mathsf{zero}, \mathsf{zero}, \lambda^{\mathbb{I}}\_.\, \mathsf{zero}) \\ s_{\boldsymbol{x}} &\coloneqq \lambda g.\, \mathsf{extent}_{\boldsymbol{x}}(g; m_0.\mathsf{suc}(m_0), m_1.\mathsf{suc}(m_1), m_0.m_1.g'.\boldsymbol{y}.S) \\ &\qquad \text{where } S = \mathsf{gel}_{\boldsymbol{y}}(\mathsf{suc}(m_0), \mathsf{suc}(m_1), \lambda^{\mathbb{I}} z.\, \mathsf{suc}(\mathsf{ungel}(\boldsymbol{y}.g'\, \boldsymbol{y})\, z)) \end{aligned}$$

The term $z_{\boldsymbol{x}}$ carries the reflexive path $\mathsf{zero} \rightsquigarrow \mathsf{zero}$, while $s_{\boldsymbol{x}}$ takes a gel term containing a path $m_0 \rightsquigarrow m_1$ and returns a path $\mathsf{suc}(m_0) \rightsquigarrow \mathsf{suc}(m_1)$. Now we get a function from Nat to its path relation.

$$F_{\boldsymbol{x}} \coloneqq \lambda b.\, \mathsf{elim}_{\mathsf{Nat}}(\_.G_{\boldsymbol{x}}; b; z_{\boldsymbol{x}}, \_.g.s_{\boldsymbol{x}}\, g) \in \mathsf{Nat} \to G_{\boldsymbol{x}}$$

Given a bridge $q : \mathsf{Bridge}(\mathsf{Nat}, n_0, n_1)$, we apply $F_{\boldsymbol{x}}$ pointwise to get a path.

$$F \coloneqq \mathsf{ungel}(\boldsymbol{x}.F_{\boldsymbol{x}}\, (q\, \boldsymbol{x})) \in \mathsf{Path}(\mathsf{Nat}, F_{\mathbf{0}}\, n_0, F_{\mathbf{1}}\, n_1)$$

By inspection, we have $F_\varepsilon\, n = \mathsf{elim}_{\mathsf{Nat}}(\_.\mathsf{Nat}; n; \mathsf{zero}, \_.m.\mathsf{suc}(m)) \in \mathsf{Nat}$ for $\varepsilon \in \{0, 1\}$, and the latter is path-equal to n by induction. □

The bridge-discrete universe even inherits univalence and relativity. This means in particular that we can use internal parametricity to characterize functions out of the bridge-discrete universe. For example, we can show that the "bridge discrete Church boolean" type, $(A : \mathsf{U}_{\mathsf{bdisc}}) \to \mathsf{fst}(A) \to \mathsf{fst}(A) \to \mathsf{fst}(A)$, is also isomorphic to Bool.

Theorem 10.3.9. $\mathsf{U}_{\mathsf{bdisc}}$ is univalent, in the sense that $\mathsf{Path}(\mathsf{U}_{\mathsf{bdisc}}, A, B)$ is isomorphic to $\mathsf{fst}(A) \simeq \mathsf{fst}(B)$ by way of the coercion isomorphism map for any $A, B : \mathsf{U}_{\mathsf{bdisc}}$.

Proof. By univalence of U, it is equivalent to show that the projection function from $\mathsf{Path}(\mathsf{U}_{\mathsf{bdisc}}, A, B)$ to $\mathsf{Path}(\mathsf{U}, \mathsf{fst}(A), \mathsf{fst}(B))$ is an isomorphism. This follows quickly from Lemma 3.2.4 and the fact that $\mathsf{IsBDisc}(C)$ is a proposition for any $C : \mathsf{U}$. □

Relativity comes down to the closure of the bridge-discrete universe under Gel-types.

Theorem 10.3.10. Let A, B type and $a : A, b : B \gg R$ type be given. If A, B are bridge-discrete and R is pointwise bridge-discrete, then $\mathsf{Gel}_{\boldsymbol{x}}(A, B, a.b.R)$ is bridge-discrete for any fresh $\boldsymbol{x}$.

Proof. Set $G_{\boldsymbol{x}} :\equiv \mathsf{Gel}_{\boldsymbol{x}}(A, B, a.b.R)$. We aim to show that paths and bridges from g_0 to g_1 in $G_{\boldsymbol{x}}$ are isomorphic for any $g_0, g_1 : G_{\boldsymbol{x}}$. By applying extent, it suffices to show this is the case when either $\boldsymbol{x}$ is an endpoint or g_0 and g_1 are points on bridges. When $\boldsymbol{x}$ is an endpoint, we apply the loosen_A or loosen_B isomorphism accordingly. For the remaining case, we need

$$\mathsf{Path}(G_{\boldsymbol{x}}, q_0\,\boldsymbol{x}, q_1\,\boldsymbol{x}) \simeq \mathsf{Bridge}(G_{\boldsymbol{x}}, q_0\,\boldsymbol{x}, q_1\,\boldsymbol{x})$$

for all $q_0 : \mathsf{Bridge}(\boldsymbol{x}.G_{\boldsymbol{x}}, a_0, b_0)$ and $q_1 : \mathsf{Bridge}(\boldsymbol{x}.G_{\boldsymbol{x}}, a_1, b_1)$, coherently with the endpoint cases. Now, by Lemma 10.2.2, it suffices to construct an isomorphism of the following type for any $p \in \mathsf{Path}(A, a_0, a_1)$ and $p' \in \mathsf{Path}(B, b_0, b_1)$.

$$\begin{gathered}\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Path}(G_{\boldsymbol{x}}, q_0\,\boldsymbol{x}, q_1\,\boldsymbol{x}), p, p') \\ \simeq \\ \mathsf{Bridge}(\boldsymbol{x}.\mathsf{Bridge}(G_{\boldsymbol{x}}, q_0\,\boldsymbol{x}, q_1\,\boldsymbol{x}), \mathsf{loosen}_A\,p, \mathsf{loosen}_B\,p')\end{gathered}$$

By Lemma 3.2.3, we can assume that the paths p and p' are reflexive; together with the fact that loosen takes reflexive paths to reflexive bridges, this simplifies our goal to the following.

$$\begin{gathered}\mathsf{Bridge}(\boldsymbol{x}.\mathsf{Path}(G_{\boldsymbol{x}}, q_0\,\boldsymbol{x}, q_1\,\boldsymbol{x}), \lambda^{\mathbb{I}}\_.\, a_0, \lambda^{\mathbb{I}}\_.\, b_0) \\ \simeq \\ \mathsf{Bridge}(\boldsymbol{x}.\mathsf{Bridge}(G_{\boldsymbol{x}}, q_0\,\boldsymbol{x}, q_1\,\boldsymbol{x}), \lambda^{\mathbf{I}}\_.\, a_0, \lambda^{\mathbf{I}}\_.\, b_0)\end{gathered}$$

Next, we know that swapping iterated bridge and path types is an isomorphism, so it is enough to prove the following isomorphism where we have flipped the order of type constructors on either side.

$$\mathsf{Path}(\mathsf{Bridge}(\boldsymbol{x}.G_{\boldsymbol{x}}, a, b), q_0, q_1) \simeq \mathsf{Bridge}(\mathsf{Bridge}(\boldsymbol{x}.G_{\boldsymbol{x}}, a, b), q_0, q_1)$$

Finally, we know that $\mathsf{Bridge}(\boldsymbol{x}.G_{\boldsymbol{x}}, a, b) \simeq R$. Thus this follows directly from bridge-discreteness of R. □

Corollary 10.3.11. $\mathsf{U}_{\mathsf{bdisc}}$ is relativistic. That is, for any $A, B{:}\mathsf{U}_{\mathsf{bdisc}}$, we have the following isomorphism with the forward map given by the bridge type former (which preserves bridge-discreteness per Theorem 10.3.6).

$$\mathsf{Bridge}(\mathsf{U}_{\mathsf{bdisc}}, A, B) \simeq (\mathsf{fst}(A) \times \mathsf{fst}(B) \to \mathsf{U}_{\mathsf{bdisc}})$$

Proof. Suppose $A = \langle A_0, p\rangle$ and $B = \langle B_0, q\rangle$. By Lemma 9.2.3, $\mathsf{Bridge}(\mathsf{U}_{\mathsf{bdisc}}, A, B)$ is isomorphic to the following.

$$(C : \mathsf{Bridge}(\mathsf{U}, A_0, B_0)) \times \mathsf{Bridge}(\boldsymbol{x}.\mathsf{IsBDisc}(C\,\boldsymbol{x}), p, q)$$

The right hand type, meanwhile, is also isomorphic to a product.

$$(R : A_0 \times B_0 \to \mathsf{U}) \times ((a : A)\ (b : B) \to \mathsf{IsBDisc}(R\,\langle a, b\rangle))$$

We have an isomorphism between the first components by relativity of U, implemented by the bridge and Gel types. Each second component, meanwhile, is a proposition. (For the first, it is straightforward to check that the bridge type of a proposition is a proposition.) It therefore suffices to show that the isomorphism of the first components takes C such that $\mathsf{Bridge}(\boldsymbol{x}.\mathsf{IsBDisc}(C\,\boldsymbol{x}), p, q)$ to R such that $((a : A)\ (b : B) \to \mathsf{IsBDisc}(R\,\langle a, b\rangle))$ and vice versa. The forward direction is the fact that bridge-types preserve bridge-discreteness, the converse is the fact that Gel-types preserve the same. □

Note that the relativity of $\mathsf{U}_{\mathsf{bdisc}}$ implies that $\mathsf{U}_{\mathsf{bdisc}}$ itself is not bridge discrete.

Example 10.3.12. $\mathbb{B}_{\mathsf{bdisc}} \coloneqq (A{:}\mathsf{U}_{\mathsf{bdisc}}) \to \mathsf{fst}(A) \to \mathsf{fst}(A) \to \mathsf{fst}(A)$ is isomorphic to Bool.

Proof (Sketch). Suppose we are given $c : \mathbb{B}_{\mathsf{bdisc}}$. Given $A : \mathsf{U}_{\mathsf{bdisc}}$, $t : \mathsf{fst}(A)$ and $f : \mathsf{fst}(A)$, we define a relation $R \in \mathsf{Bool} \times \mathsf{fst}(A) \to \mathsf{U}$ as in Theorem 10.1.2.

$$R \coloneqq \lambda\langle b, a\rangle.\, \mathsf{Path}(\mathsf{fst}(A), \mathsf{elim}_{\mathsf{Bool}}(\_.\mathsf{fst}(A); b; t, f), a) \in \mathsf{Bool} \times \mathsf{fst}(A) \to \mathsf{U}$$

The type $\mathsf{fst}(A)$ is bridge-discrete by assumption and bridge-discrete types are closed under path types, so this relation is pointwise bridge-discrete. Thus Theorem 10.3.10 gives us a bridge in $\mathsf{U}_{\mathsf{bdisc}}$ from Bool (coupled with the proof of bridge-discreteness from Theorem 10.3.7) to A corresponding to R. Applying c at this bridge, we then proceed as in the proof of Theorem 10.3.10. □

10.4 The excluded middle

The bridge-discreteness of Bool implies a cute result concerning the law of the excluded middle in parametric type theory.

Definition 10.4.1. Write $\neg A :\equiv (A \to \mathsf{Void})$ for intuitionistic negation. We define three varieties of excluded middle.

$$\begin{aligned}
\mathsf{LEM}_\infty &:\equiv (A : \mathsf{U}) \to (b : \mathsf{Bool}) \times \mathsf{elim}_{\mathsf{Bool}}(_.\mathsf{U}; b; A, \neg A) \\
\mathsf{LEM}_{-1} &:\equiv (A : \mathsf{U}) \to \mathsf{IsProp}(A) \to (b : \mathsf{Bool}) \times \mathsf{elim}_{\mathsf{Bool}}(_.\mathsf{U}; b; A, \neg A) \\
\mathsf{LEM}_\neg &:\equiv (A : \mathsf{U}) \to (b : \mathsf{Bool}) \times \mathsf{elim}_{\mathsf{Bool}}(_.\mathsf{U}; b; \neg A, \neg\neg A)
\end{aligned}$$

We call LEM_∞ the *unrestricted excluded middle*, LEM_{-1} the *excluded middle for propositions*, and $\mathsf{LEM}_\neg$ the *weak excluded middle*.

Clearly LEM_∞ implies LEM_{-1}. Moreover, LEM_{-1} implies $\mathsf{LEM}_\neg$, as every negated type is a proposition—this is a consequence of function extensionality and the fact that the empty type is a proposition.

The unrestricted excluded middle is independent of ITT, but is contradictory to the univalence axiom. We refer to [Uni13, Corollary 4.2.7] for a full proof, but the basic intuition is as follows. An element $d : \mathsf{LEM}_\infty$ picks out a distinguished element of every inhabited type; in particular, some distinguished element M of Bool. At the same time, univalence implies that d has an action on isomorphisms. By examining the action of d on the automorphism $\mathsf{not} \in \mathsf{Bool} \simeq \mathsf{Bool}$ that swaps the two booleans, we can derive a path from $(\mathsf{not}\, M)$ to M, a clear contradiction.

The excluded middle for propositions, on the other hand, is perfectly consistent with homotopy type theory and is validated in the simplicial model thereof [KL20]; propositions have at most one element up to path equality, so there is no problem choosing elements uniformly with respect to isomorphisms. By contrast, even the weak law of the excluded middle is refuted in parametric type theory.

Lemma 10.4.2. If A type is bridge-discrete, then any function $f : \mathsf{U} \to A$ is constant.

Proof. For any pair of types $B_0, B_1 : \mathsf{U}$, we have an abundance of bridges between them; to choose one, we have a bridge $\lambda^{\mathbb{I}} \boldsymbol{x}.\, \mathsf{Gel}_{\boldsymbol{x}}(B_0, B_1, \_.\_.\mathsf{Void}) \in \mathsf{Bridge}(\mathsf{U}, B_0, B_1)$ given by the empty relation. By applying f pointwise, we obtain a bridge $\lambda^{\mathbb{I}} \boldsymbol{x}.\, f\ (\mathsf{Gel}_{\boldsymbol{x}}(B_0, B_1, \_.\_.\mathsf{Void}))$ from $f\, B_0$ to $f\, B_1$. As A is bridge-discrete, this bridge induces a path between the same. □

Theorem 10.4.3. The weak excluded middle is refuted.

Proof. Suppose we are given $d : \mathsf{LEM}_\neg$. Then $\lambda A.\, \mathsf{fst}(d\, A)$ is a function $\mathsf{U} \to \mathsf{Bool}$, so is constant by Lemma 10.4.2 and Theorem 10.3.7. But this implies that $\mathsf{fst}(d\, \mathsf{Unit})$ and $\mathsf{fst}(d\, \mathsf{Void})$ have the same value, from which we readily derive a contradiction. □

Corollary 10.4.4. The excluded middle for propositions is refuted.

For further analysis of the relationship between classical principles and parametricity, we refer to Booij et al. [BELS16].

10.5 Iterated smash products

Finally, we return to our motivating example of an application of parametricity unique to higher-dimensional type theory: coherence laws for the smash product. Recall from Chapter 8 that the smash product of two pointed types $A_*, B_* \in \mathsf{U}_* :\equiv (A : \mathsf{U}) \times A$ is the following higher inductive type.

$$
\begin{aligned}
&A_* : \mathsf{U}_*, B_* : \mathsf{U}_* \gg \textbf{inductive}\ A_* \wedge B_* \ \textbf{where}\\
&\mid \langle\!\langle a : A, b : B \rangle\!\rangle \in A_* \wedge B_*\\
&\mid \circledast^{\mathsf{L}} \in A_* \wedge B_*\\
&\mid \mathsf{spoke}^{\mathsf{L}}(b : B, x : \mathbb{I}) \in A_* \wedge B_* \quad [x \equiv 0 \hookrightarrow \circledast^{\mathsf{L}} \mid x \equiv 1 \hookrightarrow \langle\!\langle a_0, b \rangle\!\rangle]\\
&\mid \circledast^{\mathsf{R}} \in A_* \wedge B_*\\
&\mid \mathsf{spoke}^{\mathsf{R}}(a : A, x : \mathbb{I}) \in A_* \wedge B_* \quad [x \equiv 0 \hookrightarrow \circledast^{\mathsf{R}} \mid x \equiv 1 \hookrightarrow \langle\!\langle a, b_0 \rangle\!\rangle]
\end{aligned}
$$

Notation 10.5.1 (Recollections from Chapter 8). We abbreviate $A :\equiv \mathsf{fst}(A_*) \in \mathsf{U}$ and $a_0 :\equiv \mathsf{snd}(A_*) \in A$ for the underlying type and point of a given pointed type $A_* \in \mathsf{U}_*$. Given $A_*, B_* \in \mathsf{U}_*$, we have the type $(A_* \to B_*) :\equiv (f : A \to B) \times \mathsf{Path}(B, f\, a_0, b_0) \in \mathsf{U}$ of functions that send the basepoint of A to that of B. For the pointed type of such functions, we write $(A_* \to_* B_*) :\equiv \langle A_* \to B_*, \langle \lambda\_.\, b_0, \lambda^{\mathbb{I}}\_.\, b_0 \rangle \rangle \in \mathsf{U}_*$; we write f_* for elements of this type and abbreviate $f :\equiv \mathsf{fst}(f_*)$ and $f_0 :\equiv \mathsf{snd}(f)$ as with types. A pointed isomorphism, written $A_* \simeq B_*$, is an isomorphism whose underlying function is pointed.

The elements of the smash product are pairs $\langle\!\langle a : A, b : B \rangle\!\rangle$ but with all elements of the form $\langle\!\langle a_0, b \rangle\!\rangle$ identified with a distinguished point $\circledast^{\mathsf{L}}$ and all elements of the form $\langle\!\langle a, b_0 \rangle\!\rangle$ identified with $\circledast^{\mathsf{R}}$. We write $A_* \wedge_* B_*$ for the pointed type $\langle A_* \wedge B_*, \langle\!\langle a_0, b_0 \rangle\!\rangle \rangle$.

In Chapter 8, we imagined that various coherence conditions expected of the commutator and associator—themselves feasible if tedious to construct—could be verified automatically by using parametricity. First, we note that it suffices to characterize the inhabitants of types of the following form, where the input and output smash products are both associated in the same (arbitrary) way.

$$(A_{1*}, \dots, A_{n*} : \mathsf{U}_*) \to (A_{1*} \wedge_* \cdots \wedge_* A_{n*}) \to (A_{1*} \wedge_* \cdots \wedge_* A_{n*}) \qquad (*)$$

To show that a commutator $F \in (A_*, B_* : \mathsf{U}_*) \to A_* \wedge_* B_* \to B_* \wedge_* A_*$ is an isomorphism, for example, it suffices to show that the composite $\lambda c.\, F\, B_*\, A_*\, (F\, B_*\, A_*\, c)$ is the (pointed) identity function for every $A_*, B_* : \mathsf{U}_*$. By the same token, we can show that a pair of associator functions

$$G \in (A_*, B_*, C_* : \mathsf{U}_*) \to (A_* \wedge_* B_*) \wedge_* C_* \to A_* \wedge_* (B_* \wedge_* C_*)$$
$$H \in (A_*, B_*, C_* : \mathsf{U}_*) \to A_* \wedge_* (B_* \wedge_* C_*) \to (A_* \wedge_* B_*) \wedge_* C_*$$

constitute an isomorphism by showing that the two round-trip composites are identities. The pentagon identity displayed in Chapter 8 can also be cast as the equality of a round-trip composite to the identity function at a type of the form (∗); higher coherences amount to equalities between such equalities. We cannot expect that *every* parametric term of the form (∗) is an identity function, because the existence of basepoints makes the pointed constant function a possibility. However, we will see that this is the only exception. It is moreover easy to check that such a function is not constant by testing it on small inputs, namely the pointed type $\mathsf{Bool}_* :\equiv \langle \mathsf{Bool}, \mathsf{tt} \rangle$. For example, $K \in (A_*, B_* : \mathsf{U}_*) \to A_* \wedge_* B_* \to A_* \wedge_* B_*$ is an identity function if and only if we have $K\, \mathsf{Bool}_*\, \mathsf{Bool}_*\, \langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle \rightsquigarrow \langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle$.

To illustrate the argument, we start with the binary case.

Theorem 10.5.2. Any function $(A_*, B_* : \mathsf{U}_*) \to A_* \wedge_* B_* \to A_* \wedge_* B_*$ is either the polymorphic identity or the polymorphic constant pointed function.

The proof of this theorem will involve a bit of serious higher-dimensional programming. We want to avoid clouding the main thrust of the proof with routine verification of boundary conditions, so we will mainly dispatch higher-dimensional obligations without much comment. Our argument is not that one is *completely* saved from verifying such conditions. Rather, our claim is that parametricity permits the characterization of terms of the form (∗) without being swamped in complexity as n increases.

We first introduce a couple of auxiliary terms that will come in handy for checking coherence conditions.

Definition 10.5.3 (Concatenation by inverse). let $M \in A$, $r \in \mathbb{I}$, and $x : \mathbb{I} \gg N \in A$ with $r \equiv 1 \gg M = N[1/x] \in A$ be given. For any $s \in \mathbb{I}$, define $\mathsf{conc\text{-}inv}^{r,s}_A(M, x.N) \in A$ as follows.

$$\mathsf{conc\text{-}inv}^{r,s}_A(M, x.N) :\equiv \mathsf{hcom}^{1 \to s}_A(M; r \equiv 0 \hookrightarrow \_.M, r \equiv 1 \hookrightarrow x.N)$$

The term $\mathsf{conc\text{-}inv}^{r,0}_A(M, x.N)$ is the result of concatenating M (as a path in direction r) with the inverse of $x.N$; we will use the general form $\mathsf{conc\text{-}inv}^{r,s}_A(M, x.N)$ to relate the composite to other terms.

Lemma 10.5.4 (Join connection). For any $a_0, a_1 : A$ and $p : \mathsf{Path}(A, a_0, a_1)$, we have a term as follows, a square with p on the two "0" sides and reflexivity on the two "1" sides.

$$\mathsf{cnx}_A(p) \in \mathsf{Path}(x.\mathsf{Path}(A, p\,x, a_1), p, \lambda^{\mathbb{I}}\_.\, a_1)$$

Proof. By J for paths (Lemma 3.2.3), it suffices to construct such a term in the case that p is a reflexive path $\lambda^{\mathbb{I}}\_.\, a$, in which case we may take $\lambda^{\mathbb{I}}\_.\, \lambda^{\mathbb{I}}\_.\, a$. □

Our uses of parametricity for this theorem are limited to cases where the relation is the graph of a function, so we introduce some notation for this case.

Notation 10.5.5. Given $f : A \to B$, write $\mathsf{Gr}_r(A, B, f) :\equiv \mathsf{Gel}_r(A, B, a.b.\mathsf{Path}(B, f\,a, b))$. Given $f_* : A_* \to B_*$, define $\mathsf{Gr}_r(A_*, B_*, f_*) :\equiv \langle \mathsf{Gr}_r(A, B, f), \mathsf{gel}_r(a_0, b_0, f_0)\rangle \in \mathsf{U}_*$.

The first property we need of the smash product is that it acts on pointed functions in either argument.

Definition 10.5.6. Given pointed functions $f_* : A_* \to C_*$ and $g_* : B_* \to D_*$, we define a map $f_* \wedge g_* \in (A_* \wedge B_*) \to (C_* \wedge D_*)$ by smash product elimination as follows.

$$(f_* \wedge g_*)\, s :\equiv \left[\begin{array}{l} \textbf{case } s \textbf{ of} \\ \mid \langle\!\langle a, b\rangle\!\rangle \mapsto \langle\!\langle f\,a, g\,b\rangle\!\rangle \\ \mid \circledast^{\mathsf{L}} \mapsto \circledast^{\mathsf{L}} \\ \mid \mathsf{spoke}^{\mathsf{L}}(b, y) \mapsto \mathsf{conc\text{-}inv}^{y,0}_{C_* \wedge D_*}(\mathsf{spoke}^{\mathsf{L}}(g\,b, y), z.\langle\!\langle f_0\,z, g\,b\rangle\!\rangle) \\ \mid \circledast^{\mathsf{R}} \mapsto \circledast^{\mathsf{R}} \\ \mid \mathsf{spoke}^{\mathsf{R}}(a, x) \mapsto \mathsf{conc\text{-}inv}^{x,0}_{C_* \wedge D_*}(\mathsf{spoke}^{\mathsf{R}}(f\,a, y), z.\langle\!\langle f\,a, g_0\,z\rangle\!\rangle) \end{array}\right]$$

This map is basepoint-preserving; we write $f_* \wedge_* g_* :\equiv \langle f_* \wedge g_*, \lambda^{\mathbb{I}} x.\, \langle\!\langle f_0\,x, g_0\,x\rangle\!\rangle\rangle$ for the pointed function.

The second is that $\mathsf{Bool}_* :\equiv \langle \mathsf{Bool}, \mathsf{tt}\rangle$ is a unit for the smash product; actually, we only need the special case $\mathsf{Bool}_* \wedge \mathsf{Bool}_* \simeq \mathsf{Bool}$.

Lemma 10.5.7 (Smash of booleans). $\mathsf{Bool}_* \wedge \mathsf{Bool}_*$ is isomorphic to Bool; in particular, any element of $\mathsf{Bool}_* \wedge \mathsf{Bool}_*$ is path-equal to either $\langle\!\langle \mathsf{tt}, \mathsf{tt}\rangle\!\rangle$ or $\langle\!\langle \mathsf{ff}, \mathsf{ff}\rangle\!\rangle$.

Proof. In one direction, we define $F \in \mathsf{Bool} \to \mathsf{Bool}_* \wedge \mathsf{Bool}_*$ to send tt to $\langle\!\langle \mathsf{tt}, \mathsf{tt}\rangle\!\rangle$ and ff to $\langle\!\langle \mathsf{ff}, \mathsf{ff}\rangle\!\rangle$. In the other, we define $G \in \mathsf{Bool}_* \wedge \mathsf{Bool}_* \to \mathsf{Bool}$ to send $\langle\!\langle \mathsf{ff}, \mathsf{ff}\rangle\!\rangle$ to ff and all other constructors to tt. Clearly $\lambda b.\, G\,(F\,b)$ is the identity. For the other inverse condition, we show $(s{:}\mathsf{Bool}_* \wedge \mathsf{Bool}_*) \to \mathsf{Path}(\mathsf{Bool}_* \wedge \mathsf{Bool}_*, s, F\,(G\,s))$ by smash product induction as follows.

- Case $\langle\!\langle \mathsf{tt}, \mathsf{tt} \rangle\!\rangle$: Reflexivity.
- Case $\langle\!\langle \mathsf{tt}, \mathsf{ff} \rangle\!\rangle$:
  $\lambda^{\mathbb{I}} y.\, \mathsf{hcom}^{0 \to 1}_{\mathsf{Bool}_* \wedge \mathsf{Bool}_*}(\mathsf{spoke}^{\mathsf{L}}(\mathsf{tt}, y); y \equiv 0 \hookrightarrow x.\mathsf{spoke}^{\mathsf{L}}(\mathsf{ff}, x), y \equiv 1 \hookrightarrow \_.\langle\!\langle \mathsf{tt}, \mathsf{tt} \rangle\!\rangle)$.
- Case $\langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle$: Reflexivity.
- Case $\circledast^{\mathsf{L}}$: $\lambda^{\mathbb{I}} y.\, \mathsf{spoke}^{\mathsf{L}}(\mathsf{tt}, y)$.
- Case $\mathsf{spoke}^{\mathsf{L}}(\mathsf{tt}, x)$: $\mathsf{cnx}_{\mathsf{Bool}_* \wedge \mathsf{Bool}_*}(\lambda^{\mathbb{I}} y.\, \mathsf{spoke}^{\mathsf{L}}(\mathsf{tt}, y))\, x$.
- Case $\mathsf{spoke}^{\mathsf{L}}(\mathsf{ff}, x)$:
  $\lambda^{\mathbb{I}} y.\, \mathsf{hcom}^{0 \to x}_{\mathsf{Bool}_* \wedge \mathsf{Bool}_*}(\mathsf{spoke}^{\mathsf{L}}(\mathsf{tt}, y); y \equiv 0 \hookrightarrow x.\mathsf{spoke}^{\mathsf{L}}(\mathsf{ff}, x), y \equiv 1 \hookrightarrow \_.\langle\!\langle \mathsf{tt}, \mathsf{tt} \rangle\!\rangle)$.

The cases for $\langle\!\langle \mathsf{tt}, \mathsf{ff} \rangle\!\rangle$, $\circledast^{\mathsf{R}}$, and $\mathsf{spoke}^{\mathsf{R}}$ are obtained by taking the cases for $\langle\!\langle \mathsf{ff}, \mathsf{tt} \rangle\!\rangle$, $\circledast^{\mathsf{L}}$, and $\mathsf{spoke}^{\mathsf{L}}$ respectively and replacing $\mathsf{spoke}^{\mathsf{L}}$ with $\mathsf{spoke}^{\mathsf{R}}$ everywhere. □

Finally, we need part of a characterization of bridges across smash product types. For our purposes, we only need to analyze bridges across $\boldsymbol{x}.(\mathrm{Gr}_{\boldsymbol{x}}(A_*, C_*, f_*) \wedge \mathrm{Gr}_{\boldsymbol{x}}(B_*, D_*, g_*))$; we also do not need a full isomorphism, only a map in one direction.

Lemma 10.5.8 (Graph Lemma for ∧). For any $\boldsymbol{r} : \mathbf{I}$, there is a map

$$\wedge\text{-graph}_{\boldsymbol{r}} \in \mathrm{Gr}_{\boldsymbol{r}}(A_*, C_*, f_*) \wedge \mathrm{Gr}_{\boldsymbol{r}}(B_*, D_*, g_*) \to \mathrm{Gr}_{\boldsymbol{r}}(A_* \wedge B_*, C_* \wedge D_*, f_* \wedge g_*)$$

equal to the identity function on $A_* \wedge B_*$ when $\boldsymbol{r} = \mathbf{0}$ and on $C_* \wedge D_*$ when $\boldsymbol{r} = \mathbf{1}$.

Proof. We define the map by induction on the smash product in the domain.

- Case $\langle\!\langle m, n \rangle\!\rangle$: We test whether $\boldsymbol{r}$ is a constant or variable using extent. In the constant cases, we return $\langle\!\langle m, n \rangle\!\rangle$. In the case $\boldsymbol{r}$ is a variable $\boldsymbol{x}$, we learn that m and n are the instantiation at $\boldsymbol{x}$ of bridges over their types; by uniqueness, they are of the form $m = \mathsf{gel}_{\boldsymbol{x}}(a, c, p)$ and $n = \mathsf{gel}_{\boldsymbol{x}}(b, d, q)$. We return $\mathsf{gel}_{\boldsymbol{x}}(\langle\!\langle a, b \rangle\!\rangle, \langle\!\langle c, d \rangle\!\rangle, \lambda^{\mathbb{I}} z.\, \langle\!\langle p\, y, q\, y \rangle\!\rangle)$.
- Case $\circledast^{\mathsf{L}}$: We return $\mathsf{gel}_{r}(\circledast^{\mathsf{L}}, \circledast^{\mathsf{L}}, \lambda^{\mathbb{I}} \_.\, \circledast^{\mathsf{L}})$.
- Case $\circledast^{\mathsf{R}}$: Symmetric to $\circledast^{\mathsf{L}}$.
- Case $\mathsf{spoke}^{\mathsf{L}}(n, y)$: We test whether $\boldsymbol{r}$ is a constant or variable using extent. In the constant cases, we return $\mathsf{spoke}^{\mathsf{L}}(n, y)$. In the case $\boldsymbol{r}$ is a variable $\boldsymbol{x}$, we learn that n is the instantiation at $\boldsymbol{x}$ of a bridge; by uniqueness, it is of the form $n = \mathsf{gel}_{\boldsymbol{x}}(b, d, q)$. We return $\mathsf{gel}_{\boldsymbol{x}}(\mathsf{spoke}^{\mathsf{L}}(b, y), \mathsf{spoke}^{\mathsf{L}}(d, y), \lambda^{\mathbb{I}} z.\, \cdots)$, where $\cdots$ is the following composite.

$$\mathsf{hcom}^{1 \to 0}_{C_* \wedge D_*}\left(\begin{array}{lcl} \mathsf{spoke}^{\mathsf{L}}(q\, z, y); & y \equiv 0 & \hookrightarrow \quad \_.\circledast^{\mathsf{L}} \\ & y \equiv 1 & \hookrightarrow \quad w.\langle\!\langle \mathsf{cnx}_A(f_0)\, z\, w, q\, z \rangle\!\rangle \\ & z \equiv 0 & \hookrightarrow \quad w.\mathsf{conc\text{-}inv}^{y,w}_{C_* \wedge D_*}(\mathsf{spoke}^{\mathsf{L}}(g\, b, y), z.\langle\!\langle f_0\, z, g\, b \rangle\!\rangle) \\ & z \equiv 1 & \hookrightarrow \quad \_.\mathsf{spoke}^{\mathsf{L}}(d, y) \end{array}\right)$$

- Case $\mathsf{spoke}^{\mathsf{R}}(m, y)$: Symmetric to $\mathsf{spoke}^{\mathsf{L}}(n, y)$.

When $\boldsymbol{r}$ is a constant, the resulting function simplifies to a term path-equal to the identity function on $A_* \wedge B_*$. We may therefore apply an hcom to adjust the boundary and obtain a function that is exactly the identity when $\boldsymbol{r} = \mathbf{0}$ or $\boldsymbol{r} = \mathbf{1}$. □

The following lemma represents the sole use of parametricity in the final proof.

Lemma 10.5.9 (Workhorse lemma). Let $F \in (A_*, B_* : \mathsf{U}_*) \to A \to B \to A_* \wedge_* B_*$. Then F is path equal to one of either $(\lambda \_.\, \lambda \_.\, \lambda a.\, \lambda b.\, \langle\!\langle a, b \rangle\!\rangle)$ or $(\lambda A_*.\, \lambda B_*.\, \lambda \_.\, \lambda \_.\, \langle\!\langle a_0, b_0 \rangle\!\rangle)$.

Proof. We show that F is determined by the value of $F\,\mathsf{Bool}_*\,\mathsf{Bool}_*\,\mathsf{ff}\,\mathsf{ff}$. Let $A_* : \mathsf{U}_*$, $B_* : \mathsf{U}_*$, $a : A$, and $b : B$ be given.

We have a pointed function $[a]_* \in \mathsf{Bool}_* \to A_*$ sending tt to a_0 and ff to a, likewise $[b]_* \in \mathsf{Bool}_* \to B_*$ sending tt to b_0 and ff to b. Abstract a fresh bridge variable $\boldsymbol{x} : \mathbf{I}$. We abbreviate $G^a_* \coloneqq \mathsf{Gr}_{\boldsymbol{x}}(\mathsf{Bool}_*, A_*, [a]_*)$ and $G^b_* \coloneqq \mathsf{Gr}_{\boldsymbol{x}}(\mathsf{Bool}_*, B_*, [b]_*)$. Applying F at G^a_* and G^b_*, we have the following.

$$F\,G^a_*\,G^b_*\,(\mathsf{gel}_{\boldsymbol{x}}(\mathsf{ff}, a, \lambda^{\mathbb{I}} \_.\, a))\,(\mathsf{gel}_{\boldsymbol{x}}(\mathsf{ff}, b, \lambda^{\mathbb{I}} \_.\, b)) \in G^a_* \wedge G^b_*$$

At $\boldsymbol{x} = \mathbf{0}$, this term is $F\,\mathsf{Bool}_*\,\mathsf{Bool}_*\,\mathsf{ff}\,\mathsf{ff}$, while at $\boldsymbol{x} = \mathbf{1}$ it is $F\,A_*\,B_*\,a\,b$. Now we apply the Graph Lemma to obtain a term in $\mathsf{Gr}_{\boldsymbol{x}}(\mathsf{Bool}_* \wedge \mathsf{Bool}_*, A_* \wedge B_*, [a]_* \wedge [b]_*)$ with the same boundary. Finally, we apply ungel to extract a path from $([a]_* \wedge [b]_*)(F\,\mathsf{Bool}_*\,\mathsf{Bool}_*\,\mathsf{ff}\,\mathsf{ff})$ to $F\,A_*\,B_*\,a\,b$. We thereby conclude that F is the pairing function if $F\,\mathsf{Bool}_*\,\mathsf{Bool}_*\,\mathsf{ff}\,\mathsf{ff}$ is $\langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle$ and the constant function if it is $\langle\!\langle \mathsf{tt}, \mathsf{tt} \rangle\!\rangle$; by Lemma 10.5.7, we are in one of these two cases. □

Corollary 10.5.10. $(A_*, B_* : \mathsf{U}_*) \to A \to B \to A_* \wedge_* B_*$ is a *set*, which is to say that every path type in this type is a proposition.

Proof. Lemma 10.5.9 shows that the type is isomorphic to Bool, which is a set. □

This is everything we need to prove the final result.

Proof (of Theorem 10.5.2). Let $F_* \in (A_*, B_* : \mathsf{U}_*) \to A_* \wedge_* B_* \to A_* \wedge_* B_*$ be given. To characterize F_*, we need to characterize its behavior on each constructor of $A_* \wedge B_*$ as well as the proof that it preserves the basepoint of $A_* \wedge_* B_*$.

First, by Lemma 10.5.9, we know that $\lambda a.\, \lambda b.\, F\,A_*\,B_*\,\langle\!\langle a, b \rangle\!\rangle$ is either pairing or constant. The values of $F\,A_*\,B_*\,\circledast^{\mathsf{L}}$ and $F\,A_*\,B_*\,\circledast^{\mathsf{R}}$ must be path-equal to $\circledast^{\mathsf{L}}$ and $\circledast^{\mathsf{R}}$ respectively, as F is basepoint-preserving and $\circledast^{\mathsf{L}}$ ($\circledast^{\mathsf{R}}$) is connected to the basepoint by $\mathsf{spoke}^{\mathsf{L}}(b_0, -)$ ($\mathsf{spoke}^{\mathsf{R}}(a_0, -)$).

Next, observe that we can capture the behavior of F on $\mathsf{spoke}^{\mathsf{L}}$ by the following term, which is a path in $(A_*, B_* : \mathsf{U}_*) \to A \to B \to A_* \wedge_* B_*$ between $\lambda A_*.\, \lambda B_*.\, \lambda \_.\, \lambda \_.\, F\, A_*\, B_* \circledast^{\mathsf{L}}$ and $\lambda A_*.\, \lambda B_*.\, \lambda \_.\, \lambda b.\, F\, A_*\, B_*\, \langle\!\langle a, b \rangle\!\rangle$.

$$\lambda^{\mathbb{I}} y.\, \lambda A_*.\, \lambda B_*.\, \lambda \_.\, \lambda b.\, F\, A_*\, B_*\, (\mathsf{spoke}^{\mathsf{L}}(b, y))$$

By Corollary 10.5.10, this path is path-equal to any other path in this type, in particular path-equal to whatever we need it to be to complete this proof. The same applies to $\circledast^{\mathsf{R}}$. Finally, we can apply the same trick for the basepoint path, writing it as a path in the type from Corollary 10.5.10 as follows.

$$\lambda^{\mathbb{I}} y.\, \lambda A_*.\, \lambda B_*.\, \lambda \_.\, \lambda \_.\, f_0\, A_*\, B_*\, y$$ □

Now we argue that this strategy can be used to prove the n-ary generalization in a uniform way. (The binary version is in fact not very useful on its own; the direct proof of commutativity for the smash product is uncharacteristically straightforward because the definition of $\wedge$ is completely symmetric.)

Theorem 10.5.11. Any function of the form (∗) is either the polymorphic identity or the polymorphic constant pointed function.

Proof. First, consider the case where we associate to the left everywhere in $(*)$. We show by induction on $i \leq n + 1$ that any

$$A_0 \to \cdots \to A_{n-i} \to (A_{(n-i+1)*} \wedge_* \cdots \wedge_* A_{n*}) \to (A_{0*} \wedge_* \cdots \wedge_* A_{n*})$$

polymorphic in $A_{0*}, \ldots, A_{n*} : \mathsf{U}_*$ is either given by iterated pairing or constant. For $i = 0$, it follows from a simple n-ary generalization of the workhorse lemma (instantiating each type argument with a graph and applying the binary Graph Lemma repeatedly). For $i > 0$, it follows from the induction hypothesis by the same argument as in the proof of Theorem 10.5.2.

The case where we associate to the right everywhere then follows from commutativity of the smash product. These two cases are sufficient to prove associativity, from which the theorem follows for all other associations. □

The key here is that we are never involved in an iterated induction on smash products: for each i in the proof of Theorem 10.5.11, we have an argument by induction on one occurrence of the smash product, but these arguments do not overlap.

Chapter 11

Formalism and models

To extend the cubical formalism sketched in Section 3.3 to include parametricity primitives, the essential task is to develop an algebraic equivalent of the interval restriction operator $- \setminus \boldsymbol{r}$, which is necessary to capture the affine quality of bridge interval variables. In the type theory of Chapter 9, $- \setminus \boldsymbol{r}$ is defined as an operator on raw contexts; like substitution as an operator on raw terms, this must be avoided in an algebraic formalism.

In the following, we therefore develop a novel formulation that regards $- \setminus \boldsymbol{r}$ as a primitive context former, characterized by an adjoint relationship with context extension by an interval variable. We note that this issue is not addressed in Bernardy, Coquand, and Moulin's account of internal parametricity [BCM15]. In the formalism they present, rules that we would express using restriction are expressed by including interval variables in the context of the conclusion as in the following rule for bridge elimination.

$$\frac{\Gamma.\mathbf{I} \vdash A \text{ type} \qquad \Gamma \vdash M_0 : A[\mathbf{0}_{\mathbf{I}}] \qquad \Gamma \vdash M_1 : A[\mathbf{1}_{\mathbf{I}}] \qquad \Gamma \vdash P : \mathsf{Bridge}(A, M_0, M_1)}{\Gamma.\mathbf{I} \vdash P\, \mathsf{v}_{\mathbf{I}} : A}$$

While this does ensure bridge variables are only used affinely, the calculus fails to satisfy cut elimination, which in the algebraic setting means that it is not always possible to reduce away an explicit substitution. In this case, there is no way to reduce $(P\, \mathsf{v}_{\mathbf{I}})[\gamma]$ given $\Gamma' \vdash \gamma : \Gamma.\mathbf{I}$. In our calculus, by contrast, this term can reduce to $P[\gamma \setminus \mathsf{v}_{\mathbf{I}}]\, \mathsf{v}_{\mathbf{I}}[\gamma]$, using the functorial action of interval restriction on substitutions.

In addition to the judgments of Section 3.3, we now have judgments for well-formedness and equality of bridge interval variables.

Judgment	Presuppositions	Reading
$\Gamma \vdash \boldsymbol{r} : \mathbf{I}$	(Γ ctx)	$\boldsymbol{r}$ is a bridge interval variable
$\Gamma \vdash \boldsymbol{r} = \boldsymbol{r}' : \mathbf{I}$	($\Gamma \vdash \boldsymbol{r}, \boldsymbol{r}' : \mathbf{I}$)	$\boldsymbol{r}$ and $\boldsymbol{r}'$ are equal bridge interval variables

Like path intervals, we may add a bridge interval to the context, in which case we have a variable interval term.

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma.\mathbf{I}\ \mathsf{ctx}} \qquad \frac{\Gamma\ \mathsf{ctx}}{\Gamma.\mathbf{I} \vdash \mathsf{v}_{\mathbf{I}} : \mathbf{I}} \qquad \frac{\Gamma \vdash r : \mathbf{I} \qquad \Gamma' \vdash \gamma : \Gamma}{\Gamma' \vdash \boldsymbol{r}[\gamma] : \mathbf{I}}$$

Where the context $\Gamma.\mathbb{I}$ is characterized as the cartesian product of Γ with $\cdot.\mathbb{I}$, however, here we need the extension to behave as a *separated* product. To express this, we introduce a new context former for interval restriction.

$$\frac{\Gamma\ \mathsf{ctx} \qquad \Gamma \vdash \boldsymbol{r} : \mathbf{I}}{\Gamma.\backslash\boldsymbol{r}\ \mathsf{ctx}} \qquad \frac{\Gamma' \vdash \boldsymbol{r} : \mathbf{I} \qquad \Gamma'.\backslash\boldsymbol{r} \vdash \gamma : \Gamma}{\Gamma' \vdash \gamma.\boldsymbol{r} : \Gamma.\mathbf{I}}$$

A substitution from Γ' into some $\Gamma.\mathbf{I}$ is therefore composed of an interval term $\Gamma' \vdash \boldsymbol{r} : \mathbf{I}$ paired with a substitution $\Gamma'.\backslash\boldsymbol{r} \vdash \delta : \Gamma$, an instantiation of Γ which "does not use" $\boldsymbol{r}$. (At this point the intuition of "use" becomes more intuition than reality; in the formalism and computational interpretation, it is indeed impossible to access an interval variable from behind the restriction, but the meaning of "use" is less obvious in non-syntactic models such as the upcoming presheaf interpretation.)

Moreover, we make this principle invertible: given $\Gamma' \vdash \gamma : \Gamma.\mathbf{I}$, there is an underlying substitution into Γ that does not use the term $\Gamma' \vdash \mathsf{v}_{\mathbf{I}}[\gamma] : \mathbf{I}$ substituted for $\mathbf{I}$. We write $\gamma^\dagger$ for this substitution.

$$\frac{\Gamma' \vdash \gamma : \Gamma.\mathbf{I}}{\Gamma'.\backslash\mathsf{v}_{\mathbf{I}}[\gamma] \vdash \gamma^\dagger : \Gamma} \qquad \frac{\Gamma\ \mathsf{ctx} \qquad \Gamma' \vdash \gamma : \Gamma.\mathbf{I}}{\Gamma' \vdash \gamma = \gamma^\dagger.\mathsf{v}_{\mathbf{I}}[\gamma] : \Gamma.\mathbf{I}} \qquad \frac{\Gamma' \vdash \boldsymbol{r} : \mathbf{I} \qquad \Gamma'.\backslash\boldsymbol{r} \vdash \gamma : \Gamma}{\Gamma'.\backslash\boldsymbol{r} \vdash \gamma = (\gamma.\boldsymbol{r})^\dagger : \Gamma}$$

This sets up an adjunction between the category of contexts sliced over the bridge interval and the category of contexts. An object of said slice category is a pair $(\Gamma',\boldsymbol{r})$ consisting of a context Γ' and term $\Gamma' \vdash \boldsymbol{r} : \mathbf{I}$. Given such an object and a second context Γ, we have a correspondence between substitutions $\Gamma'.\backslash\boldsymbol{r} \vdash \gamma : \Gamma$ and substitutions $\Gamma' \vdash \gamma' : \Gamma.\mathbf{I}$ with the property that $\Gamma' \vdash \mathsf{v}_{\mathbf{I}}[\gamma'] = \boldsymbol{r} : \mathbf{I}$, instrumented by the $-.\boldsymbol{r}$ and $-^\dagger$ substitution formers. Note that we can also derive functorial actions of extension and restriction using said operators.

$$\frac{\Gamma' \vdash \gamma : \Gamma}{\Gamma'.\mathbf{I} \vdash \gamma^{\mathbf{I}} \coloneqq (\gamma \circ \mathsf{id}^\dagger).\mathsf{v}_{\mathbf{I}} : \Gamma.\mathbf{I}} \qquad \frac{\Gamma' \vdash \gamma : \Gamma \qquad \Gamma \vdash \boldsymbol{r} : \mathbf{I}}{\Gamma'.\backslash\boldsymbol{r}[\gamma] \vdash (\gamma \setminus \boldsymbol{r}) \coloneqq ((\mathsf{id}.\boldsymbol{r}) \circ \gamma)^\dagger : \Gamma.\backslash\boldsymbol{r}}$$

We make the correspondence into a genuine adjunction by additionally imposing natural-

ity equations.

$$\frac{\Gamma'' \vdash \boldsymbol{r} : \mathbf{I} \qquad \Gamma''.\backslash \boldsymbol{r} \vdash \gamma' : \Gamma' \qquad \Gamma' \vdash \gamma : \Gamma}{\Gamma'' \vdash (\gamma \circ \gamma').\boldsymbol{r} = \gamma^{\mathbf{I}} \circ (\gamma'.\boldsymbol{r}) : \Gamma.\mathbf{I}}$$

$$\frac{\Gamma'' \vdash \gamma' : \Gamma' \qquad \Gamma' \vdash \gamma : \Gamma.\mathbf{I}}{\Gamma''.\backslash \mathsf{v}_{\mathbf{I}}[\gamma \circ \gamma'] \vdash (\gamma \circ \gamma')^{\dagger} = \gamma^{\dagger} \circ (\gamma' \setminus \mathsf{v}_{\mathbf{I}}[\gamma]) : \Gamma}$$

Finally, we include the two interval constants and the additional structural rules available to bridge interval variables: weakening and exchange as well as exchange with path interval variables.

$$\frac{}{\Gamma \vdash \mathbf{0}_{\mathbf{I}} : \Gamma.\mathbf{I}} \qquad \frac{}{\Gamma \vdash \mathbf{1}_{\mathbf{I}} : \Gamma.\mathbf{I}} \qquad \frac{}{\Gamma.\mathbf{I} \vdash \mathsf{p}_{\mathbf{I}} : \Gamma} \qquad \frac{\Gamma \text{ ctx}}{\Gamma.\mathbf{I}.\mathbf{I} \vdash \mathsf{ex}_{\mathbf{II}} : \Gamma.\mathbf{I}.\mathbf{I}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma.\mathbf{I}.\mathbb{I} \vdash \mathsf{ex}_{\mathbb{I}\mathbf{I}} : \Gamma.\mathbb{I}.\mathbf{I}}$$

The constant interval *terms* are then obtained as $\Gamma \vdash \mathsf{v}_{\mathbf{I}}[\mathbf{0}_{\mathbf{I}}] : \mathbf{I}$ and $\Gamma \vdash \mathsf{v}_{\mathbf{I}}[\mathbf{1}_{\mathbf{I}}] : \mathbf{I}$. We deliberately introduce the constants as substitutions rather than as terms, as the former is stronger than the latter: given some $\Gamma \vdash \boldsymbol{r} : \mathbf{I}$, we can only construct a substitution $\Gamma \vdash \mathsf{id}.\boldsymbol{r} : \Gamma.\backslash \boldsymbol{r}.\mathbf{I}$, not a substitution from Γ to $\Gamma.\mathbf{I}$. Using $\Gamma \vdash \boldsymbol{\varepsilon}_{\mathbf{I}} : \Gamma.\mathbf{I}$, on the other hand, we are able to access hypotheses beneath restriction by a constant: $\Gamma.\backslash \mathsf{v}_{\mathbf{I}}[\boldsymbol{\varepsilon}_{\mathbf{I}}] \vdash \boldsymbol{\varepsilon}_{\mathbf{I}}{}^{\dagger} : \Gamma$.

We require the structural and endpoint substitutions to satisfy various unsurprising equations, expressing their naturality and interactions with each other. (We refer to Grandis and Mauri [GM03] for more detailed analysis of the equations generating various categories of cubical sets.)

$$\frac{\varepsilon \in \{0,1\} \qquad \Gamma' \vdash \gamma : \Gamma}{\Gamma' \vdash \gamma^{\mathbf{I}} \circ \boldsymbol{\varepsilon}_{\mathbf{I}} = \boldsymbol{\varepsilon}_{\mathbf{I}} \circ \gamma : \Gamma.\mathbf{I}} \qquad \frac{\Gamma' \vdash \gamma : \Gamma}{\Gamma'.\mathbf{I} \vdash \gamma \circ \mathsf{p}_{\mathbf{I}} = \mathsf{p}_{\mathbf{I}} \circ \gamma^{\mathbf{I}} : \Gamma}$$

$$\frac{\Gamma' \vdash \gamma : \Gamma}{\Gamma'.\mathbf{I}.\mathbf{I} \vdash \gamma^{\mathbf{II}} \circ \mathsf{ex}_{\mathbf{II}} = \mathsf{ex}_{\mathbf{II}} \circ \gamma^{\mathbf{II}} : \Gamma.\mathbf{I}.\mathbf{I}} \qquad \frac{\varepsilon \in \{0,1\}}{\Gamma \vdash \mathsf{p}_{\mathbf{I}} \circ \boldsymbol{\varepsilon}_{\mathbf{I}} = \mathsf{id} : \Gamma} \qquad \frac{\Gamma \text{ ctx}}{\Gamma.\mathbf{I}.\mathbf{I} \vdash \mathsf{p}_{\mathbf{I}} \circ \mathsf{ex}_{\mathbf{II}} = \mathsf{p}_{\mathbf{I}}{}^{\mathbf{I}} : \Gamma.\mathbf{I}}$$

$$\frac{\Gamma \text{ ctx}}{\Gamma.\mathbf{I}.\mathbf{I}.\mathbf{I} \vdash \mathsf{ex}_{\mathbf{II}}{}^{\mathbf{I}} \circ \mathsf{ex}_{\mathbf{II}} \circ \mathsf{ex}_{\mathbf{II}}{}^{\mathbf{I}} = \mathsf{ex}_{\mathbf{II}} \circ \mathsf{ex}_{\mathbf{II}}{}^{\mathbf{I}} \circ \mathsf{ex}_{\mathbf{II}} : \Gamma.\mathbf{I}.\mathbf{I}.\mathbf{I}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma.\mathbf{I}.\mathbb{I} \vdash (\mathsf{p}_{\mathbb{I}}{}^{\mathbf{I}}.\mathsf{v}_{\mathbb{I}}) \circ \mathsf{ex}_{\mathbb{I}\mathbf{I}} = \mathsf{id} : \Gamma.\mathbf{I}.\mathbb{I}}$$

$$\frac{\Gamma \text{ ctx}}{\Gamma.\mathbb{I}.\mathbf{I} \vdash \mathsf{ex}_{\mathbb{I}\mathbf{I}} \circ (\mathsf{p}_{\mathbb{I}}{}^{\mathbf{I}}.\mathsf{v}_{\mathbb{I}}) = \mathsf{id} : \Gamma.\mathbb{I}.\mathbf{I}}$$

This judgmental structure is sufficient to express the typing rules for the bridge and Gel types as well as the extent operator. We display rules for bridge and Gel types in Figures 11.1 and 11.2 respectively. (Expressing the rules for extent without named variables is sufficiently painful that we leave this as an exercise to the reader.)

$$\frac{\Gamma.\mathbf{I} \vdash A\ \mathsf{type} \qquad \Gamma \vdash M_0 : A[\mathbf{0}_\mathbf{I}] \qquad \Gamma \vdash M_1 : A[\mathbf{1}_\mathbf{I}]}{\Gamma \vdash \mathsf{Bridge}(A, M_0, M_1)\ \mathsf{type}}$$

$$\frac{\Gamma.\mathbf{I} \vdash A\ \mathsf{type} \qquad \Gamma.\mathbf{I} \vdash M : A}{\Gamma \vdash \lambda^{\mathbf{I}}(M) : \mathsf{Bridge}(A, M[\mathbf{0}_\mathbf{I}], M[\mathbf{1}_\mathbf{I}])}$$

$$\frac{\begin{array}{c}\Gamma \vdash \boldsymbol{r} : \mathbf{I} \qquad \Gamma.\backslash\boldsymbol{r}.\mathbf{I} \vdash A\ \mathsf{type} \\ \Gamma.\backslash\boldsymbol{r} \vdash M_0 : A[\mathbf{0}_\mathbf{I}] \qquad \Gamma.\backslash\boldsymbol{r} \vdash M_1 : A[\mathbf{1}_\mathbf{I}] \qquad \Gamma.\backslash\boldsymbol{r} \vdash P : \mathsf{Bridge}(A, M_0, M_1)\end{array}}{\Gamma \vdash P\, r : A[\mathsf{id}.\boldsymbol{r}]}$$

$$\frac{\begin{array}{c}\varepsilon \in \{0,1\} \\ \Gamma.\mathbf{I} \vdash A\ \mathsf{type} \qquad \Gamma \vdash M_0 : A[\mathbf{0}_\mathbf{I}] \qquad \Gamma \vdash M_1 : A[\mathbf{1}_\mathbf{I}] \qquad \Gamma \vdash P : \mathsf{Bridge}(A, M_0, M_1)\end{array}}{\Gamma \vdash P[\boldsymbol{\varepsilon}_\mathbf{I}^{\dagger}]\ \mathsf{v}_\mathbf{I}[\boldsymbol{\varepsilon}_\mathbf{I}] = M_\varepsilon : A[\boldsymbol{\varepsilon}_\mathbf{I}]}$$

$$\frac{\Gamma \vdash \boldsymbol{r} : \mathbf{I} \qquad \Gamma.\backslash\boldsymbol{r}.\mathbf{I} \vdash A\ \mathsf{type} \qquad \Gamma.\backslash\boldsymbol{r}.\mathbf{I} \vdash M : A}{\Gamma \vdash \lambda(M)\, r = M[\mathsf{id}.\boldsymbol{r}] : A[\mathsf{id}.\boldsymbol{r}]}$$

$$\frac{\Gamma.\mathbf{I} \vdash A\ \mathsf{type} \qquad \Gamma \vdash M_0 : A[\mathbf{0}_\mathbf{I}] \qquad \Gamma \vdash M_1 : A[\mathbf{1}_\mathbf{I}] \qquad \Gamma \vdash P : \mathsf{Bridge}(A, M_0, M_1)}{\Gamma \vdash P = \lambda^{\mathbf{I}}(P[\mathsf{id}^{\dagger}]\ \mathsf{v}_\mathbf{I}) : \mathsf{Bridge}(A, M_0, M_1)}$$

Figure 11.1: Rules for bridge types in a parametric type theory formalism

11.1 Bicubical set model

We can build a non-computational model in the category of Kan presheaves following the pattern established in Section 3.3.1. This time around, our presheaves are over the interval contexts of parametric cubical type theory, *i.e.*, contexts of bridge and path interval variables.

Definition 11.1.1. The *cartesian-affine bicube category* $\square_{c\times a}$ is the category whose objects are interval contexts Ψ ictx of parametric cubical type theory and whose morphisms $\psi \in \square_c[\Psi', \Psi]$ from Ψ' to Ψ are interval substitutions $\Psi' \Vdash \psi \in \Psi$.

Because we have exchange between path and bridge interval variables, $\square_{c\times a}$ is equivalent to the product $\square_c \times \square_a$ of the cartesian cube category $\square_c$ from Section 3.3.1 and the category $\square_a$ of bridge variables and substitutions, but we do not need this fact here.

Within the presheaf category $\mathit{PSh}(\square_{c\times a})$, we have two interval objects provided by the Yoneda embedding: the path interval $\mathbb{I} \coloneqq \text{ﾑ}(x{:}\mathbb{I})$ is joined by a bridge interval $\mathbf{I} \coloneqq \text{ﾑ}(\boldsymbol{x}{:}\mathbf{I})$.

$$\frac{\Gamma \vdash \boldsymbol{r} : \mathbf{I} \qquad \Gamma.\backslash \boldsymbol{r} \vdash A_0 \text{ type} \qquad \Gamma.\backslash \boldsymbol{r} \vdash A_1 \text{ type} \qquad \Gamma.\backslash \boldsymbol{r}.A_0.A_1[\mathsf{p}] \vdash R \text{ type}}{\Gamma \vdash \mathsf{Gel}_r(A_0, A_1, R) \text{ type}}$$

$$\frac{\varepsilon \in \{0,1\} \qquad \Gamma \vdash A_0 \text{ type} \qquad \Gamma \vdash A_1 \text{ type} \qquad \Gamma.A_0.A_1[\mathsf{p}] \vdash R \text{ type}}{\Gamma \vdash \mathsf{Gel}_\varepsilon(A_0[{\boldsymbol{\varepsilon}_{\mathbf{I}}}^\dagger], A_1[{\boldsymbol{\varepsilon}_{\mathbf{I}}}^\dagger], R[{\boldsymbol{\varepsilon}_{\mathbf{I}}}^{\dagger\times\times}]) = A_\varepsilon \text{ type}}$$

$$\frac{\begin{array}{c}\Gamma \vdash \boldsymbol{r} : \mathbf{I} \qquad \Gamma.\backslash \boldsymbol{r} \vdash M_0 : A_0 \\ \Gamma.\backslash \boldsymbol{r} \vdash M_1 : A_1 \qquad \Gamma.\backslash \boldsymbol{r}.A_0.A_1[\mathsf{p}] \vdash R \text{ type} \qquad \Gamma.\backslash \boldsymbol{r} \vdash P : R[\mathsf{id}.M_0.M_1]\end{array}}{\Gamma \vdash \mathsf{gel}_r(M_0, M_1, P) : \mathsf{Gel}_r(A_0, A_1, R)}$$

$$\frac{\begin{array}{c}\varepsilon \in \{0,1\} \\ \Gamma \vdash M_0 : A_0 \qquad \Gamma \vdash M_1 : A_1 \qquad \Gamma.A_0.A_1[\mathsf{p}] \vdash R \text{ type} \qquad \Gamma \vdash P : R[\mathsf{id}.M_0.M_1]\end{array}}{\Gamma \vdash \mathsf{gel}_\varepsilon(M_0[{\boldsymbol{\varepsilon}_{\mathbf{I}}}^\dagger], M_1[{\boldsymbol{\varepsilon}_{\mathbf{I}}}^\dagger], P[{\boldsymbol{\varepsilon}_{\mathbf{I}}}^\dagger]) = M_\varepsilon : A_\varepsilon}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_0 \text{ type} \\ \Gamma \vdash A_1 \text{ type} \qquad \Gamma.A_0.A_1[\mathsf{p}] \vdash R \text{ type} \qquad \Gamma.\mathbf{I} \vdash Q : \mathsf{Gel}_{\mathsf{v}_\mathbf{I}}(A_0[\mathsf{id}^\dagger], A_1[\mathsf{id}^\dagger], R[\mathsf{id}^{\dagger\times\times}])\end{array}}{\Gamma \vdash \mathsf{ungel}(Q) : R[\mathsf{id}.Q[\mathbf{0}_\mathbf{I}].Q[\mathbf{1}_\mathbf{I}]]}$$

$$\frac{\Gamma \vdash M_0 : A_0 \qquad \Gamma \vdash M_1 : A_1 \qquad \Gamma.A_0.A_1[\mathsf{p}] \vdash R \text{ type} \qquad \Gamma \vdash P : R[\mathsf{id}.M_0.M_1]}{\Gamma \vdash \mathsf{ungel}(\mathsf{gel}_{\mathsf{v}_\mathbf{I}}(M_0[\mathsf{id}^\dagger], M_1[\mathsf{id}^\dagger], P[\mathsf{id}^\dagger])) = P : R[\mathsf{id}.M_0.M_1]}$$

$$\frac{\begin{array}{c}\Gamma \vdash \boldsymbol{r} : \mathbf{I} \qquad \Gamma.\backslash \boldsymbol{r} \vdash A_0 \text{ type} \qquad \Gamma.\backslash \boldsymbol{r} \vdash A_1 \text{ type} \\ \Gamma.\backslash \boldsymbol{r}.A_0.A_1[\mathsf{p}] \vdash R \text{ type} \qquad \Gamma.\backslash \boldsymbol{r}.\mathbf{I} \vdash Q : \mathsf{Gel}_{\mathsf{v}_\mathbf{I}}(A_0[\mathsf{id}^\dagger], A_1[\mathsf{id}^\dagger], R[\mathsf{id}^{\dagger\times\times}])\end{array}}{\Gamma \vdash Q[\mathsf{id}.\boldsymbol{r}] = \mathsf{gel}_r(Q[\mathbf{0}_\mathbf{I}], Q[\mathbf{1}_\mathbf{I}], \mathsf{ungel}(Q)) : \mathsf{Gel}_r(A_0, A_1, R)}$$

Figure 11.2: Rules for Gel types in a parametric type theory formalism

We can repeat the constructions from Section 3.3.1 to define the interpretations of the judgments as well as the context, substitution, type, and term formers inherited from cubical type theory.

To interpret the parametric constructs, the first step is to identify two functors between the bicube category $\square_{c\times a}$ and its slice $\square_{c\times a}/(\boldsymbol{x} : \mathbf{I})$ over the interval context with a single bridge variable, which we henceforth abbreviate as $\square_{c\times a}/\mathbf{I}$. Objects of the latter category are pairs $(\Psi, \boldsymbol{r})$ of interval contexts Ψ ictx equipped with a distinguished $\Psi \Vdash \boldsymbol{r} \in \mathbf{I}$, while morphisms $\psi \in (\square_{c\times a}/\mathbf{I})[(\Psi', \boldsymbol{r}'), (\Psi, \boldsymbol{r})]$ are substitutions $\Psi' \Vdash \psi \in \Psi$ such that $\Psi' \Vdash \boldsymbol{r}\psi = \boldsymbol{r}' \in \mathbf{I}$. We have a functor $(-) \otimes \mathbf{I} : \square_{c\times a} \to \square_{c\times a}/\mathbf{I}$, *extension by a bridge interval*, defined on objects and morphisms as follows.

$$\Psi \otimes \mathbf{I} \coloneqq ((\Psi, \boldsymbol{x} : \mathbf{I}), \boldsymbol{x})$$
$$\psi \otimes \mathbf{I} \coloneqq (\psi, \boldsymbol{x}/\boldsymbol{x})$$

That is, we take Ψ to the extended context $(\Psi, \boldsymbol{x} : \mathbf{I})$ with its canonical variable element $\Psi, \boldsymbol{x} : \mathbf{I} \Vdash \boldsymbol{x} \in \mathbf{I}$.

We also have a second *restriction* functor $Res : \square_{c\times a}/\mathbf{I} \to \square_{c\times a}$ in the opposite direction, using the functorial action of interval restriction (Lemma 9.1.11).

$$Res(\Psi, \boldsymbol{r}) \coloneqq \Psi \setminus \boldsymbol{r}$$
$$Res(\Psi' \Vdash \psi \in \Psi) \coloneqq (\psi : \Psi) \setminus \boldsymbol{r}$$

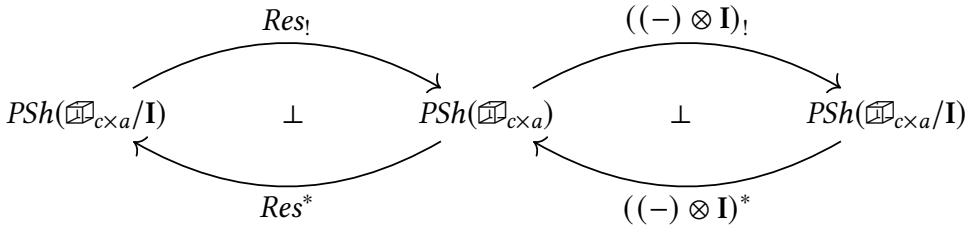
As we observed while establishing our formalism, restriction is left adjoint to extension: there is an isomorphism between $\square_{c\times a}[\Psi' \setminus \boldsymbol{r}, \Psi]$ and $(\square_{c\times a}/\mathbf{I})[(\Psi', \boldsymbol{r}), \Psi \otimes \mathbf{I}]$ for any $(\Psi', \boldsymbol{r})$ and Ψ, natural in both arguments.

A single functor $F : \mathcal{C} \to \mathcal{D}$ between index categories induces a trio of adjoint functors $F_! \dashv F^* \dashv F_*$ between the presheaf categories $PSh(\mathcal{C})$ and $PSh(\mathcal{D})$. The center functor $F^* : PSh(\mathcal{D}) \to PSh(\mathcal{C})$ is given by precomposition: $F^*(G)(c) \coloneqq G(F(c))$ for $c \in \mathcal{C}$. The left and right adjoints are given by *left* and *right Kan extension* respectively. For a thorough and general account of these we refer to [Rie14, Chapter 1]. For our purposes, we only need the left adjoint and really only need to know that it exists, but we give a description of its behavior on objects for intuition's sake. Given $G \in PSh(\mathcal{C})$ and $d \in \mathcal{D}$, we define

$$F_!(G)(d) \coloneqq \{(c, f, t) \mid c \in \mathcal{C}, f \in \mathcal{D}[d, F(c)], t \in G(c)\}/\approx$$

where $\approx$ is the equivalence relation generated by $(c, f, t) \approx (c', f', t')$ whenever there exists $g \in \mathcal{C}[c, c']$ such that $f' = F(g) \circ f$ and $G(g)(t') = t$. For us, one essential property of this definition is that it commutes with the Yoneda embedding: we have $F_!(よ(c)) \cong よ(F(c))$.

Returning to cubical sets, we thus we have two pairs of induced adjoint functors on presheaves as shown in the diagram below.



Henceforth we abbreviate $\mathbf{I}_! \coloneqq ((-)\otimes \mathbf{I})_!$ and $\mathbf{I}^* \coloneqq ((-)\otimes \mathbf{I})^*$. The fact that *Res* is left adjoint to $(-)\otimes \mathbf{I}$ moreover implies that Res^* is also left adjoint to $\mathbf{I}^*$. This implies that in fact $\mathbf{I}_! \cong Res^*$: both are left adjoint to $\mathbf{I}^*$ (or right adjoint to $Res_!$) and adjoints are uniquely determined.

Finally, the category $PSh(\square_{c\times a}/\mathbf{I})$ is equivalent to the slice category $PSh(\square_{c\times a})/\mathbf{I}$. If we thereby regard these functors as going between $PSh(\square_{c\times a})$ and $PSh(\square_{c\times a})/\mathbf{I}$, we may calculate the effect of $Res_! : PSh(\square_{c\times a})/\mathbf{I} \to PSh(\square_{c\times a})$ for $(G, g) \in PSh(\square_{c\times a})/\mathbf{I}$ and $\Psi \in \square_{c\times a}$ as follows.

$$Res_!(G,g)(\Psi) = \left\{ (\Phi, \boldsymbol{s}, \psi, t) \;\middle|\; \begin{array}{l} \Phi \text{ ictx} \\ \Phi \Vdash \boldsymbol{s} \in \mathbf{I} \\ \Psi \Vdash \psi \in \Phi \setminus \boldsymbol{s} \\ t \in P(\Phi) \\ g(\Phi)(t) = \boldsymbol{s} \end{array} \right\} /\!\approx$$

Here $\approx$ is the equivalence relation generated by $(\Phi, \boldsymbol{s}, \psi, t) \approx (\Phi', \boldsymbol{s}', \psi', t')$ whenever there is some $\Phi' \Vdash \phi \in \Phi$ such that $\Phi' \Vdash \boldsymbol{s}' = \boldsymbol{s}\phi \in \mathbf{I}$, $\Psi \Vdash \psi = ((\phi : \Phi) \setminus \boldsymbol{s})\psi' \in \Phi \setminus \boldsymbol{s}$, and $P(\phi)(t) = t'$. The functor $\mathbf{I}_!$, meanwhile, is more simply obtained as follows, reflecting its characterization as Res^*.

$$\mathbf{I}_!(G) = (G', g) \text{ where } \begin{cases} G'(\Psi) \coloneqq \sum_{\Psi \Vdash \boldsymbol{r} \in \mathbf{I}} G(\Psi \setminus \boldsymbol{r}) \\ g(\Psi)(\boldsymbol{r}, t) \coloneqq \boldsymbol{r} \end{cases}$$

With these tools in hand, we begin interpreting the parametricity elements of the formalism. We interpret bridge interval terms $\Gamma \vdash \boldsymbol{r} : \mathbf{I}$ by morphisms $[\![\boldsymbol{r}]\!] : [\![\Gamma]\!] \to \mathbf{I}$. For any semantic context G, we interpret its extension by a bridge interval by as $\pi_0(\mathbf{I}_!(G)) \in PSh(\square_{c\times a})$, with the accompanying variable projection given by $\pi_1(\mathbf{I}_!(G)) : G \to \mathbf{I}$; notice that the definition of $\mathbf{I}_!(G)$ is exactly what we would expect from context extension. From the calculation of $\mathbf{I}_!(G)$ above, it is straightforward to check that it validates the structural rules we require of the bridge interval. We likewise interpret context restriction by $Res_!$, the left adjoint to $\mathbf{I}_!$.

What remains is to interpret the various type and term formers. We will not go through these explicitly, but rather observe that the proofs of the rules in our computational interpretation can be mechanically adapted. The key here is that, like the computational interpretation, types and terms in the presheaf interpretation are defined by their behavior after "closing" substitutions. Recall, for example, that a semantic pretype over a cubical set G is a family of sets $T(\Psi, g)$ indexed by pairs of $\Psi \in \square_c$ and $g \in G(\Psi)$ and with transition functions between them. As such, T is determined by the instances $\alpha^* T$ given by substitutions $\alpha : よ(\Psi) \to G$. This means that, to prove that the presheaf interpretation interprets the various rules, it suffices to show each holds when the conclusion is in a "closed" context. In these cases, we can exploit the aforementioned key property of the left Kan extension: we have $Res_!(よ(\Psi), \boldsymbol{r}) \cong (\Psi \setminus \boldsymbol{r})$ and $\mathbf{I}_!(よ(\Psi)) \cong (\Psi, \boldsymbol{x} : \mathbf{I})$. Thus the statements of the rules on closed contexts are more or less the same in the computational interpretation and presheaf semantics.

Chapter 12

Conclusions

12.1 Related work

Internal parametricity The concept of parametricity originates with Reynolds [Rey83], who gave a relational interpretation of simply-typed λ-calculus with type variables in order to show that polymorphic functions treat their type arguments parametrically. His vision of parametricity is external and semantic: the results that follow from parametricity are theorems about the denotation of terms in a set-theoretic model. This kind of parametricity has been extended in every which direction—mostly notably for our purposes, to dependent type theory, by Atkey, Ghani, and Johann [AGJ14].

Mairson [Mai91], as well as Abadi, Cardellin, Curien, and Lévy [ACC93] and Plotkin and Abadi [PA93], developed early *syntactic* accounts of parametricity. In these systems, one has a logic on top of a type-theoretic formalism (typically the impredicative polymorphic λ-calculus) in which parametricity properties can be derived. The relational logic can then be interpreted in some setting such as Reynolds' (modulo issues of impredicativity).

Bernardy and Lasson [BL11] observed more generally that, given a pure type system (PTS) [Bar91], one can find a new, possibly stronger PTS in which the relational interpretation of the former system can be defined. Bernardy, Jansson, and Paterson [BJP10] show that in a sufficiently expressive, so-called *reflective* PTS, such as a dependent type theory, the relational interpretation can be defined in the same PTS. This is a step towards fully internal parametricity: the inputs and outputs of the parametricity translation belong to the same theory, but the translation function itself is metatheoretical. Keller and Lasson [KL12b] proved a similar result, constructing—and implementing as a tactic in the **Coq** proof assistant [Coq]—a parametricity translation from types to elements of an impredicative universe of propositions.

Krishnaswami and Dreyer [KD13], meanwhile, define a relational realizability semantics of a formalism for extensional dependent type theory that validates parametricity

theorems. They observe that the consequences of parametricity may be added as axioms to the theory without disrupting its computational character, thereby internalizing parametricity. Their relations are contentless; our own computational interpretation may, to some extent, be seen as a contentful reimagining of their semantics.

True internal parametricity in our sense was introduced by Bernardy and Moulin [BM12], who extended dependent type theory with internal operators $- \in [\![-]\!]$ and $[\![-]\!]$ computing the relational interpretations of types and terms respectively. Their formalism is moreover adequate from a computational perspective. This earliest foray into internal parametricity was substantially complicated by the higher-dimensional structure of iterated parametricity—the need to define the parametricity interpretation of $- \in [\![-]\!]$ and $[\![-]\!]$ themselves—and included operations for permuting the order of iterated parametricity applications and "hypercube" syntax. In later work, influenced by cubical type theory, these elements were replaced by interval variables [BM13; BCM15].

Our own parametric type theory is inspired directly from the formalism and refined presheaf interpretation for internal parametricity defined by Bernardy, Coquand, and Moulin (BCM) [BCM15], also described with slight differences in Moulin's dissertation [Mou16]. We enrich the theory by replacing the underlying ITT with a cubical type theory, which provides a better-behaved equality and opportunities to apply parametricity to higher-dimensional problems (as in Section 10.5). As noted in Section 11.1, the improved equality allows us to relax some equations they require, with the effect of simplifying the presheaf interpretation; they use not-quite-presheaves of *I-sets*. (Admittedly, one must in exchange deal with Kan operations and so on.) Much of the theory developed in Chapter 10, although novel, can be replicated to some extent in their theory. The inadequacies of ITT equality, however, are an ever-present irritant; for example, one cannot show that the class of bridge-discrete types is closed under function types, as ITT does not characterize the identity types of functions. A more cosmetic difference is that we use binary parametricity (based on relations) rather than unary parametricity (based on predicates).

On the formalistic level, we import Cheney's concept of name restriction, developed for a theory of nominal sets [Che12], to give rules for Bridge and Gel types that permit substitution elimination, rectifying a defect of the BCM formalism. The theory presented in [BM13] uses a system of "tainted" and "oblivious" hypotheses to enforce apartness restrictions, but is different enough from the BCM theory on the whole that it is difficult to make a comparison.

We have eschewed the BCM notation in favor of one that emphasizes the similarity with cubical type theory. To ease comparison, we provide a translation dictionary in Figure 12.1. Note that, because of the additional equations BCM impose to ensure relativity, the correspondence is not one-to-one, with the same type and term formers in their theory playing multiple roles from our perspective. In Moulin's dissertation, the notion of a function $(i : \mathbb{I}) \to A$ without a fixed endpoint (called a "ray") is included separately from bridge types, and term formers that are primitive in [BCM15] are often implemented as

This paper	[BCM15]	[Mou16]
$\mathsf{Bridge}(\boldsymbol{x}.A, a_0, a_1)$	$A \ni_{\boldsymbol{x}} a$	$(\forall \boldsymbol{x}.A) \ni a$
$\lambda^{\mathrm{I}}([])\boldsymbol{x}]a$	$a \cdot \boldsymbol{x}$	$(\langle \boldsymbol{x} \rangle a)!$
$p\,\boldsymbol{x}$	$(a,_{\boldsymbol{x}} p)$	$\langle\!\lvert a,_{\boldsymbol{x}} p \rvert\!\rangle$
$\mathsf{extent}_{\boldsymbol{x}}(-; a_0.t_0, a_1.t_1, a_0.a_1.\overline{a}.u)$	$\langle \lambda a.t,_{\boldsymbol{x}} \lambda a.\lambda \overline{a}.u \rangle$	$\langle\!\lvert \lambda a.t,_{\boldsymbol{x}} \lambda a.\lambda \overline{a}.u \rvert\!\rangle$
$\mathsf{Gel}_{\boldsymbol{x}}(A_0, A_1, a_0.a_1.R)$	$(a : A) \times_{\boldsymbol{x}} R$	$A \bowtie_{\boldsymbol{x}} R$
$\mathsf{gel}_{\boldsymbol{x}}(a_0, a_1, c)$	$(a,_{\boldsymbol{x}} c)$	$\langle\!\lvert a,_{\boldsymbol{x}} p \rvert\!\rangle$
$\mathsf{ungel}(\boldsymbol{x}.a)$	$a \cdot \boldsymbol{x}$	$(\langle \boldsymbol{x} \rangle a)!$

Figure 12.1: Translation dictionary for internal parametricity

combinations of terms relating first interval dependency to rays and then rays to bridges. In particular, $A \bowtie_{\boldsymbol{x}} R$ is syntactic sugar for a term $\langle\!\lvert A, \Psi_A R \rvert\!\rangle @ \boldsymbol{x}$, while $\langle\!\lvert f,_{\boldsymbol{x}} h \rvert\!\rangle$ is sugar for $\langle\!\lvert f, \Phi_f h \rvert\!\rangle @ \boldsymbol{x}$. As a result, equivalents of Gel and extent are sometimes called Ψ- and Φ-operators respectively in the literature.

Internal parametricity à la Nuyts et al. Nuyts, Vezzosi, and Devriese [NVD17] define a second internally parametric type theory building on Bernardy et al.'s work. Their system, **ParamDTT**, follows the BCM theory by employing intervals to express the action of terms on relations. Like our own theory, Nuyts et al.'s includes two kinds of interval, defining "bridges" and "paths", and our own use of the word "bridge" is borrowed from this word.

However, the coincidence of terminology is somewhat misleading. **ParamDTT**'s paths provide a much weaker notion of heterogeneous equality; paths are not in general required to satisfy anything like the Kan operations. The only requirement is that *homogeneous* paths give rise to identities, what Nuyts et al. call the *path degeneracy axiom*.

$$\frac{P \in \mathsf{Path}(\_.A, M_0, M_1)}{\mathsf{degax}(P) \in \mathsf{Id}(A, M_0, M_1)}$$

ParamDTT's paths are therefore closer in spirit to the heterogeneous equalities of Observational Type Theory [AMS07] than to those of cubical type theory. From our perspective, it may be more natural to think of these paths as more like a second, stronger form of bridge. Indeed, Nuyts and Devriese [ND18] have since developed a more general system that includes a tower of notions of *n-relatedness*, with **ParamDTT**'s paths and bridges as the first two levels. In order to avoid confusion with our own terminology, we henceforth

take a page from this sequel by refering to **ParamDTT**'s paths as 0-bridges and bridges as 1-bridges.

ParamDTT includes multiple function types requiring different behavior on bridges; Nuyts et al. identify *parametric* functions not as those that merely preserve n-bridges, but as those that take 1-bridges to 0-bridges. The interaction between 0- and 1-bridges is mediated by a system of modalities. In particular, variables in *type* positions are checked under a different modality than variables in *element* position. Their system therefore captures a phase distinction where term-level computation cannot depend significantly on type-level computation, an aspect of parametricity absent from our work.

The introduction of two kinds of bridge is principally motivated by the desire for an identity extension lemma. If we want to analyze a term of type $\mathsf{U} \to A$ with parametricity, we cannot have identity extension for bridges in the domain type: bridges in the universe must be given by relations, not paths. However, if we want every parametric function $\mathsf{U} \to A$ to be constant, we do need identity extension in the codomain A. This is resolved in **ParamDTT** by asking that parametric functions send 1-bridges, which do not support identity extension, to 0-bridges, which do. We take a different approach: rather than requiring any form of global identity extension lemma, we can internally identify the class of types that satisfy it. Thus not all parametric functions $\mathsf{U} \to A$ are constant, but they are if A is bridge-discrete (Lemma 10.4.2), and we can show that a large class of type formers preserve bridge-discreteness.

Another notable departure from the BCM theory is that **ParamDTT** uses *structural* variables (for both kinds of bridge), whereas we have stressed the importance of affine variables. As we have discussed in Section 9.4, proper Gel-types require an affine interval; **ParamDTT** instead uses Glue-types [CCHM15], of which V-types are a special case, and Weld-types, their dual. Recall from Section 9.4 that V-types are insufficient in our setting because degenerate type bridges need not correspond to identity relations. This is smoothed over in **ParamDTT** by the stronger requirements on parametric functions: degenerate type 0-bridges *do* correspond to identity relations. One casualty of this setup is that *iterated* parametricity becomes impossible: the function arguments to Glue and Weld types are checked under a *pointwise* modality that prevents such types from stacking. Thus the coherence functions produced by a parametricity argument are not themselves guaranteed to be parametric. This situation is improved in [ND18], where the infinite hierarchy of n-relations prevents parametricity from "running out".

Nuyts [Nuy20] has also developed a unified treatment of V, Gel, and similar types as instances of what he calls a *transpension type*. We discuss this in more detail in Section 17.1.

Higher-dimensional parametricity Outside the area of internal parametricity, higher-dimensional or contentful (or *proof-relevant*) parametricity, as well as logical relations

more generally, have been explored in a number of contexts.

Benton, Hofmann, and Nigam [BHN13; BHN14] use a proof-relevant logical relation to analyze abstract effects, making use in particular of proof-relevant existential quantification over allocations to a heap. More recently, proof-relevant logical relations have been exploited to cleanly obtain canonicity and normalization results for dependent type theories [Shu15; Coq19; CHS19; KHS19; SAG19; GKNB20]. Proof-relevant relations naturally accommodate the universes of dependent type theory, which frequently beg for proof-relevant interpretation. In the case of parametricity, for example, one wants to interpret the universe as the proof-relevant relation of relations—that is, one wants "$\mathsf{Bridge}(\mathsf{U}, A, B) \simeq (A \times B \to \mathsf{U})$". Sterling and Harper [SH20] use proof-relevant (and syntactic) parametricity to obtain an abstraction theorem for a program module calculus; proof-relevance becomes critical because modules can contain not only terms but types, which again have a naturally contentful parametricity interpretation.

Higher-dimensionally parametric models of the impredicative polymorphic λ-calculus have been explicitly explored by Ghani, Nordvall Forsberg, and Orsanigo [GNO16] as well as Sojakova and Johann [SJ18]. Johann and Sojakova have moreover defined a notion of n-dimensionally parametric model for $n \leq \infty$ based on cubical sets [JS17].

Directed type theory Riehl and Shulman's *directed type theory* formalism [RS17], and its fibrant presheaf model in particular, bears a strong resemblance to parametric cubical type theory. Like our theory, it combines higher-dimensional equality structure (here cubical, there simplicial) with a second layer of relational structure. In their work, the objective is to identify the types whose relational structure is ∞-categorical, *i.e.*, supports composition of morphisms in an appropriate sense, enabling the use of the theory as a language for synthetic higher category theory. However, the theory itself allows arbitrarily relational structure, in order to avoid involving issues of variance at the judgmental level.

It was initially suspected that this model would contain a universe satisfying what we call relativity, but Cavallo, Riehl, and Sattler later found that this was not the case [Rie18]. Our comparative analysis of Gel and V in Section 9.4 provides some intuition for this failure: relativity relies on the peculiar structure of the affine cube category, failing in more "structural" settings like cartesian cubical and simplicial sets. Subsequent work on this flavor of directed type theory has focused on instead constructing a *covariant universe* in which morphisms/bridges correspond to *functors* [Rie18]; Weaver and Licata have developed a structural cubical model of directed type theory containing such a universe [WL20].

One could try developing a version of directed type theory based on affine cubical sets in order to obtain a relativistic universe, but is unlikely that the concept of $(\infty, 1)$-category theory it produced would be equivalent to the classical one. Sattler has shown that the BCH model, as a setting for homotopy theory (*i.e.*, Quillen model category), is

not equivalent to the classical model in spaces. It is unclear whether one can in some way get "the best of both worlds": a relational setting that contains a relativistic universe but becomes equivalent to a classical setting when restricted to $(\infty, 1)$-categories or $(\infty, 1)$-groupoids.

Substructural cubes Our parametric type theory, following Bernardy et al., adopts the affine cubical structure used in Bezem, Coquand and Huber's cubical model of ITT with the univalence axiom. This model has been largely abandoned in favor of structural cubical type theories, in part because of the comparative intuitive simplicity of structural variables, but also due to the difficulty of interpreting higher inductive types in this model.

To get an intuitive sense of the problem, consider the following "interval" higher inductive type, consisting of two points with a path between them.

$$\begin{array}{l}\textbf{inductive}\ \mathsf{Ival}\ \textbf{where}\\ \mid \mathsf{zero} \in \mathsf{Ival}\\ \mid \mathsf{one} \in \mathsf{Ival}\\ \mid \mathsf{seg}(x : \mathbb{I}) \in \mathsf{Ival} \quad [x \equiv 0 \hookrightarrow \mathsf{zero}, x \equiv 0 \hookrightarrow \mathsf{one}]\end{array}$$

We would expect an eliminator for this type validating the following rule.

$$\frac{\begin{array}{ccc} & i : \mathsf{Ival} \gg A\ \mathsf{type} & M \in \mathsf{Ival} \\ Z \in A[\mathsf{zero}/i] \qquad O \in A[\mathsf{one}/i] & & \boldsymbol{x} : \mathbf{I} \gg S \in A[\mathsf{seg}(x)/i] \end{array}}{\mathsf{elim}(i.A; M; Z, O, x.S) \in A[M/i]}$$

When we attempt to devise an operational semantics for this eliminator, however, we get stuck: how should $\mathsf{elim}(i.A; \mathsf{seg}(y); Z, O, x.S)$ reduce? Following Part II, we would like to reduce to $S[y/x]$, but the typing rule does not guarantee that S is apart from y, so this substitution is not permitted for affine interval variables. On a more conceptual level, the elimination principle sets up an isomorphism between structural functions $f : \mathsf{Ival} \to A$ and bridges of type $\mathsf{Bridge}(A, f\,\mathsf{zero}, f\,\mathsf{one})$; higher inductive types are in a way inherently structural.

By leveraging the Kan operations to simulate structural substitution, it is possible to model an interval higher inductive type in affine cubical sets that contains an eliminator with the above type. In the non-dependent case, to give an idea, we can define the reduction for the path constructor as follows.

$$\mathsf{elim}(\_.A; \mathsf{seg}(y); Z, O, x.S) \longmapsto \mathsf{hcom}_A^{0 \to y}(S[0/x]; x = 0 \hookrightarrow S[0/x], x = 1 \hookrightarrow z.S[z/x])$$

Given $Z, O, x.S$ with types as in the rule above, we can construct a path in $\mathsf{Path}(A, Z, O)$ from $\lambda^{\mathbb{I}} y.\, \mathsf{elim}(\_.A; \mathsf{seg}(y); Z, O, x.S)$ to $\lambda^{\mathbb{I}} x.\, S$, although we do not obtain it as an exact equality.

This means that at least some simple higher inductive types can be obtained in Bezem, Coquand, and Huber's model, although the result is certainly less usable than in the structural case, and it is unclear whether, *e.g.*, parameterized HITs exist. On the other hand, the problem spells disaster for any hope of higher inductive types "in the bridge direction", that is, inductive types with bridge rather than path constructors. In that case, we cannot rely on Kan operations to get out of a jam. In fact, we can confirm using relativity that no "bridge interval HIT" can exist.

Theorem 12.1.1. There is no type inductively generated by points zero and one and a bridge seg between them.

Proof. In ITT, the existence of such a type proves function extensionality [Hof95, §3.2.7; Uni13, Lemma 6.3.2]. The same applies here: given $p : (a : A) \to \mathsf{Bridge}(B, f_0\, a, f_1\, a)$, we can derive a map $F \in A \to \mathsf{Ival} \to B$ such that $F\, a\, \mathsf{zero} = f_0\, a \in B$ and $F\, a\, \mathsf{one} = f_1\, a \in B$, swap arguments to get $\lambda i.\, \lambda a.\, F\, i\, a \in \mathsf{Ival} \to A \to B$, then extract a bridge:

$$\lambda^{\mathbf{I}} \boldsymbol{x}.\, \lambda a.\, F\, (\mathsf{seg}(\boldsymbol{x}))\, a \in \mathsf{Bridge}(A \to B, f_0, f_1)$$

Generalizing to allow some dependency, the same argument shows that a pointwise family $(a : A) \to \mathsf{Bridge}(\boldsymbol{x}.B, f_0\, a, f_1\, a)$ implies $\mathsf{Bridge}(\boldsymbol{x}.(a : A) \to B, f_0, f_1)$ for any $\boldsymbol{x}.B$.

Next, we show that this function extensionality for bridges is contradictory. Define $I \in (A_0, A_1 : \mathsf{U}) \to \mathsf{Bridge}(\mathsf{U}, \mathsf{Unit}, \mathsf{Unit})$ like so.

$$I\, A_0\, A_1\, \boldsymbol{x} \coloneqq \mathsf{Gel}_{\boldsymbol{x}}(\mathsf{Unit}, \mathsf{Unit}, \_.\_.A_0 \simeq A_1)$$

Then extent gives us an induced term $B \in \mathsf{Bridge}(\mathsf{U} \to \mathsf{U}, \lambda\_.\, \mathsf{Unit}, \lambda\_.\, \mathsf{Unit})$.

$$B\, \boldsymbol{x}\, A \coloneqq \mathsf{extent}_{\boldsymbol{x}}(A; \_.\mathsf{Unit}, \_.\mathsf{U}, A_0.A_1.\_.\_.\boldsymbol{x}.I\, A_0\, A_1\, \boldsymbol{x})$$

We have a term P as follows, where $\mathsf{idiso}(A)$ is the identity isomorphism at A.

$$P \coloneqq \lambda A.\, \lambda^{\mathbf{I}} \boldsymbol{x}.\, \mathsf{gel}_{\boldsymbol{x}}(\star, \star, \mathsf{idiso}(A)) \in (A : \mathsf{U}) \to \mathsf{Bridge}(\boldsymbol{x}.B\, \boldsymbol{x}\, A, \star, \star)$$

By applying first the just-derived function extensionality and then our characterization of functions at bridge type, we derive the following.

$$(A_0, A_1 : \mathsf{U})\ (p : \mathsf{Bridge}(\mathsf{U}, A_0, A_1)) \to \mathsf{Bridge}(\boldsymbol{x}.B\, \boldsymbol{x}\, (p\, \boldsymbol{x}), \star, \star)$$

But by definition of Gel, this means that any bridge $p : \mathsf{Bridge}(\mathsf{U}, A_0, A_1)$ induces an isomorphism $A_0 \simeq A_1$, which clearly contradicts relativity. □

12.2 Outlook

We have brought internal parametricity to cubical type theory, showing that the latter is a solid backdrop against which to develop the general consequences of internal parametricity and to prove concrete free theorems. We hope that the preliminary results of Chapter 10—extending Bernardy and Moulin's methodology for proving free theorems to HITs and introducing notions such as bridge-discreteness—can serve as a jumping-off point for further investigation of the internally parametric world. Our computational interpretation and formalism likewise set the stage for implementation and metatheoretic analysis.

The theory described in Chapter 11 is a first step towards the study of parametric formalisms, but we have yet to develop metatheorems such as normalization which would be necessary to validate it as a "good" definition. We have developed an experimental typechecker for a (non-cubical) parametric formalism, **`ptt`**[1], forked from Gratzer, Sterling and Birkedal's **`blott`** typechecker for a modal type theory [GSB19]. Based on *normalization by evaluation* [BS91; Abe13], it rests on an algorithm for normalizing terms, but we have not done any work to verify its correctness. As **`ptt`** uses named variables for usability's sake, its relationship with the formalism in Chapter 11—in which we use a novel setup to capture affine variables—is also not immediately clear.

Zooming out, parametric type theory is just one point in a design space of higher-dimensional type theories that is still poorly understood. On the one hand, we have cubical type theory, where structural cubes (in any of several varieties) are preferable and lines in the universe correspond to isomorphisms; on the other, we have parametric type theory, which seems naturally affine and where lines in the universe correspond to relations. Work on *directed type theory* has moreover exhibited universes where lines correspond to functions [Rie18; WL20]. It is at present unclear whether these three varieties can be situated in a broader spectrum of higher-dimensional type theories with univalence-like properties.

[1] `https://github.com/ecavallo/ptt`

Part IV

Cohesive parametricity

Chapter 13

Introduction

We saw in Part III that internal parametricity can be a powerful tool, mechanically resolving problems of considerable complexity in cubical type theory. In a sense, however, we have merely shifted the goalposts: we have not actually proven anything about the smash product of cubical type theory, only theorems about the smash product in a different theory we invented. To exaggerate a little, it is as if we added associativity as an axiom and claimed to have proven it. In particular, our formalism for parametric type theory would not be interpretable using the computational interpretation or presheaf model of cubical type theory introduced in Part I; the elements of those models do not in general satisfy parametricity theorems. In contrast, Reynolds' original work established a property of a non-parametric, set-theoretic model, namely that any element of this model definable in a certain type-theoretic formalism is parametric. The parametric type theory of Part III is a priori useless for this purpose.

In this part, we address the objection by making a further extension to parametric cubical type theory. We separate the theory into two *modes*: one for parametric constructions, one for non-parametric ("pointwise") constructions. Each mode comes with its own notion of context, type, and term; thus we essentially have two separate type theories, with the judgments of the former matching those of Part III and of the latter matching those of Parts I and II. The two halves are able to influence each other, however, via *modal* operators that transform parametric contexts/types into pointwise contexts/types and vice versa.

Using this judgmental and type structure, we are able to move in between the parametric and pointwise worlds, making use of parametricity results also in non-parametric settings. To understand how this plays out, let us draw an analogy with the Reynolds' original methodology. The pointwise type theory is like the set-theoretic model: the elements of its types need not satisfy parametricity properties in general. The parametric type theory, meanwhile, corresponds to the formalism: the elements of its types are guaranteed to behave parametrically. For types such as $(A{:}\mathsf{U}) \to A \to A$, a parametric element

has an underlying pointwise element derivable through the use of the modal operators, mirroring the interpretation of the formalism into the set-theoretic model. What we can say, then, is that *pointwise terms that arise from parametric terms satisfy parametricity properties.*

Getting a bit more specific, we draw our modal operators from the theory of *axiomatic cohesion*, defined by Lawvere [Law07] in a categorical setting and first formulated in type-theoretic terms by Schreiber and Shulman [SS12; Shu18]. To say that a category $\mathcal{C}$ is *cohesive over* another category $\mathcal{D}$ is, on an intuitive level, to say that objects of $\mathcal{C}$ are "spaces" whose collections of "underlying points" are objects of $\mathcal{D}$. (A *category* is a collection of objects equipped with a notion of function between objects satisfying certain axioms.)

As a representative example, let us consider the category of cartesian cubical sets $PSh(\square_c)$, which we have used to model a cubical formalism in Section 3.3.1. Recalling briefly the definition from that section, an object of $PSh(\square_c)$ is a family of sets indexed by contexts of interval variables, with functions between them for each interval substitution.

Definition (Replica of Definition 3.3.2). A cubical set G consists of the following data.

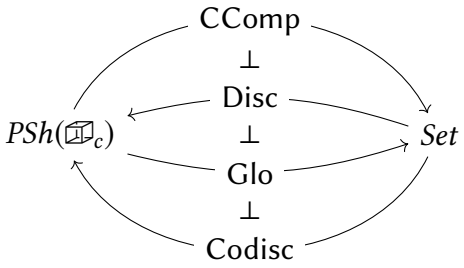
- For every context $\Psi = (x_1 : \mathbb{I}, \ldots, x_n : \mathbb{I})$, a set $G(\Psi)$.
- For every substitution $\psi = (r_1/x_1, \ldots, r_n/x_n)$ replacing the variables of a context Ψ as above with terms in a context Ψ' (variables or 0,1), a function $G(\psi) : G(\Psi) \to G(\Psi')$.

We ask that G preserve identity and composition of substitutions.

The intuition is that a cubical set G is a "space" described as an assemblage of higher-dimensional cubes. Each set $G(x_1 : \mathbb{I}, \ldots, x_n : \mathbb{I})$ is the collection of n-dimensional cubes of the space: $G(\,\cdot\,)$ is the set of points, $G(x : \mathbb{I})$ is the set of lines, $G(x : \mathbb{I}, y : \mathbb{I})$ is the set of squares, and so on. The substitution functions, meanwhile, explain how the cubes attach to each other. Given a line $g \in G(x : \mathbb{I})$, for example, we have a pair of points $G(0/x)(g), G(1/x)(g) \in G(\,\cdot\,)$ representing the endpoints of that line.

The category of cubical sets is cohesive over the category of sets, *Set*: a cubical set G consists of a set $G(\,\cdot\,)$ of underlying points equipped with spatial information in the form of higher-dimensional path structure. In Lawvere's formulation, this is captured by a chain of four functors (functions between categories) relating the two and satisfying

certain properties.



The third functor in this chain, Glo, is the *global sections* functor, which takes a cubical set G and produces its set of underlying points $\mathsf{Glo}(G) \coloneqq G(\,\cdot\,)$. Above it is Disc, the *discrete embedding*, which takes a set S and produces a cubical set with a point for every point of S and trivial higher-dimensional path structure: $\mathsf{Disc}(S)(\Psi) \coloneqq S$ for all Ψ. Disc is *adjoint* to Glo, which means that cubical set functions $\mathsf{Disc}(S) \to G$ are in natural correspondence with set functions $S \to \mathsf{Glo}(G)$: drawing a picture of a cubical set $\mathsf{Disc}(S)$ consisting only of points S in the cubical set G is the same as drawing a picture of S in the set $\mathsf{Glo}(G)$ of points of G. We say that Disc is the *left adjoint* and Glo is the *right adjoint* and write $\mathsf{Disc} \dashv \mathsf{Glo}$ to express the relationship between them.

On the other side of Disc, a right adjoint *codiscrete embedding* Codisc turns a set into a cubical set by adding paths between every pair of elements (and higher-dimensional cubes between these paths); here we have a correspondence between set functions $\mathsf{Glo}(G) \to S$ and cubical set functions $G \to \mathsf{Codisc}(S)$, making Glo left adjoint to Codisc. Finally, the furthest left adjoint is the *connected components* functor, which takes a cubical set to a set by quotienting the set of points (*i.e.*, global sections) by the path relation: we define $\mathsf{CComp}(G) \coloneqq G(\,\cdot\,)/{\approx}$ where $\approx$ is the following relation.

$$a \approx b :\Longleftrightarrow \exists p \in G(x : \mathbb{I}).\, G(0/x)(p) = a \wedge G(1/x)(p) = b$$

For cohesive parametric type theory, we are interested in the cohesive structure of parametric cubical type theory over ordinary cubical type theory. Thus the objects of both the "cohesive" and "underlying points" categories are equipped with cubical structure, but the objects of the cohesive category also carry *bridge* structure.

To translate this picture into our type-theoretic setting, we follow Shulman [Shu18] in using a system of *modalities*. Loosely speaking, a modality is simply a unary operator on types; the terminology originates in *modal logic*, which generalizes formal logic from statements about truth—*e.g.*, "the proposition P is true"—to statements such as "P is necessary" or "P is possible". These different *modes* in which we may consider a statement are related by *modalities*, operators on propositions that transfer between modes. For example, we might define the proposition "$\Box P$" ("necessarily P") to be *true* when P is *necessary*.

The main challenge in designing modal logics and type theories is in handling hypothetical judgments, that is, formulating the interaction between modalities and the context. Turning to the example of cohesion, suppose we have a type $\Gamma \gg A$ type in the parametric mode and wish to take its type of global sections, $\mathsf{Glo}(A)$. In what context does this type live? It is nonsensical to ask that it live over Γ, which is after all a pointwise rather than a parametric context. In truth, the more relevant question is the opposite one: if we want to show $\Gamma \gg \mathsf{Glo}(A)$ type, in what context should A be well-typed?

There are many ways to approach this question, as we discuss further in Section 17.1. Here we emulate the *Fitch style* [Clo18; BCMEPS20], arriving at an answer by exploiting the fact the structure of Glo as a right adjoint. We will formulate its left adjoint, the discrete embedding, as an operator $-.\mathsf{dsc}$ on contexts. We then have the following formation and introduction rules for the global sections type—note that we annotate each judgment with a mode.

$$\frac{\Gamma.\mathsf{dsc} \gg A\ \mathsf{type}\ @\ \mathsf{par}}{\Gamma \gg \mathsf{Glo}(A)\ \mathsf{type}\ @\ \mathsf{pt}} \qquad \frac{\Gamma.\mathsf{dsc} \gg M \in A\ @\ \mathsf{par}}{\Gamma \gg \mathsf{mod}(M) \in \mathsf{Glo}(A)\ @\ \mathsf{pt}}$$

The introduction rule—which in a way forces the formation rule—provides some intuition: the adjunction between Disc and Glo means that "maps" from Γ to $\mathsf{Glo}(A)$ correspond to "maps" from $\Gamma.\mathsf{dsc}$ to A. Following this pattern, our type-theoretic incarnation of cohesion will see the three left adjoints (CComp, Disc, Glo) appearing as operations on contexts, while the three right adjoints (Disc, Glo, Codisc) will be internalizable as type formers. The formulation of elimination rules, meanwhile, raises its own questions of context we defer for now.

Once the modal apparatus is in place, we can apply it to convert between parametric and pointwise results. The main players are the discrete embedding Disc and global sections functor Glo. (Indeed, the only role of the connected components functor is to enable the formulation of rules for Disc, while the codiscrete embedding is principally useful because its existence implies properties of Disc and Glo.) Suppose, for example, we are given a parametric function $F \in (X : \mathsf{U}) \to X \to X \to X\ @\ \mathsf{par}$. This function is defined on all types A in the parametric universe. But the pointwise universe is embedded in the parametric universe via Disc, so we can also apply F at pointwise types A:

$$F\,(\mathsf{Disc}(A)) \in \mathsf{Disc}(A) \to \mathsf{Disc}(A) \to \mathsf{Disc}(A)\ @\ \mathsf{par}$$

With some further work, we can demonstrate that this function between discrete types in the parametric mode corresponds to a function $A \to A \to A$ in the pointwise mode. We thus have an interpretation of the parametric function as a pointwise function. The main result, then, is that this pointwise function inherits the parametricity theorems enjoyed by its parametric equivalent.

Outline In Chapter 14, we develop our cohesive extension of parametric cubical type theory. In Chapter 15, we apply the theory, showing how to take advantage of parametricity in the non-parametric theory. In Chapter 16, we describe an extension of our previously developed formalisms to match the computational extension and sketch a cubical set model. We discuss related work and future directions in Chapter 17.

Chapter 14

Cohesive parametric type theory

We develop a framework for cohesive parametric type theories following the pattern of definition first established in Chapter 3. In Section 14.1, we first define an interval theory, then give a notion of value type system that defines the value types and elements in each interval context. A value type system induces definitions of the closed judgments in the usual way. Up to this point, we are straightforwardly setting the theories of Parts I and III side by side, defining judgments $\Psi \Vdash M \in A \mathbin{@} m$ in each mode $m \in \{\mathsf{par}, \mathsf{pt}\}$.

The next step, taken in Section 14.2, is to extend the closed judgments to open judgments. It is easy enough to give the definition: an open judgment holds when it holds after any closing substitution. It is significantly more complicated to show that this definition satisfies the properties we need, as the forms of context are much more complex than in previous iterations. We spend Section 14.3 doing so. Everything flows from the need to formulate the rules for modal types, to which we finally arrive in Section 14.4. These motivate first modal context operators, then *endpoint hypotheses* and *modal hypotheses*.

Context operators As sketched in Chapter 13, we will have a context operator for each left adjoint of the cohesion situation and a modal type for each right adjoint, as in the following rules for $\mathsf{Disc}(A)$.

$$\frac{\Gamma.\mathsf{cc} \gg A \ \mathsf{type} \mathbin{@} \mathsf{pt}}{\Gamma \gg \mathsf{Disc}(A) \ \mathsf{type} \mathbin{@} \mathsf{par}} \qquad \frac{\Gamma.\mathsf{cc} \gg M \in A \mathbin{@} \mathsf{pt}}{\Gamma \gg \mathsf{mod}(M) \in \mathsf{Disc}(A) \mathbin{@} \mathsf{par}}$$

Thus we must define three modal context operators. We write $-.\mathsf{cc}$ for the connected components functor, $-.\mathsf{dsc}$ for the discrete embedding, and $-.\mathsf{glo}$ for global sections.

$$\frac{\Gamma \ \mathsf{ctx} \mathbin{@} \mathsf{par}}{\Gamma.\mathsf{cc} \ \mathsf{ctx} \mathbin{@} \mathsf{pt}} \qquad \frac{\Gamma \ \mathsf{ctx} \mathbin{@} \mathsf{pt}}{\Gamma.\mathsf{dsc} \ \mathsf{ctx} \mathbin{@} \mathsf{par}} \qquad \frac{\Gamma \ \mathsf{ctx} \mathbin{@} \mathsf{par}}{\Gamma.\mathsf{glo} \ \mathsf{ctx} \mathbin{@} \mathsf{pt}}$$

We define each of these by recursion on the raw context Γ; we then must show that each modal operator takes well-typed contexts (those satisfying Γ ctx @ m) to well-formed contexts.

One wrinkle appears when we try to define the global sections of the context $(\boldsymbol{x} : \mathbf{I})$. The bridge interval is meant to have two global sections, namely the endpoints $\mathbf{0}$ and $\mathbf{1}$. To express this, we introduce a new form of *endpoint hypothesis* that ranges over the two constants.

$$\frac{\Gamma \text{ ctx} \mathbin{@} m}{(\Gamma, \boldsymbol{x} : 2) \text{ ctx} \mathbin{@} m}$$

We can then define $(\Gamma, \boldsymbol{x} : \mathbf{I}).\mathrm{glo} \coloneqq \Gamma.\mathrm{glo}, \boldsymbol{x} : 2$. One unfortunate consequence of this definition is that $-.\mathrm{glo}$ does not restrict to an operator on interval contexts: $(\boldsymbol{x} : \mathbf{I})$ is an interval context but $(\boldsymbol{x} : \mathbf{I}).\mathrm{glo} = (\boldsymbol{x} : 2)$ is not. This is a source of friction when we develop the theory of closing substitutions.

Aside from this exception, the behaviors of the context operators on interval hypotheses are straightforward. The connected components operator deletes bridge interval hypotheses, in effect collapsing them: the bridge interval has a single connected component.

$$(\Gamma, \boldsymbol{x} : \mathbf{I}).\mathrm{cc} \coloneqq \Gamma.\mathrm{cc}$$

It is useful to think of $-.\mathrm{cc}$ as having a similar character to the interval restriction operator $- \setminus \boldsymbol{x}$: where restriction deletes a single bridge interval variable $\boldsymbol{x}$, cc deletes *all* bridge interval variables. The discrete embedding $-.\mathrm{dsc}$ is not defined on bridge interval hypotheses, as these only appear in parametric contexts. Each operator commutes with path interval and endpoint hypotheses.

A final question that needs answering is how to define the action of modalities on *term* hypotheses; this we defer for the moment.

Negative elimination We take two different approaches to elimination: one for the global type $\mathrm{Glo}(A)$ and codiscrete type $\mathrm{Codisc}(A)$, one for the discrete type $\mathrm{Disc}(A)$. The former two have additional structure we can exploit to give simple projection rules: not only are they right adjoints, but their left adjoints are themselves right adjoints. Taking $\mathrm{Glo}(A)$ as our example, we are able to give the following projection, reduction, and

uniqueness rules.

$$\frac{\Gamma.\mathrm{cc.dsc} \gg A\ \mathrm{type}\ @\ \mathrm{par} \qquad \Gamma.\mathrm{cc} \gg P \in \mathrm{Glo}(A)\ @\ \mathrm{pt}}{\Gamma \gg \mathrm{unmod}(P) \in A\ @\ \mathrm{par}}$$

$$\frac{\Gamma.\mathrm{cc.dsc} \gg A\ \mathrm{type}\ @\ \mathrm{par} \qquad \Gamma.\mathrm{cc.dsc} \gg M \in A\ @\ \mathrm{par}}{\Gamma \gg \mathrm{unmod}(\mathrm{mod}(M)) = M \in A\ @\ \mathrm{par}}$$

$$\frac{\Gamma.\mathrm{dsc} \gg A\ \mathrm{type}\ @\ \mathrm{par} \qquad \Gamma \gg P \in \mathrm{Glo}(A)\ @\ \mathrm{pt}}{\Gamma \gg P = \mathrm{mod}(\mathrm{unmod}(P)) \in \mathrm{Glo}(A)\ @\ \mathrm{pt}}$$

We motivate these rules by the following categorical intuition. Per the adjunction between connected components and the discrete embedding, any $\Gamma.\mathrm{cc} \gg P \in \mathrm{Glo}(A)\ @\ \mathrm{pt}$ corresponds to a term $\Gamma \gg P' \in \mathrm{Disc}(\mathrm{Glo}(A))\ @\ \mathrm{par}$. Meanwhile, the adjunction between the discrete embedding and global sections functor provides a *counit* map $\mathrm{Disc}(\mathrm{Glo}(A)) \to A$ induced by the identity function $\mathrm{Glo}(A) \to \mathrm{Glo}(A)$. The projector unmod is then the composite of these two steps.

We note the similarity between these rules and the rules for the bridge application: $\mathrm{Glo}(A)$ is analogous to $\mathrm{Bridge}(\boldsymbol{x}.A, M_0, M_1)$, $-.\mathrm{dsc}$ to $(-, \boldsymbol{x} : \mathbf{I})$, and $-.\mathrm{cc}$ to context restriction $- \setminus -$. In that case, too, we have an adjoint relationship between $- \setminus -$ and $(-, \boldsymbol{x} : \mathbf{I})$, as discussed in Chapter 11. Definitions of this kind are explored in more generality in [GCKGB21].

Positive elimination With the discrete type, on the other hand, we have no further left adjoint upon which to rely. Instead, we formulate a positive elimination rule by introducing a new context former, the modal hypothesis. Recall once more the introduction rule for $\mathrm{Disc}(A)$.

$$\frac{\Gamma.\mathrm{cc} \gg M \in A\ @\ \mathrm{pt}}{\Gamma \gg \mathrm{mod}(M) \in \mathrm{Disc}(A)\ @\ \mathrm{par}}$$

Elements of $\mathrm{Disc}(A)$ are elements of A well-typed under $-.\mathrm{cc}$. If we want to inhabit some type family $d : \mathrm{Disc}(A) \gg B\ \mathrm{type}$, then, it would suffice to show that $B[\mathrm{mod}(a)/d]$ holds given a *modal variable* $(\mathrm{cc} \mid a : A)$. Such variables range exactly over terms that are well-typed under some modality; we will have the following defining rules for contexts and substitutions.

$$\frac{\Gamma\ \mathrm{ctx}\ @\ \mathrm{par} \qquad \Gamma.\mathrm{cc} \gg A\ \mathrm{pretype}}{(\Gamma, (\mathrm{cc} \mid a : A))\ \mathrm{ctx}\ @\ \mathrm{par}} \qquad \frac{\Gamma' \gg \gamma \in \Gamma\ @\ \mathrm{par} \qquad \Gamma'.\mathrm{cc} \gg M \in A\gamma\ @\ \mathrm{pt}}{\Gamma' \gg (\gamma, M/a) \in (\Gamma, (\mathrm{cc} \mid a : A))\ @\ \mathrm{par}}$$

Note in particular that substitutions from Γ' into $(\mathsf{cc} \mid a : A)$ correspond to substitutions from $\Gamma'.\mathsf{cc}$ into A. Thus $(\mathsf{cc} \mid a : -)$ represents the right adjoint to $-.\mathsf{cc}$, which is to say the discrete embedding.

We may now formulate an elimination rule for $\mathsf{Disc}(A)$ as suggested above.

$$\frac{\begin{array}{cc}\Gamma.\mathsf{cc} \gg A\ \mathsf{type}\ @\ \mathsf{pt} & \Gamma, d : \mathsf{Disc}(A) \gg B\ \mathsf{type}\ @\ \mathsf{par} \\ \Gamma \gg P \in \mathsf{Disc}(A)\ @\ \mathsf{par} & \Gamma, (\mathsf{cc} \mid a : A) \gg N \in B[\mathsf{mod}(a)/d]\ @\ \mathsf{par}\end{array}}{\Gamma \gg \mathsf{letdisc}(d.B, P, a.N) \in B[P/d]\ @\ \mathsf{par}}$$

Modal hypotheses and context operators It is useful to more generally allow modal hypotheses under arbitrary compound modalities, sequences $\mu = (\mu_1, \ldots, \mu_n)$ where each μ_i is one of cc, dsc, or glo. Here we follow Gratzer, Kavvos, Nuyts, and Birkedal's **MTT** framework for modal type theories [GKNB20]. This is not only convenient in practice, but also gives us a way to define the right adjoint modal context operators ($-.\mathsf{dsc}$ and $-.\mathsf{glo}$) on term hypotheses.

$$(\Gamma, (\mu \mid a : A)).\mathsf{dsc} \coloneqq \Gamma.\mathsf{dsc}, (\mathsf{cc}, \mu \mid a : A)$$
$$(\Gamma, (\mu \mid a : A)).\mathsf{glo} \coloneqq \Gamma.\mathsf{glo}, (\mathsf{dsc}, \mu \mid a : A)$$

Recall that a modal hypothesis over cc can be thought of as a hypothesis of discrete type. Thus we apply $-.\mathsf{dsc}$ to a modal term hypothesis by adding cc to its modality. By the same token, a modal hypothesis over dsc corresponds to a hypothesis of global section type. Thus we define these operators by what Nuyts, Vezzosi, and Devriese call *left division* [NVD17], that is, by adjusting the modality of each hypothesis.

The leftmost adjoint again demands special treatment. To apply the connected components modality to a hypothesis, we check if it is already typed under the connected components modality. If so, the context application cancels the hypothesis modality. Other hypotheses are simply thrown away; there is no way to access an ordinary term hypothesis beneath $-.\mathsf{cc}$.

$$(\Gamma, (\mu \mid a : A)).\mathsf{cc} \coloneqq \begin{cases} \Gamma.\mathsf{cc}, (\mu' \mid a : A), & \text{if } \mu = \mathsf{cc}, \mu' \\ \Gamma.\mathsf{cc}, & \text{otherwise} \end{cases}$$

Again, it is instructive to draw a parallel with interval restriction. The restriction $- \setminus \boldsymbol{x}$ deletes term hypotheses that succeed $\boldsymbol{x}$ in the context, as these could be instantiated with terms that use $\boldsymbol{x}$. Likewise, $-.\mathsf{cc}$ deletes hypotheses that could use *any* bridge interval variable, which is to say all hypotheses except those hidden behind cc.

Many of the complications of the theory developed below have their root in modal hypotheses. For example, to check that $-.\mathsf{glo}$ takes well-formed contexts to well-formed contexts, we must first know that $\Gamma.\mu \gg A\ \mathsf{pretype}$ implies $\Gamma.\mu.\mathsf{glo}.\mathsf{dsc} \gg A\ \mathsf{pretype}$. Thus careful staging is required.

14.1 Interval theory and type systems

Now we begin making the preceding sketch precise. Every judgment in our cohesive type theory is indexed by a mode, par (parametric) or pt (pointwise).

Definition 14.1.1. The *modes*, m mode, are generated by the following inference rules.

$$\frac{}{\text{par mode}} \qquad \frac{}{\text{pt mode}}$$

The characteristic difference between the two modes is that the parametric mode includes bridges, both on the judgmental and on the type level. We see this first in the definition of interval contexts: bridge interval variables can only be hypothesized in the parametric mode.

Definition 14.1.2. The interval m-contexts, Ψ ictx $@$ m for m mode, are inductively generated by the following inference rules.

$$\frac{}{\cdot \text{ ictx } @\ m} \qquad \frac{\Psi \text{ ictx } @\ m}{(\Psi, x : \mathbb{I}) \text{ ictx } @\ m} \qquad \frac{\Psi \text{ ictx } @ \text{ par}}{(\Psi, \boldsymbol{x} : \mathbf{I}) \text{ ictx } @ \text{ par}}$$

Aside from the restriction of parametric elements to the parametric mode, the development of the interval theory proceeds without change from the single-mode parametric cubical case. Note that we do still allow the formation of constraints such as $\mathbf{0} \equiv \mathbf{1}$, which involve bridge terms but not bridge *variables*, in the pointwise mode.

Definition 14.1.3 (Closed interval elements).

- $\Psi \Vdash r \in \mathbb{I} \ @\ m$ holds when either $r = 0$, $r = 1$, or $r = x$ with $(x : \mathbb{I}) \in \Psi$.
- $\Psi \Vdash \boldsymbol{r} \in \mathbf{I} \ @\ m$ holds when either $\boldsymbol{r} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{1}$, or $\boldsymbol{r} = \boldsymbol{x}$ with $(\boldsymbol{x} : \mathbf{I}) \in \Psi$.

Definition 14.1.4 (Interval substitutions). The judgment $\Psi' \Vdash \psi \in \Psi \ @\ m$ is generated by the following rules.

$$\frac{}{\Psi' \Vdash \cdot \in \cdot \ @\ m} \qquad \frac{\Psi' \Vdash \psi \in \Psi \ @\ m \qquad \Psi' \Vdash r \in \mathbb{I} \ @\ m}{\Psi' \Vdash (\psi, r/x) \in (\Psi, x : \mathbb{I}) \ @\ m}$$

$$\frac{\Psi' \Vdash \boldsymbol{r} \in \mathbf{I} \ @ \text{ par} \qquad \Psi' \setminus \boldsymbol{r} \Vdash \psi \in \Psi \ @ \text{ par}}{\Psi' \Vdash (\psi, \boldsymbol{r}/\boldsymbol{x}) \in (\Psi, \boldsymbol{x} : \mathbf{I}) \ @ \text{ par}}$$

We similarly enrich type systems, and subsequently the closed judgments, by a mode parameter. Again, these definitions are not notably different from the their ordinary parametric (indeed, ordinary cubical) equivalents.

Definition 14.1.5. Given Ψ ictx @ m, an (m, Ψ)-relation is a family of relations indexed by interval substitutions into Ψ.

Definition 14.1.6. A *candidate modal type system* is a five-place relation τ relating modes m, interval m-contexts Ψ, values V and V' with free variables contained in Ψ, and value-coherent (m, Ψ)-PERs R. A candidate is a *modal type system* if it meets the requirements of Definition 3.1.16 at each mode. We write $\tau \vDash \Psi \Vdash V \approx V' \downarrow R \mathbin{@} m$ to mean that $(m, \Psi, V, V', R) \in \tau$.

Given a modal candidate τ, we have component pointwise and parametric candidates (that is, candidate type systems for plain cubical and parametric cubical type theory) defined as follows.

$$\tau_{\mathsf{pt}} \vDash \Psi \Vdash V \approx V' \downarrow R :\Longleftrightarrow \tau \vDash \Psi \Vdash V \approx V' \downarrow R \mathbin{@} \mathsf{pt}$$
$$\tau_{\mathsf{par}} \vDash \Psi \Vdash V \approx V' \downarrow R :\Longleftrightarrow \tau \vDash \Psi \Vdash V \approx V' \downarrow R \mathbin{@} \mathsf{par}$$

Conversely, any pointwise or parametric candidate may be regarded as a modal candidate that contains types only in a single mode.

We define the closed judgments induced by a value type system in the usual way.

Definition 14.1.7 (Closed judgments). Fix a candidate modal type system τ.

- $\Psi \Vdash A = A'$ pretype @ m holds when $A \approx A' \in \Downarrow\tau[R]$ for some (m, Ψ)-PER R.
- $\Psi \Vdash M = M' \in A \mathbin{@} m$ holds when $A \in \Downarrow\tau[R]$ for some (m, Ψ)-PER R such that $M \approx M' \in \Downarrow R$.
- $\Psi \Vdash A = A'$ type @ m holds when $\Psi \Vdash A = A'$ pretype @ m support coercion and homogeneous composition.

14.2 Open judgments

We now fix an ambient candidate modal type system τ and begin deriving the open judgments. Here we get to the meat of the cohesive structure: the modalities.

Definition 14.2.1 (Modalities). The *modalities*, $\mu : m \to n$, are inductively generated by the following rules.

$$\frac{}{\mathsf{id} : m \to m} \qquad \frac{\mu : m \to \mathsf{pt}}{(\mathsf{cc}, \mu) : m \to \mathsf{par}} \qquad \frac{\mu : m \to \mathsf{par}}{(\mathsf{dsc}, \mu) : m \to \mathsf{pt}} \qquad \frac{\mu : m \to \mathsf{pt}}{(\mathsf{glo}, \mu) : m \to \mathsf{par}}$$

Note that modalities will act contravariantly on contexts: given $\mu : m \to n$, the operator $-.\mu$ takes contexts in mode n to contexts in mode m.

Raw contexts are drawn from the following grammar: they consist of modal term hypotheses, bridge and path interval hypotheses, constraints, and the new bridge endpoint hypotheses.

$$\Gamma ::= \cdot \mid \Gamma, (\mu \mid a : A) \mid \Gamma, \boldsymbol{x} : \mathbf{I} \mid \Gamma, x : \mathbb{I} \mid \Gamma, \xi \mid \Gamma, \boldsymbol{x} : \mathbf{2}$$

Notation 14.2.2. We write $a : A$ as shorthand for $(\mathsf{id} \mid a : A)$.

As usual, the definition of the context judgment Γ ctx $@$ m will be one of our last. We first define the closing substitutions and then the open judgments as ranging over raw contexts, with the aim that these be well-behaved when the context arguments are well-formed.

14.2.1 Interval judgments

Before getting into term judgments, we define the open interval and bridge interval endpoint judgments. As before, term hypotheses have no bearing on interval judgments; in particular, a contradictory term assumption like v : Void does not imply any interval equalities.

Definition 14.2.3 (Open interval judgments).

- $\Gamma \gg r \in \mathbb{I} \mathbin{@} m$ holds when either $r = 0$, $r = 1$, or $r = x$ with $(x : \mathbb{I}) \in \Gamma$.
- $\Gamma \gg \boldsymbol{r} \in \mathbf{2} \mathbin{@} m$ holds when either $\boldsymbol{r} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{1}$, or $\boldsymbol{r} = \boldsymbol{x}$ with $(\boldsymbol{x} : \mathbf{2}) \in \Gamma$.
- $\Gamma \gg \boldsymbol{r} \in \mathbf{I} \mathbin{@} m$ holds when either $\Gamma \gg \boldsymbol{r} \in \mathbf{2} \mathbin{@} m$ or $\boldsymbol{r} = \boldsymbol{x}$ with $(\boldsymbol{x} : \mathbf{I}) \in \Gamma$.

An equality—$\Gamma \gg r = r' \in \mathbb{I} \mathbin{@} m$, $\Gamma \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{2} \mathbin{@} m$, or $\Gamma \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I} \mathbin{@} m$—is defined to hold when it follows from the equivalence relation closure of the constraint hypotheses appearing in Γ.

We define the judgments $\Gamma \gg \xi = \xi' \in \mathbb{F} \mathbin{@} m$ and $\Gamma \gg \xi$ satisfied $@$ m as generated by the following rules.

$$\frac{\Gamma \gg r = r' \in \mathbb{I} \mathbin{@} m \qquad \Gamma \gg s = s' \in \mathbb{I} \mathbin{@} m}{\Gamma \gg (r \equiv s) = (r' \equiv s') \in \mathbb{F} \mathbin{@} m} \qquad \frac{\Gamma \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I} \mathbin{@} m \qquad \boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}}{\Gamma \gg (\boldsymbol{r} \equiv \boldsymbol{\varepsilon}) = (\boldsymbol{r}' \equiv \boldsymbol{\varepsilon}) \in \mathbb{F} \mathbin{@} m}$$

$$\frac{\Gamma \gg r = s \in \mathbb{I} \mathbin{@} m}{\Gamma \gg r \equiv s \text{ satisfied } @ m} \qquad \frac{\boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}}{\Gamma \gg \boldsymbol{\varepsilon} \equiv \boldsymbol{\varepsilon} \text{ satisfied } @ m}$$

Definition 14.2.4. Given $\boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}$, we define $\neg\boldsymbol{\varepsilon}$ to be its opposite.

$$\neg\mathbf{0} \coloneqq \mathbf{1} \qquad\qquad \neg\mathbf{1} \coloneqq \mathbf{0}$$

14.2.2 Context operators: modalities and restriction

To state the rules for closing substitutions, we must first define the modal operators on contexts, as these appear in the defining rule for substitutions into modal hypotheses. The intent is that we have $\Gamma.\mu\ \mathrm{ctx}\ @\ m$ whenever $\Gamma\ \mathrm{ctx}\ @\ n$ and $\mu : m \to n$.

Definition 14.2.5. Given a context Γ, we define the context $\Gamma.\mu$ for the three basic modalities (cc, dsc, and glo) in Figure 14.1. Application of a compound modality is defined by sequential application of basic modalities: $\Gamma.(\mathrm{cc}, \mu) \coloneqq \Gamma.\mathrm{cc}.\mu$ and so on.

The salient aspects of these definitions are their behavior on bridge interval hypotheses and term hypotheses, reproduced below.

$$(\boldsymbol{x} : \mathbf{I}).\mathrm{cc} \coloneqq \cdot \qquad (\mu \mid a : A).\mathrm{cc} \coloneqq \begin{cases} (\mu' \mid a : A), & \text{if } \mu = \mathrm{cc}, \mu' \\ \cdot, & \text{otherwise} \end{cases}$$

$$(\mu \mid a : A).\mathrm{dsc} \coloneqq (\mathrm{cc}, \mu \mid a : A)$$

$$(\boldsymbol{x} : \mathbf{I}).\mathrm{glo} \coloneqq \boldsymbol{x} : \mathbf{2} \qquad (\mu \mid a : A).\mathrm{glo} \coloneqq (\mathrm{dsc}, \mu \mid a : A)$$

The connected components operator squashes interval hypotheses, while the global sections operator replaces them with endpoint hypotheses. The two right adjoints evaluate on term hypotheses by adding their left adjoints to the hypothesis modality, while the connected components operator removes all term hypotheses not beneath cc. The evaluation of cc on constraints is also somewhat tricky: it leaves constraints on endpoints alone and squashes consistent equations on variables while preserving inconsistent equations. Each modality also induces a functorial action on substitutions following the same pattern. Here we intend to have $\Gamma'.\mu \gg (\gamma : \Gamma) \otimes \mu \in \Gamma.\mu\ @\ m$ whenever $\mu : m \to n$ and $\Gamma' \gg \gamma \in \Gamma\ @\ n$.

Definition 14.2.6. Given a context Γ and substitution γ into Γ, we define the substitution $(\gamma : \Gamma) \otimes \mu$ for the basic modalities in Figure 14.2. The action of compound modalities is defined as with contexts.

Remark 14.2.7. The effect of a substitution $(\gamma : \Gamma) \otimes \mu$ on syntax is the same as that of γ. That is, if M is a term depending only on the variables in $\Gamma.\mu$, then $M[(\gamma : \Gamma) \otimes \mu] = M\gamma$.

Finally, we update the definition of interval restriction (Definition 9.1.9) to handle the new forms of hypothesis. We also specify that restriction by an endpoint variable, like that by an endpoint constant, is the identity.

Definition 14.2.8 (Interval restriction). Given a context Γ and term $\Gamma \gg \boldsymbol{r} \in \mathbf{I}$, we define $\Gamma \setminus \boldsymbol{r}$ in Figure 14.3. The action $(\gamma : \Gamma) \setminus \boldsymbol{r}$ is defined analogously.

Connected components (Γ.cc)

$$
\begin{aligned}
\cdot.\mathsf{cc} &\coloneqq \cdot \\
(\Gamma, x : \mathbb{I}).\mathsf{cc} &\coloneqq \Gamma.\mathsf{cc}, x : \mathbb{I} \\
(\Gamma, \boldsymbol{x} : \mathbf{2}).\mathsf{cc} &\coloneqq \Gamma.\mathsf{cc}, \boldsymbol{x} : \mathbf{2} \\
(\Gamma, \boldsymbol{x} : \mathbf{I}).\mathsf{cc} &\coloneqq \Gamma.\mathsf{cc} \\
(\Gamma, r \equiv s).\mathsf{cc} &\coloneqq \Gamma.\mathsf{cc}, r \equiv s \\
(\Gamma, \boldsymbol{r} \equiv \boldsymbol{\varepsilon}).\mathsf{cc} &\coloneqq \begin{cases} \Gamma.\mathsf{cc}, \boldsymbol{r} \equiv \boldsymbol{\varepsilon}, & \text{if } \Gamma \gg \boldsymbol{r} \in \mathbf{2} \ @\ \mathsf{par} \\ \Gamma.\mathsf{cc}, \boldsymbol{\neg\varepsilon} \equiv \boldsymbol{\varepsilon}, & \text{if not but } \Gamma \gg \boldsymbol{r} = \boldsymbol{\neg\varepsilon} \in \mathbf{I} \ @\ \mathsf{par} \\ \Gamma.\mathsf{cc}, & \text{otherwise} \end{cases} \\
(\Gamma, (\mu \mid a : A)).\mathsf{cc} &\coloneqq \begin{cases} \Gamma.\mathsf{cc}, (\mu' \mid a : A), & \text{if } \mu = \mathsf{cc}, \mu' \\ \Gamma.\mathsf{cc}, & \text{otherwise} \end{cases}
\end{aligned}
$$

Discrete embedding (Γ.dsc)

$$
\begin{aligned}
\cdot.\mathsf{dsc} &\coloneqq \cdot \\
(\Gamma, x : \mathbb{I}).\mathsf{dsc} &\coloneqq \Gamma.\mathsf{dsc}, x : \mathbb{I} \\
(\Gamma, \boldsymbol{x} : \mathbf{2}).\mathsf{dsc} &\coloneqq \Gamma.\mathsf{dsc}, \boldsymbol{x} : \mathbf{2} \\
(\Gamma, \xi).\mathsf{dsc} &\coloneqq \Gamma.\mathsf{dsc}, \xi \\
(\Gamma, (\mu \mid a : A)).\mathsf{dsc} &\coloneqq \Gamma.\mathsf{dsc}, (\mathsf{cc}, \mu \mid a : A)
\end{aligned}
$$

Global sections (Γ.glo)

$$
\begin{aligned}
\cdot.\mathsf{glo} &\coloneqq \cdot \\
(\Gamma, x : \mathbb{I}).\mathsf{glo} &\coloneqq \Gamma.\mathsf{glo}, x : \mathbb{I} \\
(\Gamma, \boldsymbol{x} : \mathbf{2}).\mathsf{glo} &\coloneqq \Gamma.\mathsf{glo}, \boldsymbol{x} : \mathbf{2} \\
(\Gamma, \boldsymbol{x} : \mathbf{I}).\mathsf{glo} &\coloneqq \Gamma.\mathsf{glo}, \boldsymbol{x} : \mathbf{2} \\
(\Gamma, \xi).\mathsf{glo} &\coloneqq \Gamma.\mathsf{glo}, \xi \\
(\Gamma, (\mu \mid a : A)).\mathsf{glo} &\coloneqq \Gamma.\mathsf{glo}, (\mathsf{dsc}, \mu \mid a : A)
\end{aligned}
$$

Figure 14.1: Definitions of the modal context operators

Connected components $((\gamma : \Gamma) \otimes \mathrm{cc})$

$$
\begin{aligned}
(\cdot \,:\, \cdot) \otimes \mathrm{cc} &\coloneqq \cdot \\
((\gamma, M/a) : (\Gamma, (\mu \mid a : A))) \otimes \mathrm{cc} &\coloneqq \begin{cases} ((\gamma : \Gamma) \otimes \mathrm{cc}, M/a), & \text{if } \mu = \mathrm{cc}, \mu' \\ (\gamma : \Gamma) \otimes \mathrm{cc}, & \text{otherwise} \end{cases} \\
((\gamma, r/x) : (\Gamma, x : \mathbb{I})) \otimes \mathrm{cc} &\coloneqq ((\gamma : \Gamma) \otimes \mathrm{cc}, r/x) \\
((\gamma, \boldsymbol{r}/\boldsymbol{x}) : (\Gamma, \boldsymbol{x} : \mathbf{2})) \otimes \mathrm{cc} &\coloneqq ((\gamma : \Gamma) \otimes \mathrm{cc}, \boldsymbol{r}/\boldsymbol{x}) \\
((\gamma, \boldsymbol{r}/\boldsymbol{x}) : (\Gamma, \boldsymbol{x} : \mathbf{I})) \otimes \mathrm{cc} &\coloneqq (\gamma : \Gamma) \otimes \mathrm{cc} \\
(\gamma : (\Gamma, \xi)) \otimes \mathrm{cc} &\coloneqq (\gamma : \Gamma) \otimes \mathrm{cc}
\end{aligned}
$$

Discrete embedding $((\gamma : \Gamma) \otimes \mathrm{dsc})$

$$(\gamma : \Gamma) \otimes \mathrm{dsc} \coloneqq \gamma$$

Global sections $((\gamma : \Gamma) \otimes \mathrm{glo})$

$$(\gamma : \Gamma) \otimes \mathrm{glo} \coloneqq \gamma$$

Figure 14.2: Definitions of the modal substitution operators

Interval restriction $(\Gamma \setminus \boldsymbol{r})$
If a bridge term $\boldsymbol{r}$ is equal to an endpoint term, then restriction has no effect.

$$\Gamma \setminus \boldsymbol{r} \coloneqq \Gamma \quad \text{if } \Gamma \gg \boldsymbol{r} = \boldsymbol{s} \in \mathbf{I} \;@\; \mathrm{par} \text{ for some } \Gamma \gg \boldsymbol{s} \in \mathbf{2} \;@\; \mathrm{par}$$

Otherwise, restriction is defined as follows.

$$
\begin{aligned}
(\Gamma, y : \mathbb{I}) \setminus \boldsymbol{x} &\coloneqq (\Gamma \setminus \boldsymbol{x}), y : \mathbb{I} \\
(\Gamma, \boldsymbol{y} : \mathbf{2}) \setminus \boldsymbol{x} &\coloneqq (\Gamma \setminus \boldsymbol{x}), \boldsymbol{y} : \mathbf{2} \\
(\Gamma, \boldsymbol{y} : \mathbf{I}) \setminus \boldsymbol{x} &\coloneqq \begin{cases} \Gamma & \text{if } \boldsymbol{x} = \boldsymbol{y} \\ (\Gamma \setminus \boldsymbol{x}), \boldsymbol{y} : \mathbf{I} & \text{otherwise} \end{cases} \\
(\Gamma, y : \mathbb{I}) \setminus \boldsymbol{x} &\coloneqq \Gamma \setminus \boldsymbol{x}, y : \mathbb{I} \\
(\Gamma, \xi) \setminus \boldsymbol{x} &\coloneqq (\Gamma \setminus \boldsymbol{x}), \xi \\
(\Gamma, (\mu \mid a : A)) \setminus \boldsymbol{x} &\coloneqq \begin{cases} \Gamma \setminus \boldsymbol{x}, (\mu \mid a : A), & \text{if } \mu = (\mathrm{cc}, \mu') \\ \Gamma \setminus \boldsymbol{x}, & \text{otherwise} \end{cases}
\end{aligned}
$$

Figure 14.3: Definition of interval restriction

Notable in this definition is the effect on term hypotheses. In plain parametric type theory, restriction deletes hypotheses that proceed the interval hypothesis; here we have nearly the same behavior, but hypotheses under the connected component modality can be left alone, as they cannot depend on any interval variables.

We can observe a couple of equations already.

Proposition 14.2.9. The following equations on contexts hold up to syntactic equality.

$$(\Gamma \setminus \boldsymbol{r}).\mathrm{cc} = \Gamma.\mathrm{cc} \qquad\qquad \Gamma.\mathrm{dsc}.\mathrm{cc} = \Gamma$$

The first of these equations matches the previously mentioned intuition that where restriction removes a single interval variable, cc removes all bridge interval variables.

14.2.3 Extended interval contexts

The nature of the computational interpretation of type theory is that the open judgments are defined from the closed judgments; consequently, the properties of open judgments flow from the properties of the closed judgments. In our cohesive type theory, however, this is complicated by the fact that the "closed" (*i.e.*, interval) contexts and substitutions are not closed under the modalities: $(\boldsymbol{x} : \mathbf{I})$ is an interval context, but $(\boldsymbol{x} : \mathbf{I}).\mathrm{glo} \coloneqq (\boldsymbol{x} : 2)$ is not.

It is therefore technically convenient to introduce a notion of "extended" closed judgment which allows for endpoint hypotheses but still excludes term hypotheses. This class of contexts and substitutions *is* closed under the modalities and will help us get off the ground on our way to the open judgments.

Definition 14.2.10. The *extended interval m-contexts*, Υ eictx @ m for m mode, are inductively generated by the following inference rules.

$$\frac{}{\cdot\ \mathsf{eictx}\ @\ m} \qquad \frac{\Upsilon\ \mathsf{eictx}\ @\ m}{(\Upsilon, x : \mathbb{I})\ \mathsf{eictx}\ @\ m} \qquad \frac{\Upsilon\ \mathsf{eictx}\ @\ m}{(\Upsilon, \boldsymbol{x} : 2)\ \mathsf{eictx}\ @\ m} \qquad \frac{\Upsilon\ \mathsf{eictx}\ @\ \mathsf{par}}{(\Upsilon, \boldsymbol{x} : \mathbf{I})\ \mathsf{eictx}\ @\ \mathsf{par}}$$

Proposition 14.2.11. If Υ eictx @ n and $\mu : m \to n$, then $\Upsilon.\mu$ eictx @ m.

Definition 14.2.12 (Extended interval substitutions). The extended interval substitutions, $\Upsilon' \gg \psi \in \Upsilon\ @\ m$, are inductively defined by the following rules.

$$\frac{}{\Upsilon' \gg \cdot \in \cdot\ @\ m} \qquad \frac{\Upsilon' \gg \psi \in \Upsilon\ @\ m \qquad \Upsilon' \gg r \in \mathbb{I}\ @\ m}{\Upsilon' \gg (\psi, r/x) \in (\Upsilon, x : \mathbb{I})\ @\ m}$$

$$\frac{\Upsilon' \gg \psi \in \Upsilon\ @\ m \qquad \Upsilon' \gg \boldsymbol{r} \in 2\ @\ m}{\Upsilon' \gg (\psi, \boldsymbol{r}/\boldsymbol{x}) \in (\Upsilon, \boldsymbol{x} : 2)\ @\ m} \qquad \frac{\Upsilon' \gg \boldsymbol{r} \in \mathbf{I}\ @\ \mathsf{par} \qquad \Upsilon' \setminus \boldsymbol{r} \gg \psi \in \Upsilon\ @\ \mathsf{par}}{\Upsilon' \gg (\psi, \boldsymbol{r}/\boldsymbol{x}) \in (\Upsilon, \boldsymbol{x} : \mathbf{I})\ @\ \mathsf{par}}$$

When the domain is a genuine interval context, we write $\Psi \Vdash \psi = \psi' \in \Upsilon \mathbin{@} m$.

It is simple to check directly that the extended closed substitutions satisfy the properties we eventually hope to extend to *all* substitutions: the well-formedness of the actions on substitutions and the adjunctions between successive modalities.

Proposition 14.2.13. Given any $\Upsilon' \gg \psi \in \Upsilon \mathbin{@} n$ and $\mu : m \to n$, the action of μ on ψ is well-typed: we have $\Upsilon'.\mu \gg (\psi : \Upsilon) \otimes \mu \in \Upsilon \mathbin{@} m$.

Proposition 14.2.14 (Adjunctions).

- We have $\Upsilon'.\mathsf{cc} \gg \psi \in \Upsilon \mathbin{@} \mathsf{pt}$ if and only if $\Upsilon' \gg \psi \in \Upsilon.\mathsf{dsc} \mathbin{@} \mathsf{par}$.
- We have $\Upsilon'.\mathsf{dsc} \gg \psi \in \Upsilon \mathbin{@} \mathsf{par}$ if and only if $\Upsilon' \gg \psi \in \Upsilon.\mathsf{glo} \mathbin{@} \mathsf{pt}$.

Using the notion of extended substitution, we extend the closed judgments to extended contexts in the standard way: a judgment holds when all of its closed instantiations hold. In turn we get a definition of extended closing substitution.

Definition 14.2.15 (Extended closed judgments). We extend the typing judgments to extended interval contexts pointwise. $\Upsilon \gg A = A'$ pretype $@\ m$ is defined to hold when $\Psi \Vdash A\psi = A'\psi$ pretype $@\ m$ for all $\Psi \Vdash \psi \in \Upsilon \mathbin{@} m$, and we define $\Upsilon \gg A = A'$ type $@\ m$ and $\Upsilon \gg M = M' \in A \mathbin{@} m$ analogously.

Definition 14.2.16 (Extended closing substitutions). We define the extended closing substitutions $\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@} m$ inductively as follows.

$$\frac{\Upsilon\ \mathsf{ictx} \mathbin{@} m}{\Upsilon \gg \cdot = \cdot \in \cdot \mathbin{@} m}$$

$$\frac{\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@} n \qquad \mu : m \to n \qquad \Upsilon.\mu \gg M = M' \in A\gamma \mathbin{@} m}{\Upsilon \gg (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma, (\mu \mid a : A)) \mathbin{@} n}$$

$$\frac{\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@} m \qquad \Upsilon \gg r \in \mathbb{I} \mathbin{@} m}{\Upsilon \gg (\gamma, r/x) = (\gamma', r/x) \in (\Gamma, x : \mathbb{I}) \mathbin{@} m} \qquad \frac{\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@} m \qquad \boldsymbol{\varepsilon} \in \{\mathbf{0}, \mathbf{1}\}}{\Upsilon \gg (\gamma, \boldsymbol{\varepsilon}/\boldsymbol{x}) = (\gamma', \boldsymbol{\varepsilon}/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{2}) \mathbin{@} m}$$

$$\frac{\Upsilon \setminus \boldsymbol{r} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{par} \qquad \Upsilon \gg \boldsymbol{r} \in \mathbf{I} \mathbin{@} \mathsf{par}}{\Upsilon \gg (\gamma, \boldsymbol{r}/\boldsymbol{x}) = (\gamma', \boldsymbol{r}/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{I}) \mathbin{@} \mathsf{par}}$$

$$\frac{\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@} m \qquad \Upsilon \gg \xi\gamma\ \mathsf{satisfied} \mathbin{@} m}{\Upsilon \gg \gamma = \gamma' \in (\Gamma, \xi) \mathbin{@} m}$$

We write $\Psi \gg \gamma = \gamma' \in \Gamma \mathbin{@} m$ when the domain is a genuine interval context.

Note the appearance of $\Upsilon.\mu$ in the rule for modal hypotheses. Even if Υ is a genuine interval context, $\Upsilon.\mu$ need not be. Thus the extended judgments are essential to give a definition of closing substitution.

14.2.4 Open type and term judgments

As always, the closing substitutions give us the open term judgments and in turn the context and substitution judgments.

Definition 14.2.17 (Open judgments). We define the open typing judgments pointwise as follows.

- $\Gamma \gg A = A'$ pretype $@\ m$ when $\Psi \Vdash A\gamma = A'\gamma'$ pretype $@\ m$ for all $\Psi \Vdash \gamma = \gamma' \in \Gamma\ @\ m$.
- $\Gamma \gg A = A'$ type $@\ m$ when $\Psi \Vdash A\gamma = A'\gamma'$ type $@\ m$ for all $\Psi \Vdash \gamma = \gamma' \in \Gamma\ @\ m$.
- $\Gamma \gg M = M' \in A\ @\ m$ when $\Psi \Vdash M\gamma = M'\gamma' \in A\gamma\ @\ m$ for all $\Psi \Vdash \gamma = \gamma' \in \Gamma\ @\ m$.

Definition 14.2.18 (Substitutions). We define the substitution judgment, $\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ m$, as inductively generated by the following rules.

$$\frac{}{\Gamma' \gg \cdot = \cdot \in \cdot\ @\ m}$$

$$\frac{\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ n \qquad \mu : m \to n \qquad \Gamma'.\mu \gg M = M' \in A\gamma\ @\ m}{\Gamma' \gg (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma, (\mu \mid a : A))\ @\ n}$$

$$\frac{\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ m \qquad \Gamma' \gg r = r' \in \mathbb{I}\ @\ m}{\Gamma' \gg (\gamma, r/x) = (\gamma', r'/x) \in (\Gamma, x : \mathbb{I})\ @\ m}$$

$$\frac{\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ m \qquad \Gamma' \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{2}\ @\ m}{\Gamma' \gg (\gamma, \boldsymbol{r}/\boldsymbol{x}) = (\gamma', \boldsymbol{r}'/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{2})\ @\ m}$$

$$\frac{\Gamma' \setminus \boldsymbol{r} \gg \gamma = \gamma' \in \Gamma\ @\ \mathsf{par} \qquad \Gamma' \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I}\ @\ \mathsf{par}}{\Gamma' \gg (\gamma, \boldsymbol{r}/\boldsymbol{x}) = (\gamma', \boldsymbol{r}'/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{I})\ @\ \mathsf{par}}$$

$$\frac{\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ m \qquad \Gamma' \gg \xi\gamma\ \mathsf{satisfied}\ @\ m}{\Gamma' \gg \gamma = \gamma' \in (\Gamma, \xi)\ @\ m}$$

Definition 14.2.19 (Contexts). We define the context judgment, $\Gamma = \Gamma'\ \mathsf{ctx}\ @\ m$, as generated by the following rules.

$$\frac{}{\cdot = \cdot\ \mathsf{ctx}\ @\ m} \qquad \frac{\mu : m \to n \qquad \Gamma = \Gamma'\ \mathsf{ctx}\ @\ n \qquad \Gamma.\mu \gg A = A'\ \mathsf{pretype}\ @\ m}{(\Gamma, (\mu \mid a : A)) = (\Gamma', (\mu \mid a : A'))\ \mathsf{ctx}\ @\ n}$$

$$\frac{\Gamma = \Gamma'\ \mathsf{ctx}\ @\ m}{(\Gamma, x : \mathbb{I}) = (\Gamma', x : \mathbb{I})\ \mathsf{ctx}\ @\ m} \qquad \frac{\Gamma = \Gamma'\ \mathsf{ctx}\ @\ m}{(\Gamma, \boldsymbol{x} : 2) = (\Gamma', \boldsymbol{x} : 2)\ \mathsf{ctx}\ @\ m}$$

$$\frac{\Gamma = \Gamma'\ \mathsf{ctx}\ @\ \mathsf{par}}{(\Gamma, \boldsymbol{x} : \mathbf{I}) = (\Gamma', \boldsymbol{x} : \mathbf{I})\ \mathsf{ctx}\ @\ \mathsf{par}} \qquad \frac{\Gamma = \Gamma'\ \mathsf{ctx}\ @\ m \qquad \Gamma \gg \xi = \xi' \in \mathbb{F}\ @\ m}{(\Gamma, \xi) = (\Gamma', \xi')\ \mathsf{ctx}\ @\ m}$$

14.3 Rules for modal operators and hypotheses

With the definitions of the judgments complete, we now verify that modal context operators and hypotheses validate the expected rules. Namely, the context operators should take well-formed contexts to well-formed contexts, their actions on substitutions should be likewise well-behaved, and we should be able to access modal variables from the context under the right conditions.

As the route to these theorems is somewhat circuitous, we encourage readers disinterested in the gnarly details to skip to Section 14.3.2, where the main results can be found.

14.3.1 Extended closing substitutions

As the open judgments are defined by closing substitutions, we start with their properties. It is fruitful to more generally consider extended closing substitutions. First we have some basic stability results, straightforward consequences of the definitions.

Proposition 14.3.1 (Extended closed stability for terms). Given any $\Upsilon' \gg \psi \in \Upsilon\ @\ m$ and any $\Upsilon \gg A = A'\ \mathsf{pretype}\ @\ m$, we have $\Upsilon' \gg A\psi = A'\psi\ \mathsf{pretype}\ @\ m$; likewise for types and terms.

Lemma 14.3.2 (Extended closed stability for substitutions). Given $\Upsilon' \gg \psi \in \Upsilon\ @\ m$ and $\Upsilon \gg \gamma = \gamma' \in \Gamma\ @\ m$, we have $\Upsilon' \gg \gamma\psi = \gamma'\psi \in \Gamma\ @\ m$.

Proof. By induction on $\Upsilon \gg \gamma = \gamma' \in \Gamma\ @\ m$. The modal hypothesis case relies on the functorial action of modalities on extended closing substitutions (Proposition 14.2.13). □

Corollary 14.3.3 (Extended instantiation). Given any $\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@} m$ and any $\Gamma \gg A = A'$ pretype $@\ m$, we have $\Upsilon \gg A\gamma = A'\gamma'$ pretype $@\ m$; likewise for types and terms.

In order to avoid repetition, we introduce the following notion of modality *division*, following Nuyts, Vezzosi, and Devriese [NVD17].

Definition 14.3.4 (Division). We define $\mu \div \nu : m \to n$ as a partial function of $\mu : m \to p$ and $\nu : n \to p$ as follows.

$$\begin{aligned} \mu \div \mathsf{id} &\coloneqq \mu \\ (\mathsf{cc}, \mu) \div (\mathsf{cc}, \nu) &\coloneqq \mu \div \nu \\ \mu \div (\mathsf{dsc}, \nu) &\coloneqq (\mathsf{cc}, \mu) \div \nu \\ \mu \div (\mathsf{glo}, \nu) &\coloneqq (\mathsf{dsc}, \mu) \div \nu \end{aligned}$$

This expresses compactly the effect of context operators on modal hypotheses, as shown by the following equations.

$$\begin{aligned} &\Gamma, (\mu \mid a : A).\nu = \Gamma.\nu, (\mu \div \nu \mid a : A) && \text{if } \mu \div \nu \text{ is defined} \\ &\Gamma, (\mu \mid a : A).\nu = \Gamma.\nu && \text{otherwise} \end{aligned}$$

The following lemma is key; it tells us that modal hypotheses remain well-typed after the application of a modality.

Lemma 14.3.5 (Division of extended closed terms). Let $\mu : m \to p$ and $\nu : n \to p$. If $\Upsilon.\mu \gg M = M' \in A \mathbin{@} m$ and $\mu \div \nu$ is defined, then $\Upsilon.\nu.(\mu \div \nu) \gg M = M' \in A \mathbin{@} m$.

Proof. By induction on ν.

- Case: $\nu = \mathsf{id}$. Immediate.
- Case: $\nu = (\mathsf{cc}, \nu')$. Then we must have $\mu = (\mathsf{cc}, \mu')$, and the result follows by induction hypothesis applied with ν' and μ'.
- Case: $\nu = (\mathsf{dsc}, \nu')$. We have $\Upsilon = \Upsilon.\mathsf{dsc}.\mathsf{cc}$, so we can apply the induction hypothesis with ν' and (cc, μ) at $\Upsilon.\mathsf{dsc}.\mathsf{cc}.\mu \gg M = M' \in A \mathbin{@} m$ to get the result.
- Case: $\nu = (\mathsf{glo}, \nu')$. We have a substitution $\Upsilon.\mathsf{glo}.\mathsf{dsc}.\mu \gg \mathsf{id}_{\Upsilon.\mu} \in \Upsilon.\mu$ by the action of modalities on and adjunction laws for extended interval substitutions. It follows by stability that $\Upsilon.\mathsf{glo}.\mathsf{dsc}.\mu \gg M = M' \in A \mathbin{@} m$. Applying the induction hypothesis with ν' and (dsc, μ) gives the result. □

Corollary 14.3.6 (Modalities on extended substitutions). Given $\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@} n$ and $\mu : m \to n$, we have $\Upsilon.\mu \gg (\gamma : \Gamma) \otimes \mu = (\gamma' : \Gamma) \otimes \mu \in \Gamma.\mu \mathbin{@} m$.

Proof. By induction on $\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@} n$, applying Lemma 14.3.5 in the modal hypothesis case. □

We have an analogous lemma for the action of interval restriction.

Lemma 14.3.7 (Restriction on extended substitutions). Given $\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@}$ par and any $\Gamma \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I} \mathbin{@}$ par, we have $\Upsilon \setminus \boldsymbol{r}\gamma \gg (\gamma : \Gamma) \setminus \boldsymbol{r} = (\gamma' : \Gamma') \setminus \boldsymbol{r}' \in \Gamma \setminus \boldsymbol{r} \mathbin{@}$ par.

Proof. If $\Gamma \gg \boldsymbol{r} = \boldsymbol{s} \in \mathbf{I} \mathbin{@}$ par for some $\Gamma \gg \boldsymbol{s} \in \mathbf{2} \mathbin{@}$ par, then the result is trivial. If not, we proceed by induction on $\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@}$ par.

- Case: $\Upsilon \gg \cdot = \cdot \in \cdot \mathbin{@}$ par. Immediate.
- Case: $\Upsilon \gg (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma, (\mu \mid a : A)) \mathbin{@}$ par.
  - Case: $\mu = (\mathrm{cc}, \mu')$. By assumption, $\Upsilon \gg \gamma = \gamma' \in \Gamma \mathbin{@}$ par. We can conclude $\Gamma \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I} \mathbin{@}$ par from $\Gamma, (\mu \mid a : A) \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I} \mathbin{@}$ par. Thus $\Upsilon \setminus \boldsymbol{r}\gamma \gg (\gamma : \Gamma) \setminus \boldsymbol{r} = (\gamma' : \Gamma) \setminus \boldsymbol{r}' \in \Gamma \setminus \boldsymbol{r} \mathbin{@}$ par by induction hypothesis. We moreover have $\Upsilon.\mathrm{cc}.\mu' \gg M = M' \in A\gamma \mathbin{@} m$, and we know that $\Upsilon.\mathrm{cc}.\mu' = \Upsilon \setminus \boldsymbol{r}\gamma.\mathrm{cc}.\mu'$.
  - Case: $\mu = \cdot$ or $\mu = (\mathrm{glo}, \mu')$. Immediate by induction hypothesis.
- Case: $\Upsilon \gg (\gamma, s/x) = (\gamma', s/x) \in (\Gamma, x : \mathbb{I}) \mathbin{@}$ par. By induction hypothesis and the substitution formation rule.
- Case: $\Upsilon \gg (\gamma, \boldsymbol{s}/\boldsymbol{x}) = (\gamma', \boldsymbol{s}/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{2}) \mathbin{@}$ par. As $\boldsymbol{r}$ is not identified with an endpoint, we know that $\boldsymbol{r} \neq \boldsymbol{x}$. It follows that $\Gamma \gg \boldsymbol{r} = \boldsymbol{r}' \in \mathbf{I} \mathbin{@}$ par. By induction hypothesis we then have $\Upsilon \setminus \boldsymbol{r}\gamma \gg (\gamma : \Gamma) \setminus \boldsymbol{r} = (\gamma' : \Gamma') \setminus \boldsymbol{r}' \in \Gamma \setminus \boldsymbol{r} \mathbin{@}$ par. As $\Upsilon \gg \boldsymbol{s} \in \mathbf{2} \mathbin{@}$ par, we also have $\Upsilon \setminus \boldsymbol{r}\gamma \gg \boldsymbol{s} \in \mathbf{2} \mathbin{@}$ par.
- Case: $\Upsilon \gg (\gamma, \boldsymbol{s}/\boldsymbol{x}) = (\gamma', \boldsymbol{s}/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{I}) \mathbin{@}$ par.
  - Case: $\boldsymbol{r} = \boldsymbol{x}$.
    By the assumptions of this case, we have $\Upsilon \setminus \boldsymbol{r} \gg \gamma = \gamma' \in \Gamma \mathbin{@}$ par as required.
  - Case: $\boldsymbol{r} \neq \boldsymbol{x}$.
    Then $\Gamma \gg \boldsymbol{r} \in \mathbf{I} \mathbin{@}$ par. By induction hypothesis we get $\Upsilon \setminus \boldsymbol{s} \setminus \boldsymbol{r}\gamma \gg (\gamma : \Gamma) \setminus \boldsymbol{r} = (\gamma' : \Gamma') \setminus \boldsymbol{r}' \in \Gamma \setminus \boldsymbol{r} \mathbin{@}$ par, and we can see that $\Upsilon \setminus \boldsymbol{s} \setminus \boldsymbol{r}\gamma = \Upsilon \setminus \boldsymbol{r}\gamma \setminus \boldsymbol{s}$.
- Case: $\Upsilon \gg \gamma = \gamma' \in (\Gamma, \xi) \mathbin{@} m$. By induction hypothesis and the substitution formation rule. As $\boldsymbol{r}$ is not identified with an endpoint, we know that ξ does not mention $\boldsymbol{r}$. □

We next check that the components-discrete and discrete-global adjunctions hold for extended closing substitutions.

Lemma 14.3.8 (Components-discrete adjunction). We have $\Upsilon.\mathrm{cc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathrm{pt}$ if and only if $\Upsilon \gg \gamma = \gamma' \in \Gamma.\mathrm{dsc} \mathbin{@} \mathrm{par}$.

Proof. If $\Upsilon.\mathrm{cc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathrm{pt}$, then $\Upsilon.\mathrm{cc}.\mathrm{dsc} \gg (\gamma : \Gamma) \otimes \mathrm{dsc} = (\gamma' : \Gamma) \otimes \mathrm{dsc} \in \Gamma.\mathrm{dsc} \mathbin{@} \mathrm{par}$ by Corollary 14.3.6. Note that $(\gamma : \Gamma) \otimes \mathrm{dsc} = \gamma$ and $(\gamma' : \Gamma) \otimes \mathrm{dsc} = \gamma'$. By the adjunction for interval substitutions, we have $\Upsilon \gg \mathrm{id}_{\Upsilon.\mathrm{cc}} \in \Upsilon.\mathrm{cc}.\mathrm{dsc} \mathbin{@} \mathrm{par}$. Hence $\Upsilon \gg \gamma = \gamma' \in \Gamma.\mathrm{dsc} \mathbin{@} \mathrm{par}$ by stability of closing substitutions.

Conversely, if $\Upsilon \gg \gamma = \gamma' \in \Gamma.\mathrm{dsc} \mathbin{@} \mathrm{par}$, then we have $\Upsilon.\mathrm{cc} \gg (\gamma : \Gamma.\mathrm{dsc}) \otimes \mathrm{cc} = (\gamma' : \Gamma.\mathrm{dsc}) \otimes \mathrm{cc} \in \Gamma.\mathrm{dsc}.\mathrm{cc} \mathbin{@} \mathrm{pt}$ by Corollary 14.3.6. By inspection we have $\Gamma.\mathrm{dsc}.\mathrm{cc} = \Gamma$, $(\gamma : \Gamma.\mathrm{dsc}) \otimes \mathrm{cc} = \gamma$, and $(\gamma' : \Gamma.\mathrm{dsc}) \otimes \mathrm{cc} = \gamma$ up to syntactic equality. Thus $\Upsilon \gg \gamma = \gamma' \in \Gamma.\mathrm{dsc} \mathbin{@} \mathrm{par}$. □

Lemma 14.3.9 (Discrete-global adjunction). We have $\Upsilon.\mathrm{dsc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathrm{par}$ if and only if $\Upsilon \gg \gamma = \gamma' \in \Gamma.\mathrm{glo} \mathbin{@} \mathrm{pt}$.

Proof. For the forward direction, we first have $\Upsilon.\mathrm{dsc}.\mathrm{glo} \gg (\gamma : \Gamma) \otimes \mathrm{glo} = (\gamma' : \Gamma) \otimes \mathrm{glo} \in \Gamma.\mathrm{glo} \mathbin{@} \mathrm{pt}$ by Corollary 14.3.6. Then $\Upsilon.\mathrm{dsc}.\mathrm{glo} = \Upsilon$, $(\gamma : \Gamma) \otimes \mathrm{glo} = \gamma$, and $(\gamma' : \Gamma) \otimes \mathrm{glo} = \gamma'$, so $\Upsilon \gg \gamma = \gamma' \in \Gamma.\mathrm{glo} \mathbin{@} \mathrm{pt}$. For the converse, we go by induction the shape of Γ and on $\Upsilon \gg \gamma = \gamma' \in \Gamma.\mathrm{glo} \mathbin{@} \mathrm{pt}$.

- Case: $\cdot$ and $\Upsilon \gg \cdot = \cdot \in \cdot \mathbin{@} \mathrm{pt}$. Immediate.
- Case: $(\Gamma, (\mu \mid a : A))$ and $\Upsilon \gg (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma.\mathrm{glo}, (\mathrm{dsc}, \mu \mid a : A)) \mathbin{@} \mathrm{pt}$ where $\Upsilon \gg \gamma = \gamma' \in \Gamma.\mathrm{glo} \mathbin{@} \mathrm{pt}$ and $\Upsilon.\mathrm{dsc}.\mu \gg M = M' \in A\gamma \mathbin{@} m$. By induction hypothesis, we have $\Upsilon.\mathrm{dsc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathrm{par}$, and so the result follows by the modal hypothesis rule.
- Case: $(\Gamma, \boldsymbol{x} : \mathbf{I})$ and $\Upsilon \gg (\gamma, \boldsymbol{r}/\boldsymbol{x}) = (\gamma', \boldsymbol{r}/\boldsymbol{x}) \in (\Gamma, \boldsymbol{x} : \mathbf{I}).\mathrm{glo} \mathbin{@} \mathrm{pt}$. By induction hypothesis, we have $\Upsilon.\mathrm{dsc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathrm{par}$, and $\Upsilon \gg \boldsymbol{r} \in \mathbf{2} \mathbin{@} \mathrm{pt}$ implies $\Upsilon.\mathrm{dsc} \gg \boldsymbol{r} \in \mathbf{I} \mathbin{@} \mathrm{par}$.

The cases for path interval, bridge endpoint, and constraint hypotheses follow the same pattern of argument. □

Finally, we prove a result aimed at generalizing Lemma 14.3.5—which says that whenever $\Upsilon.\mu \gg M \in A \mathbin{@} m$, we have $\Upsilon.\nu.(\mu \div \nu) \gg M \in A \mathbin{@} m$—to open judgments. As the open judgments are defined by closing substitutions, this result will fall out of showing that every $\Psi \Vdash \gamma = \gamma' \in \Gamma.\nu.(\mu \div \nu) \mathbin{@} m$ induces a corresponding $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mu \mathbin{@} m$. To get there, we first prove the following lemma. Item (1), which is a special case of the

property we need, is the one we are really after; the others represent a strengthening of induction hypothesis.

Lemma 14.3.10. Let $\mu : m \to n$. Then the following hold.

(1) If $n = \mathsf{par}$ and $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{glo}.\mathsf{dsc}.\mu \mathbin{@} m$, then there exist $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mu \mathbin{@} m$.

(2) If $n = \mathsf{par}$ and $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mu \mathbin{@} m$, then there exist $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mathsf{cc}.\mathsf{dsc}.\mu \mathbin{@} m$.

(3) If $n = \mathsf{pt}$ and $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mu \mathbin{@} m$, then there exist $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mathsf{dsc}.\mathsf{glo}.\mu \mathbin{@} m$.

(4) If $n = \mathsf{pt}$ and $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{dsc}.\mathsf{glo}.\mu \mathbin{@} m$, then there exist $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mu \mathbin{@} m$.

(5) If $n = \mathsf{pt}$ and $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{glo}.\mu \mathbin{@} m$, then there exist $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mathsf{cc}.\mu \mathbin{@} m$.

Moreover, in each case, we have $M\gamma_+ = M\gamma$ and $M\gamma'_+ = M\gamma'$ for any term M, up to syntactic equality.

Proof. By induction on the length of μ, proving all of the above simultaneously as follows.

(1) By cases on μ.

- $\mu = \mathsf{id}$. Then by the adjunctions on closing substitutions, we have $\Psi.\mathsf{dsc}.\mathsf{cc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{par}$, and we have $\Psi.\mathsf{dsc}.\mathsf{cc} = \Psi$.
- $\mu = (\mathsf{cc}, \mu')$. Then we are given $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{glo}.\mathsf{dsc}.\mathsf{cc}.\mu' \mathbin{@} m$. As $-.\mathsf{cc}$ cancels $-.\mathsf{dsc}$, this means we have $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{glo}.\mu' \mathbin{@} m$. It follows from (5) applied with μ' that we have some $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mu \mathbin{@} m$.
- $\mu = (\mathsf{glo}, \mu')$. Then we are given $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{glo}.\mathsf{dsc}.\mathsf{glo}.\mu' \mathbin{@} m$. It follows from (4) applied with μ' that we have some $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mu \mathbin{@} m$.

(2) By cases on μ.

- $\mu = \mathsf{id}$. Then by the action of cc on closing substitutions we have some $\Psi.\mathsf{cc} \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mathsf{cc} \mathbin{@} \mathsf{pt}$, and it follows by the components-discrete adjunction that $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mathsf{cc}.\mathsf{dsc} \mathbin{@} \mathsf{par}$.
- $\mu = (\mathsf{cc}, \mu')$. Then we are given $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{cc}.\mu' \mathbin{@} m$, and it follows that $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{cc}.\mathsf{dsc}.\mathsf{cc}.\mu' \mathbin{@} m$ because $-.\mathsf{cc}$ cancels $-.\mathsf{dsc}$ (up to syntactic equality).
- $\mu = (\mathsf{glo}, \mu')$. Then we are given $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{glo}.\mu' \mathbin{@} m$. It follows from (5) applied with μ' that $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{cc}.\mu' \mathbin{@} m$, and then we proceed as in the previous case.

(3) By cases on μ.

- $\mu = \mathsf{id}$. We have $\Psi = \Psi.\mathsf{dsc.cc}$, so $\Psi.\mathsf{dsc.cc} \Vdash \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{pt}$, and then $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{dsc.glo} \mathbin{@} \mathsf{pt}$ follows from the components-discrete and discrete-global adjunctions.
- $\mu = (\mathsf{dsc}, \mathsf{id})$. By the components-discrete adjunction, we have $\Psi.\mathsf{cc} \Vdash \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{pt}$. By the argument of the previous case, it follows that $\Psi.\mathsf{cc} \Vdash \gamma = \gamma' \in \Gamma.\mathsf{dsc.glo} \mathbin{@} \mathsf{pt}$, and then $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mathsf{dsc.glo.dsc} \mathbin{@} \mathsf{par}$ follows by applying the adjunction in reverse.
- $\mu = (\mathsf{dsc}, \mathsf{cc}, \mu')$. Then as $-.\mathsf{cc}$ cancels $-.\mathsf{dsc}$, this follows by (3) applied with μ'.
- $\mu = (\mathsf{dsc}, \mathsf{glo}, \mu')$. Then this follows by (3) applied with μ'.

(4) By cases on μ.

- $\mu = \mathsf{id}$. Then by the discrete-global and components-discrete adjunctions, it follows that $\Psi.\mathsf{dsc.cc} \Vdash \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{pt}$, and we have $\Psi.\mathsf{dsc.cc} = \Psi$.
- $\mu = (\mathsf{dsc}, \mu')$. Then this follows from (1) applied with μ'.

(5) By cases on μ.

- $\mu = \mathsf{id}$. By the discrete-global adjunction we have $\Psi.\mathsf{dsc} \Vdash \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{par}$. Then we apply the action of cc on closing substitutions and the fact that $\Psi.\mathsf{dsc.cc} = \Psi$ to get some $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mathsf{cc} \mathbin{@} \mathsf{pt}$.
- $\mu = (\mathsf{dsc}, \mu')$. Then the result follows by applying first (1) and then (2) with μ'. □

Lemma 14.3.11. Let $\mu : m \to n$ and $\nu : p \to m$ be given such that $\mu \div \nu$ is defined. If $\Psi \Vdash \gamma = \gamma' \in \Gamma.\nu.(\mu \div \nu) \mathbin{@} m$, then there exist some $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mu \mathbin{@} m$ such that $M\gamma_+ = M\gamma$ and $M\gamma'_+ = M\gamma'$ for any term M.

Proof. By induction on ν. Suppose $\Psi \Vdash \gamma = \gamma' \in \Gamma.\nu.(\mu \div \nu) \mathbin{@} m$; we have four cases.

- $\nu = \mathsf{id}$. Then $\Gamma.\nu.(\mu \div \nu) = \Gamma.\mu$.
- $\nu = (\mathsf{cc}, \nu')$. As we assumed $\mu \div \nu$ is defined, we must have $\mu = (\mathsf{cc}, \mu')$ for some μ'. Then $\Gamma.\nu.(\mu \div \nu) = \Gamma.\mathsf{cc}.\nu'.(\mu' \div \nu')$. The result thus follows by induction hypothesis applied at $\Gamma.\mathsf{cc}$, μ', and ν'.
- $\nu = (\mathsf{dsc}, \nu')$. Then $\Gamma.\nu.(\mu \div \nu) = \Gamma.\mathsf{dsc}.\nu'.((\mathsf{cc}, \mu) \div \nu')$. By induction hypothesis applied at $\Gamma.\mathsf{dsc}$, ν', and (cc, μ), we have some $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mathsf{dsc}.(\mathsf{cc}, \mu) \mathbin{@} m$, and $\Gamma.\mathsf{dsc.cc} = \Gamma$ by inspection.
- $\nu = (\mathsf{glo}, \nu')$. Then $\Gamma.\mathsf{glo}.\nu'.((\mathsf{dsc}, \mu) \div \nu')$. By induction hypothesis applied at $\Gamma.\mathsf{glo}$, ν', and (dsc, μ), we have some $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mathsf{glo}.(\mathsf{dsc}, \mu) \mathbin{@} m$. By property (1) of Lemma 14.3.10, it follows that we have some $\Psi \Vdash \gamma_{++} = \gamma'_{++} \in \Gamma.\mu \mathbin{@} m$. □

14.3.2 Contexts and substitutions

Finally, we use the properties of the extended closing substitutions to bootstrap our way to the open judgments and then general substitutions. The first main result is that $-.\mu$ preserves well-formed contexts, which follows from the division lemma foreshadowed in the previous section.

Lemma 14.3.12 (Division of open judgments). Let $\mu : m \to n$ and $\nu : p \to m$ be given and assume $\mu \div \nu$ is defined.

- Given $\Gamma.\mu \gg A = A'$ pretype $@\ m$, we have $\Gamma.\nu.(\mu \div \nu) \gg A = A'$ pretype $@\ m$.
- Given $\Gamma.\mu \gg M = M' \in A\ @\ m$, we have $\Gamma.\nu.(\mu \div \nu) \gg M = M' \in A\ @\ m$.

Proof. Without loss of generality we focus on the first property; we go by definition of the open pretype judgment. Let closing substitutions $\Psi \Vdash \gamma = \gamma' \in \Gamma.\nu.(\mu \div \nu)\ @\ m$ be given. By Lemma 14.3.11, we derive substitutions $\Psi \Vdash \gamma_+ = \gamma'_+ \in \Gamma.\mu\ @\ m$. Then by definition of $\Gamma.\mu \gg A = A'$ pretype $@\ m$, we have $\Psi \Vdash A\gamma_+ = A'\gamma'_+$ pretype $@\ m$, which is to say $\Psi \Vdash A\gamma = A'\gamma'$ pretype $@\ m$. □

Theorem 14.3.13 (Modal context operators).

$$\frac{\Gamma = \Gamma'\ \mathsf{ctx}\ @\ n \qquad \mu : m \to n}{\Gamma.\mu = \Gamma'.\mu\ \mathsf{ctx}\ @\ m}$$

Proof. By induction on $\Gamma = \Gamma'$ ctx $@\ n$, using Lemma 14.3.12 in the modal hypothesis case. □

The second essential result is the variable rule for modal hypotheses.

Lemma 14.3.14. For any $\mu : m \to n$, $(\mu \div \mu)$ is defined and does not contain glo.

Proof. After generalizing to the claim that $(\nu, \mu) \div \mu$ is defined and does not contain glo for any $\nu : n \to n$ not containing glo, this follows straightforwardly by induction on μ. □

Theorem 14.3.15 (Variable).

$$\frac{\mu : m \to n \qquad \Gamma.\mu \gg A\ \mathsf{pretype}\ @\ m}{\Gamma, (\mu \mid a : A).\mu \gg a \in A\ @\ m}$$

Proof. We have $\Gamma, (\mu \mid a : A).\mu = \Gamma.\mu, (\mu \div \mu \mid a : A)$ by Lemma 14.3.14. Note that we have $\Gamma.\mu, (\mu \div \mu \mid a : A) \gg A$ pretype $@\ m$ by weakening, which is an immediate consequence of the definitions of the open judgments. Let a closing substitution $\Psi \Vdash (\gamma, M/a) = (\gamma', M'/a) \in (\Gamma.\mu, (\mu \div \mu \mid a : A))\ @\ m$ be given; we have $\Psi \Vdash \gamma = \gamma' \in \Gamma.\mu\ @\ m$ and $\Psi.(\mu \div \mu) \Vdash M = M' \in A\gamma\ @\ m$. Using that $\mu \div \mu$ does not contain glo, we can see that $\Psi \Vdash \mathsf{id}_\Psi \in \Psi.(\mu \div \mu)\ @\ m$. By stability of the element judgment, we thus have $\Psi \Vdash M = M' \in A\gamma\ @\ m$ as needed. □

Corollary 14.3.16 (Action of modal hypotheses).

$$\frac{\Gamma', \Gamma\ \mathsf{ctx}\ @\ n \qquad \Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ n \qquad \mu : m \to n \qquad \Gamma.\mu \gg A\ \mathsf{pretype}\ @\ m}{\Gamma', (\mu \mid a : A\gamma) \gg (\gamma, a/a) = (\gamma', a/a) \in \Gamma, (\mu \mid a : A)\ @\ n}$$

Proof. By the substitution formation rule and variable rule. □

Corollary 14.3.17 (Identity substitution).

$$\frac{\Gamma = \Gamma'\ \mathsf{ctx}\ @\ m}{\Gamma \gg \mathsf{id}_\Gamma = \mathsf{id}_{\Gamma'} \in \Gamma\ @\ m}$$

Proof. By induction on $\Gamma = \Gamma'\ \mathsf{ctx}\ @\ m$, using the action of each context constructor. □

The remaining properties of the open judgments now require little ingenuity to verify, being for the most part a rehash of the corresponding properties of extended closing substitutions. We leave the construction of detailed proofs as an exercise to the reader.

Proposition 14.3.18 (Instantiation of substitutions). If $\Psi \Vdash \delta = \delta' \in \Gamma'\ @\ m$ and $\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ m$, then $\Psi \Vdash \gamma\delta = \gamma'\delta' \in \Gamma\ @\ m$.

Proof. By induction on the derivation of $\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ m$. In the case of a modal hypothesis, we use the functorial action of modalities on extended closing substitutions.□

Corollary 14.3.19 (Stability of open typing judgments). Given $\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ m$ and $\Gamma \gg A = A'$ pretype $@\ m$, we have $\Gamma' \gg A\gamma = A'\gamma'$ pretype $@\ m$; likewise for types and terms.

Proposition 14.3.20 (Action by modalities).

$$\frac{\Gamma' \gg \gamma = \gamma' \in \Gamma\ @\ n \qquad \mu : m \to n}{\Gamma'.\mu \gg (\gamma : \Gamma) \otimes \mu = (\gamma' : \Gamma) \otimes \mu \in \Gamma.\mu\ @\ m}$$

Proof. Following the proof of Corollary 14.3.6, now using Lemma 14.3.12. □

Proposition 14.3.21 (Components-discrete adjunction).

$$\frac{\Gamma'.\mathsf{cc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{pt}}{\Gamma' \gg \gamma = \gamma' \in \Gamma.\mathsf{dsc} \mathbin{@} \mathsf{par}} \qquad \frac{\Gamma' \gg \gamma = \gamma' \in \Gamma.\mathsf{dsc} \mathbin{@} \mathsf{par}}{\Gamma'.\mathsf{cc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{pt}}$$

Proof. Following the proof of Lemma 14.3.8. □

Proposition 14.3.22 (Discrete-global adjunction).

$$\frac{\Gamma'.\mathsf{dsc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{par}}{\Gamma' \gg \gamma = \gamma' \in \Gamma.\mathsf{glo} \mathbin{@} \mathsf{pt}} \qquad \frac{\Gamma' \gg \gamma = \gamma' \in \Gamma.\mathsf{glo} \mathbin{@} \mathsf{pt}}{\Gamma'.\mathsf{dsc} \gg \gamma = \gamma' \in \Gamma \mathbin{@} \mathsf{par}}$$

Proof. Following the proof of Lemma 14.3.9. □

Proposition 14.3.23 (Stability/composition of substitutions). If $\Gamma'' \gg \gamma'' = \gamma''' \in \Gamma' \mathbin{@} m$ and $\Gamma' \gg \gamma = \gamma' \in \Gamma \mathbin{@} m$, then $\Gamma'' \gg \gamma''\gamma = \gamma'''\gamma' \in \Gamma \mathbin{@} m$.

Proof. Following the proof of Proposition 14.3.18, using the action of modalities on substitutions and stability of open typing judgments in the modal hypothesis case. □

14.4 Modal types

With the judgmental apparatus sorted, we now construct a specific type system with types corresponding to the discrete, global, and codiscrete cohesion functors.

To avoid repetition, we introduce a uniform notation for modal types: we write $\langle \mu \,|\, A \rangle$ for the type corresponding to the right adjoint of each $\mu \in \{\mathsf{cc}, \mathsf{dsc}, \mathsf{glo}\}$. Thus we have the following encodings.

$$\mathsf{Disc}(A) :\equiv \langle \mathsf{cc} \,|\, A \rangle \qquad \mathsf{Glo}(A) :\equiv \langle \mathsf{dsc} \,|\, A \rangle \qquad \mathsf{Codisc}(A) :\equiv \langle \mathsf{glo} \,|\, A \rangle$$

This notation reflects the role of the left adjoint in the intended introduction rule for each modal type.

$$\frac{\Gamma.\mu \gg M \in A}{\Gamma \gg \mathsf{mod}(M) \in \langle \mu \,|\, A \rangle}$$

Note also the parallel with the modal hypothesis notation $(\mu \mid a : A)$. We display the operational semantics rules for modal types in Figures 14.4 and 14.5.

It will be useful to first give names to the relations that will interpret these types.

Formation

$$\frac{\mu \in \{\mathsf{cc}, \mathsf{dsc}, \mathsf{glo}\}}{\langle \mu \mid A \rangle \ \mathsf{val}}$$

Introduction

$$\frac{}{\mathsf{mod}(M) \ \mathsf{val}}$$

Projection

$$\frac{P \longmapsto P'}{\mathsf{unmod}(P) \longmapsto \mathsf{unmod}(P')} \qquad \frac{}{\mathsf{unmod}(\mathsf{mod}(M)) \longmapsto M}$$

Discrete elimination

$$\frac{P \longmapsto P'}{\mathsf{letdisc}(d.B, P, a.N) \longmapsto \mathsf{letdisc}(d.B, P', a.N)} \qquad \frac{}{\mathsf{letdisc}(d.B, \mathsf{mod}(M), a.N) \longmapsto N[M/a]}$$

$$\frac{}{\begin{array}{c}\mathsf{letdisc}(d.B, \mathsf{fhcom}^{r \to s}(P; \overrightarrow{\xi_i \hookrightarrow x.P_i}), a.N) \\ \longmapsto \\ \mathsf{com}^{r \to s}_{x.B[\mathsf{fhcom}^{r \to x}(P; \overrightarrow{\xi_i \hookrightarrow x.P_i})/d]}(\mathsf{letdisc}(d.B, P, a.N); \overrightarrow{\xi_i \hookrightarrow \mathsf{letdisc}(d.B, P_i, a.N)})\end{array}}$$

Splitting

$$\frac{}{\mathsf{split}_{\mathbf{0}}(M_0, M_1) \longmapsto M_0} \qquad \frac{}{\mathsf{split}_{\mathbf{1}}(M_0, M_1) \longmapsto M_1}$$

Figure 14.4: Operational semantics for modal parametric type theory: formation, introduction, elimination, splitting

Definition 14.4.1. Let Ψ ictx @ n and $\mu : m \to n$. Given a value $(m,\Psi.\mu)$-relation R, we define a value (n,Ψ)-relation $Mod_\mu(R)$ for $\Psi' \Vdash \psi \in \Psi$.

$$V \approx V' \in Mod_\mu(R)\langle\psi\rangle :\Longleftrightarrow \begin{cases} V = \mathsf{mod}(M) \text{ and } V' = \mathsf{mod}(M') \\ \text{with } M \approx M' \in \Downarrow R[(\psi : \Psi) \otimes \mu] \end{cases}$$

For $\mathrm{Glo}(A)$ and $\mathrm{Codisc}(A)$, the above will be their defining relation. These two types support projection rules that invert the introduction rule, a setup that reflects the status of dsc and glo as right adjoints. As such, the Kan operations for these types are easily implemented: as with functions or products, we unpack the underlying elements of A, coerce or compose them, and repackage.

Coercion

$$\frac{\mu \in \{\mathsf{dsc}, \mathsf{glo}\}}{\mathsf{coe}^{r \to s}_{x.\langle \mu | A \rangle}(P) \longmapsto \mathsf{mod}(\mathsf{coe}^{r \to s}_{x.A}(\mathsf{unmod}(P')))}$$

$$\frac{P \longmapsto P'}{\mathsf{coe}^{r \to s}_{x.\langle \mathsf{cc} | A \rangle}(P) \longmapsto \mathsf{coe}^{r \to s}_{x.\langle \mathsf{cc} | A \rangle}(P')} \qquad \frac{}{\mathsf{coe}^{r \to s}_{x.\langle \mathsf{cc} | A \rangle}(\mathsf{mod}(M)) \longmapsto \mathsf{mod}(\mathsf{coe}^{r \to s}_{x.A}(M))}$$

$$\frac{}{\begin{array}{c}\mathsf{coe}^{r \to s}_{x.\langle \mathsf{cc} | A \rangle}(\mathsf{fhcom}^{t \to u}(P; \overrightarrow{\xi_i \hookrightarrow y.P_i})) \\ \longmapsto \\ \mathsf{fhcom}^{t \to u}(\mathsf{coe}^{r \to s}_{x.\langle \mathsf{cc} | A \rangle}(P); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{coe}^{r \to s}_{x.\langle \mathsf{cc} | A \rangle}(P_i)})\end{array}}$$

Composition

$$\frac{\mu \in \{\mathsf{dsc}, \mathsf{glo}\}}{\mathsf{hcom}^{r \to s}_{\langle \mu | A \rangle}(P; \overrightarrow{\xi_i \hookrightarrow x.P_i}) \longmapsto \mathsf{mod}(\mathsf{hcom}^{r \to s}_{A}(\mathsf{unmod}(P); \overrightarrow{\xi_i \hookrightarrow x.\mathsf{unmod}(P_i)}))}$$

$$\frac{}{\mathsf{hcom}^{r \to s}_{\langle \mathsf{cc} | A \rangle}(P; \overrightarrow{\xi_i \hookrightarrow x.P_i}) \longmapsto \mathsf{fhcom}^{r \to s}(P; \overrightarrow{\xi_i \hookrightarrow x.P_i})}$$

Formal composites

$$\frac{r \neq s \qquad (\nexists i)\ \xi_i \text{ satisfied}}{\mathsf{fhcom}^{r \to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i})\ \mathsf{val}} \qquad \frac{(\nexists i)\ \xi_i \text{ satisfied}}{\mathsf{fhcom}^{r \to r}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto M}$$

$$\frac{(\nexists i < k)\ \xi_i \text{ satisfied} \qquad \xi_k \text{ satisfied}}{\mathsf{fhcom}^{r \to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \longmapsto N_k[s/x]}$$

Figure 14.5: Operational semantics for modal parametric type theory: Kan operations

For $\mathsf{Disc}(A)$, on the other hand, the relation Mod_{cc} alone does not provide enough elements to implement the Kan operations. Consider that whenever we have a value of the form $\Psi \Vdash \mathsf{mod}(M) \in \mathsf{Disc}(A)$, the term M has type $\Psi.\mathsf{cc} \Vdash M \in A$; as such, M cannot depend on any bridge interval hypotheses in Ψ. For $\mathsf{Disc}(A)$ to implement the Kan operations, however, it must contain elements that depend non-trivially on such hypotheses. In particular, we can use loosen (Definition 10.3.1) to create bridges from (possibly non-degenerate) paths.

$$\frac{P \in \mathsf{Path}(\mathsf{Disc}(A), N_0, N_1)}{\mathsf{loosen}_{\mathsf{Disc}(A)}\, P \in \mathsf{Bridge}(\mathsf{Disc}(A), N_0, N_1)}$$

The discrete type must therefore contain additional values. The values we introduce are *formal composite values*: rather than evaluating in some way, composites in the discrete type will be inert values. These will be familiar to the reader who has been through Part II—see Section 5.1 for their motivation in that case—but we re-introduce them below. As a consequence, the discrete type will fail to support a projection with an exact uniqueness principle, containing as it does values other than mod terms. Instead, we will have a dependent eliminator that extends maps $(\mathsf{cc} \mid a : A) \gg N \in B[\mathsf{mod}(a)/d]$ to maps $d : \mathsf{Disc}(A) \gg \mathsf{letdisc}(d.B, d, a.N) \in B$.

The operational semantics for formal composites are replicated in Figure 14.5: they are values unless one of their boundary conditions is satisfied, in which case they reduce. The following defines the relation $Fhcom(R)$ of formal composites formed in elements of $\Downarrow R$.

Definition 14.4.2 (Replica of Definition 6.2.10). Given a Ψ-relation R, we define a Ψ-relation $Fhcom(R)$ as inductively generated by the principle that, for each $\Psi' \Vdash \psi \in \Psi$, we have $\mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) \approx \mathsf{fhcom}^{r\to s}(M'; \overrightarrow{\xi_i \hookrightarrow x.N'_i}) \in Fhcom(R)\langle\psi\rangle$ whenever the following hold.

- $\Psi' \Vdash r, s \in \mathbb{I}$ with $r \neq s$.
- $M \approx M' \in \Downarrow R\psi$.
- $\Psi' \Vdash \xi_i \in \mathbb{F}$ for each i, and there is no ξ_i such that $\Psi' \Vdash \xi_i$ satisfied holds.
- $\Psi', \xi_i, \xi_j, x : \mathbb{I} \gg N_i \approx N'_j \in \Downarrow R\psi$ for all i, j.
- $\Psi', \xi_i \gg M \approx N_i[r/x] \in \Downarrow R\psi$ for all i.

The relation interpreting $\mathsf{Disc}(A)$ is then obtained by closing $Mod_{\mathsf{cc}}(\llbracket A \rrbracket)$ under formal composites, where $\llbracket A \rrbracket$ is the relation represented by A.

Example 14.4.3 (Small type system). We define an operator Mo on candidate type systems as follows: given τ, $Mo(\tau)$ is the union of the following clauses.

- $Mo(\tau) \vDash \Psi \Vdash \langle \mu \mid A \rangle \approx \langle \mu \mid A' \rangle \downarrow R \mathbin{@} n$ for $\mu : m \to n$ with $\mu \in \{\mathrm{dsc}, \mathrm{glo}\}$ whenever
  - $A \approx A' \in \Downarrow\tau[S]$ for some $\Psi.\mu$-PER S,
  - $R = Mod_\mu(S)$.
- $Mo(\tau) \vDash \Psi \Vdash \langle \mathrm{cc} \mid A \rangle \approx \langle \mathrm{cc} \mid A' \rangle \downarrow R \mathbin{@} \mathrm{par}$ whenever
  - $A \approx A' \in \Downarrow\tau[S]$ for some $\Psi.\mathrm{cc}$-PER S,
  - R is the least fixed-point of the operator $R \mapsto Mod_{\mathrm{cc}}(R) \cup Fhcom(R)$.

We define the candidate type system τ_0^{Mo} to be the least fixed point of the following operator, where F, H, IP are as defined in Examples 3.1.32, 6.2.22 and 9.1.13 respectively.

$$\tau \mapsto \left(\bigcup_{m \in \{\mathrm{pt},\mathrm{par}\}} F(\tau_m)\right) \cup \left(\bigcup_{m \in \{\mathrm{pt},\mathrm{par}\}} H(\tau_m)\right) \cup IP(\tau_{\mathrm{par}}) \cup Mo(\tau)$$

That is, we include the basic cubical (F) and higher inductive (H) type formers in both the pointwise and parametric modes, but restrict the Bridge and Gel types (IP) to the parametric mode.

We can construct a larger type system τ_1^{Mo} closed under these type formers and containing τ_0^{Mo} as a universe in the usual fashion. We henceforth assume we are working in such a type system.

For the first couple of rules—pretype formation and mod introduction—we can treat the three modal types uniformly.

Rule 14.4.4 (Pretype formation). The following rule is validated for $\mu \in \{\mathrm{cc}, \mathrm{dsc}, \mathrm{glo}\}$ with $\mu : m \to n$.

$$\frac{\Psi.\mu \gg A = A' \ \mathsf{type} \mathbin{@} m}{\Psi \Vdash \langle \mu \mid A \rangle = \langle \mu \mid A' \rangle \ \mathsf{pretype} \mathbin{@} n}$$

Proof. Immediate by coherent value introduction. We use the action of μ on substitutions: for any $\Psi' \Vdash \psi \in \Psi \mathbin{@} n$, we have $\Psi'.\mu \gg (\psi : \Psi) \otimes \mu \in \Psi \mathbin{@} m$, thus $\Psi'.\mu \gg A\psi = A'\psi \ \mathsf{type} \mathbin{@} m$ and therefore $\tau_0^{Mo} \vDash \Psi' \Vdash \langle \mu \mid A\psi \rangle \approx \langle \mu \mid A'\psi \rangle \downarrow R \mathbin{@} n$ for the appropriate R. □

Rule 14.4.5 (Introduction). The following rule is validated for $\mu \in \{\mathrm{cc}, \mathrm{dsc}, \mathrm{glo}\}$ with $\mu : m \to n$.

$$\frac{\Psi.\mu \gg M = M' \in A \mathbin{@} m}{\Psi \Vdash \mathsf{mod}(M) = \mathsf{mod}(M') \in \langle \mu \mid A \rangle \mathbin{@} n}$$

Proof. By coherent value introduction, following the proof of pretype formation. □

For the remainder of the rules—elimination, reduction and uniqueness equations, and the Kan operations—we must handle $\langle \mu \mid A \rangle$ for $\mu = \mathsf{dsc}$ separately from the two right adjoints dsc and glo.

14.4.1 Right adjoint modalities

The type formers Glo and Codisc do not only have left adjoint context operators; those left adjoints are themselves right adjoints, to cc and dsc respectively. This enables a *negative* treatment of Glo and Codisc, that is, one characterized by a projection operator and uniqueness principle rather than an induction principle. An analogous situation appears in Shulman's cohesive type theory: his ♯ operator, which corresponds to the composite $\mathsf{Codisc}(\mathsf{Glo}(-))$, is axiomatized negatively. We treat the two type formers uniformly by introducing the following shorthand.

Definition 14.4.6. Given $\mu \in \{\mathsf{dsc}, \mathsf{glo}\}$, define ${}^{\dashv}\mu$ as follows.

$$
\begin{aligned}
{}^{\dashv}\mathsf{dsc} &\coloneqq \mathsf{cc} \\
{}^{\dashv}\mathsf{glo} &\coloneqq \mathsf{dsc}
\end{aligned}
$$

Note that ${}^{\dashv}\mu = \mathsf{id} \div \mu$, where division is as specified in Definition 14.3.4.

In the following, it may be useful to notice the similarity to bridge types: if we think of the context operator $-.\mu$ as analogous to $\boldsymbol{x}{:}\mathbf{I}$, then $-.{}^{\dashv}\mu$ corresponds to interval restriction. Modulo the absence of endpoint constraints in the type and the binding of an interval variable, the projection rules for bridge and negative modal types then match exactly.

Rules 14.4.7 (Projection). The following rules are validated for any $\mu \in \{\mathsf{dsc}, \mathsf{glo}\}$ with $\mu : m \to n$.

$$\frac{\Psi.{}^{\dashv}\mu.\mu \gg A\ \mathsf{type} \mathbin{@} m \qquad \Psi.{}^{\dashv}\mu \gg P = P' \in \langle \mu \mid A \rangle \mathbin{@} n}{\Psi \Vdash \mathsf{unmod}(P) = \mathsf{unmod}(P') \in A \mathbin{@} m}$$

$$\frac{\Psi.{}^{\dashv}\mu.\mu \gg A\ \mathsf{type} \mathbin{@} m \qquad \Psi.{}^{\dashv}\mu.\mu \gg M \in A \mathbin{@} m}{\Psi \Vdash \mathsf{unmod}(\mathsf{mod}(M)) = M \in A \mathbin{@} m}$$

$$\frac{\Psi.\mu \gg A\ \mathsf{type} \mathbin{@} m \qquad \Psi \Vdash P \in \langle \mu \mid A \rangle \mathbin{@} n}{\Psi \Vdash P = \mathsf{mod}(\mathsf{unmod}(P)) \in \langle \mu \mid A \rangle \mathbin{@} n}$$

Proof. First, note that $\Psi \Vdash \mathsf{id}_{\Psi.\dashv\mu} \in \Psi.\dashv\mu.\mu \mathbin{@} m$, either by the components-discrete adjunction or the discrete-global adjunction. The hypothesis $\Psi.\dashv\mu.\mu \gg A\ \mathsf{type} \mathbin{@} m$ thus implies that $\Psi \Vdash A\ \mathsf{type} \mathbin{@} m$, justifying the use of A in the conclusions of the first two rules.

As usual, we prove the reduction rule first. As with A, we deduce $\Psi \Vdash M \in A \mathbin{@} m$ from $\Psi.\dashv\mu.\mu \gg M \in A \mathbin{@} m$ by stability. The rule is then immediate by coherent head expansion, as $\mathsf{unmod}(\mathsf{mod}(M))\psi \longmapsto M\psi$ for any ψ.

For the first rule, we have that unmod is eager, so we apply Lemma 3.1.38 to reduce to the case where P and P' are values of $\langle \mu \mid A \rangle$; the result then follows by applying the reduction rule on either side.

For the final rule, we first use Lemma 3.1.36 to see that $\Psi \Vdash P = \mathsf{mod}(M) \in \langle \mu \mid A \rangle \mathbin{@} n$ for some $\Psi.\mu \gg M \in A \mathbin{@} m$. Again using the adjunction from above, we have the substitution $\Psi.\mu.\dashv\mu \gg \mathsf{id}_{\Psi.\mu} \in \Psi \mathbin{@} n$. By stability, we thus have $\Psi.\mu.\dashv\mu \gg P = \mathsf{mod}(M) \in \langle \mu \mid A \rangle \mathbin{@} n$. Applying the unmod rules just proven, we obtain first $\Psi.\mu \gg \mathsf{unmod}(P) = \mathsf{unmod}(\mathsf{mod}(M)) \in A \mathbin{@} m$ and thereby $\Psi.\mu \gg \mathsf{unmod}(P) = M \in A \mathbin{@} m$. Applying the introduction rule then gives $\Psi \Vdash \mathsf{mod}(\mathsf{unmod}(P)) = \mathsf{mod}(M) \in \langle \mu \mid A \rangle \mathbin{@} n$, from which the result follows by combination with $\Psi \Vdash P = \mathsf{mod}(M) \in \langle \mu \mid A \rangle \mathbin{@} n$. □

Rule 14.4.8 (Type formation). The following rule is validated for $\mu \in \{\mathsf{dsc}, \mathsf{glo}\}$ with $\mu : m \to n$.

$$\frac{\Psi.\mu \gg A = A'\ \mathsf{type} \mathbin{@} m}{\Psi \Vdash \langle \mu \mid A \rangle = \langle \mu \mid A' \rangle\ \mathsf{type} \mathbin{@} n}$$

Proof. As the reductions for coercion and composition in $\langle \mu \mid A \rangle$ are stable under interval substitution, it suffices to check that the reducts are well-typed and satisfy the necessary boundary equations; the results then follow straightforwardly from coherent head expansion.

For coercion, suppose we have $(\Psi, x : \mathbb{I}).\mu \gg A\ \mathsf{type} \mathbin{@} m$, $\Psi \Vdash r, s \in \mathbb{I} \mathbin{@} n$, and $\Psi \Vdash P \in \langle \mu \mid A \rangle[r/x] \mathbin{@} n$. Reindexing $\Psi \Vdash P \in \langle \mu \mid A \rangle[r/x] \mathbin{@} n$ along the substitution $\Psi.\mu.\dashv\mu \gg \mathsf{id}_{\Psi.\mu} \in \Psi \mathbin{@} n$ and then applying the projection rule gives $\Psi.\mu \gg \mathsf{unmod}(P) \in A[r/x] \mathbin{@} m$. By coercion in A, we then have $\Psi.\mu \gg \mathsf{coe}^{r \to s}_{x.A}(\mathsf{unmod}(P)) \in A[s/x] \mathbin{@} m$. Hence $\Psi \gg \mathsf{mod}(\mathsf{coe}^{r \to s}_{x.A}(\mathsf{unmod}(P))) \in \langle \mu \mid A \rangle[s/x] \mathbin{@} n$ by the introduction rule. Finally, in the case $r = s$, this is equal to P by reduction of trivial coercions in A and the uniqueness rule.

For composition, suppose we have $\Psi.\mu \gg A\ \mathsf{type} \mathbin{@} m$, $\Psi \Vdash r, s \in \mathbb{I} \mathbin{@} n$, $\Psi \Vdash P \in \langle \mu \mid A \rangle \mathbin{@} n$, $\Psi \gg \xi_i \in \mathbb{F} \mathbin{@} n$ for all i, and $\Psi, x : \mathbb{I}, \xi_i \Vdash P_i \in \langle \mu \mid A \rangle \mathbin{@} n$ for all i satisfying the equations required of the cap and tube of a composite. Again reindexing along $\Psi.\mu.\dashv\mu \gg \mathsf{id}_{\Psi.\mu} \in \Psi \mathbin{@} n$ and applying the projection, we have $\Psi.\mu \Vdash \mathsf{unmod}(P) \in A \mathbin{@} m$ and $(\Psi, x : \mathbb{I}, \xi_i).\mu \Vdash P_i \in A \mathbin{@} m$ for each i. Recall that for $\mu \in \{\mathsf{dsc}, \mathsf{glo}\}$, we have $(\Psi, x : \mathbb{I}, \xi_i).\mu = \Psi.\mu, x : \mathbb{I}, \xi_i$. We can therefore form a composite in A, the term $\Psi.\mu \gg$

$\mathsf{hcom}_A^{r\to s}(\mathsf{unmod}(P); \overrightarrow{\xi_i \hookrightarrow x.\mathsf{unmod}(P_i)}) \in A \ @\ m$. Finally, we apply the introduction rule to obtain the composite in $\langle \mu \mid A \rangle$, which clearly has the necessary boundary by the equations for the composite in A. □

14.4.2 The discrete type

Unlike the other two modal types, the discrete type is defined by a modality cc with no left adjoint, so the previous approach is unavailable. Instead, we give a dependent elimination principle, a case analysis operator analogous to that used for inductive types. Again, there is an analogy to Shulman's presentation of cohesion, wherein the $\flat$ operator corresponding to the composite $\mathsf{Disc}(\mathsf{Glo}(-))$ is axiomatized in a positive style.

The first step is to confirm that formal composites are also elements of the type, which in turn implies that the type supports composition.

Rule 14.4.9 (Formal composites in the discrete type). Given $\Psi.\mathsf{cc} \gg A\ \mathsf{type}\ @\ \mathsf{pt}$, $\Psi \Vdash r, s \in \mathbb{I}\ @\ \mathsf{par}$ and $\Psi \Vdash \xi_i \in \mathbb{F}\ @\ \mathsf{par}$ for all i, the following rules are validated.

$$\frac{\Psi \Vdash M = M' \in \mathsf{Disc}(A)\ @\ \mathsf{par} \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \gg N_i = N'_j \in \mathsf{Disc}(A)\ @\ \mathsf{par} \\ (\forall i)\ \Psi, \xi_i \gg M = N_i[r/x] \in \mathsf{Disc}(A)\ @\ \mathsf{par}}{\Vdash \mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = \mathsf{fhcom}^{r\to s}(M'; \overrightarrow{\xi_i \hookrightarrow x.N'_i}) \in \mathsf{Disc}(A)\ @\ \mathsf{par}}$$

$$\frac{\Psi \Vdash M \in \mathsf{Disc}(A)\ @\ \mathsf{par} \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \gg N_i = N_j \in \mathsf{Disc}(A)\ @\ \mathsf{par} \\ (\forall i)\ \Psi, \xi_i \gg M = N_i[r/x] \in \mathsf{Disc}(A)\ @\ \mathsf{par}}{\Psi \Vdash \mathsf{fhcom}^{r\to r}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = M \in \mathsf{Disc}(A)\ @\ \mathsf{par}}$$

$$\frac{\Psi \Vdash \xi_k\ \mathsf{satisfied}\ @\ \mathsf{par} \\ \Psi \Vdash M \in \mathsf{Disc}(A)\ @\ \mathsf{par} \qquad (\forall i, j)\ \Psi, \xi_i, \xi_j, x : \mathbb{I} \gg N_i = N_j \in \mathsf{Disc}(A)\ @\ \mathsf{par} \\ (\forall i)\ \Psi, \xi_i \gg M = N_i[r/x] \in \mathsf{Disc}(A)\ @\ \mathsf{par}}{\Psi \Vdash \mathsf{fhcom}^{r\to s}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) = N_k[s/x] \in \mathsf{Disc}(A)\ @\ \mathsf{par}}$$

Proof. A by-now standard argument by coherent introduction and head expansion. For details, see the proof of the more general Lemma 6.2.15 in Part II. □

Lemma 14.4.10 (Composition). $\Psi \Vdash \mathsf{Disc}(A) = \mathsf{Disc}(A')\ \mathsf{pretype}\ @\ \mathsf{par}$ support composition for any $\Psi.\mathsf{cc} \gg A = A'\ \mathsf{type}\ @\ \mathsf{pt}$.

Proof. This follows as a corollary of Rule 14.4.9 by coherent head expansion: composites in the discrete type reduce to formal composites, which are well-typed and satisfy the necessary boundary equations. □

Coercion is more involved, as it analyzes the value of its input. In particular, because we have included formal composites as values of the discrete type, the coercion operator at the discrete type must also handle these elements. Fortunately, this is straightforwardly resolved: a coercion applied to a formal composite reduces to a formal composite of coercions. (This is the same as the reduction used for higher inductive types in Part II.)

Lemma 14.4.11 (Coercion). $\Psi \Vdash \mathsf{Disc}(A) = \mathsf{Disc}(A')$ pretype @ par support coercion for any $\Psi.\mathsf{cc} \gg A = A'$ type @ pt.

Proof. We define a value Ψ-PER Coe^{-1} by declaring that $V \approx V' \in Coe^{-1}\langle\psi\rangle$ holds for $\Psi' \Vdash \psi \in \Psi$ exactly when $\Psi' \Vdash V = V' \in \mathsf{Disc}(A)$ and the following are satisfied for all $\Psi', x : \mathbb{I} \Vdash \psi_x \in \Psi$ and $\Psi' \Vdash r, s \in \mathbb{I}$ such that $\psi_x[r/x] = \psi$.

- $\Psi' \Vdash \mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A)\psi_x}(V) = \mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A')\psi_x}(V') \in \mathsf{Disc}(A)\psi_x[s/x]$ @ par.
- $\Psi' \Vdash \mathsf{coe}^{r\to r}_{x.\mathsf{Disc}(A)\psi_x}(V) = V \in \mathsf{Disc}(A)\psi$ @ par.

Note that for any terms $N \approx N' \in \Downarrow Coe^{-1}\psi$ and ψ_x, r, s as above, we can deduce that $\Psi' \Vdash \mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A)\psi_x}(N) = \mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A')\psi_x}(N') \in \mathsf{Disc}(A)\psi_x[s/x]$ @ par and $\Psi' \Vdash \mathsf{coe}^{r\to r}_{x.\mathsf{Disc}(A)\psi_x}(N) = N \in \mathsf{Disc}(A)\psi$ @ par. This follows by Lemma 3.1.38, as coercion at the discrete type is an eager operator.

We aim to show that $[\![\mathsf{Disc}(A)]\!] \subseteq Coe^{-1}$. By definition of the former as a least fixed-point, it suffices to show that $Mod_{\mathsf{cc}}([\![A]\!]) \cup Fhcom(Coe^{-1}) \subseteq Coe^{-1}$, which is to say that $Mod_{\mathsf{cc}}([\![A]\!]) \subseteq Coe^{-1}$ and $Fhcom(Coe^{-1}) \subseteq Coe^{-1}$.

Given values $\mathsf{mod}(M) \approx \mathsf{mod}(M') \in Mod_{\mathsf{cc}}([\![A]\!])\psi$ and ψ_x, r, s as above, we have by head expansion and coercion in A that $\mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A)\psi_x}(\mathsf{mod}(M)) = \mathsf{mod}(\mathsf{coe}^{r\to s}_{x.A\psi_x}(M)) \in \mathsf{Disc}(A)\psi_x[s/x]$, likewise for M'. It follows that $Mod_{\mathsf{cc}}([\![A]\!]) \subseteq Coe^{-1}$.

As for $Fhcom(Coe^{-1}) \subseteq Coe^{-1}$, suppose we are given a pair of values in the former relation, $\mathsf{fhcom}^{t\to u}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \approx \mathsf{fhcom}^{t\to u}(M'; \overrightarrow{\xi_i \hookrightarrow y.N'_i}) \in Fhcom(Coe^{-1})\langle\psi\rangle$, and ψ_x, r, s, as above. By definition of *Fhcom* and the properties of terms in Coe^{-1}, the argument terms M, M', N_i, N'_i may be coerced to obtain well-typed elements of $\mathsf{Disc}(A')\psi_x[s/x]$, then assembled into well-typed formal composites by Rule 14.4.9. That is, the following are well-typed and moreover equal in $\mathsf{Disc}(A)\psi_x[s/x]$.

$$\mathsf{fhcom}^{t\to u}(\mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A)\psi_x}(M); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A)\psi_x}(N_i)}) \in \mathsf{Disc}(A)\psi_x[s/x] \text{ @ par}$$
$$\mathsf{fhcom}^{t\to u}(\mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A')\psi_x}(M'); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A')\psi_x}(N'_i)}) \in \mathsf{Disc}(A)\psi_x[s/x] \text{ @ par}$$

It now follows by the definition of the operational semantics and coherent head expansion that the term $\mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A)\psi_x}(\mathsf{fhcom}^{t\to u}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}))$ is equal to the former, likewise

$\mathsf{coe}^{r\to s}_{x.\mathsf{Disc}(A')\psi_x}(\mathsf{fhcom}^{t\to u}(M'; \overrightarrow{\xi_i \hookrightarrow y.N'_i}))$ to the latter. Thus the two formal composites are coercible, and we can also see that coercion $r \to r$ produces an term equal to the input. Hence $\mathit{Fhcom}(\mathit{Coe}^{-1}) \subseteq \mathit{Coe}^{-1}$ as required. □

Rule 14.4.12 (Type formation).

$$\frac{\Psi.\mathsf{cc} \gg A = A' \ \mathsf{type} \ @ \ m}{\Psi \Vdash \mathsf{Disc}(A) = \mathsf{Disc}(A') \ \mathsf{type} \ @ \ n}$$

Proof. By Lemmas 14.4.10 and 14.4.11. □

Finally, we have the elimination rule for the discrete type.

Rules 14.4.13 (Discrete elimination). The following hold for any $\Psi.\mathsf{cc} \gg A \ \mathsf{type} \ @ \ \mathsf{pt}$ and $\Psi, d : \mathsf{Disc}(A) \gg B = B' \ \mathsf{type} \ @ \ \mathsf{par}$.

$$\frac{\Psi \Vdash P = P' \in \mathsf{Disc}(A) \ @ \ \mathsf{par} \qquad \Psi, (\mathsf{cc} \mid a : A) \gg N = N' \in B[\mathsf{mod}(a)/d] \ @ \ \mathsf{par}}{\Psi \Vdash \mathsf{letdisc}(d.B, P, a.N) = \mathsf{letdisc}(d.B', P', a.N') \in B[P/d] \ @ \ \mathsf{par}}$$

$$\frac{\Psi.\mathsf{cc} \Vdash M \in A \ @ \ \mathsf{pt} \qquad \Psi, (\mathsf{cc} \mid a : A) \gg N \in B[\mathsf{mod}(a)/d] \ @ \ \mathsf{par}}{\Psi \Vdash \mathsf{letdisc}(d.B, \mathsf{mod}(M), a.N) = N[M/a] \in B[\mathsf{mod}(M)/d] \ @ \ \mathsf{par}}$$

$$\frac{\begin{array}{c}\Psi \gg r, s \in \mathbb{I} \ @ \ \mathsf{par} \qquad \Psi \Vdash P \in \mathsf{Disc}(A) \ @ \ \mathsf{par} \\ (\forall i) \ \Psi \Vdash \xi_i \in \mathbb{F} \ @ \ \mathsf{par} \qquad (\forall i, j) \ \Psi, x : \mathbb{I}, \xi_i, \xi_j \gg P_i = P_j \in \mathsf{Disc}(A) \ @ \ \mathsf{par} \\ (\forall i) \ \Psi, \xi_i \gg P = P_i[r/x] \in \mathsf{Disc}(A) \ @ \ \mathsf{par} \\ F_x \coloneqq \mathsf{fhcom}^{r\to s}(P; \overrightarrow{\xi_i \hookrightarrow x.P_i}) \qquad \Psi, (\mathsf{cc} \mid a : A) \gg N \in B[\mathsf{mod}(a)/d] \ @ \ \mathsf{par} \\ T \coloneqq \mathsf{com}^{r\to s}_{x.B[F_x/d]}(\mathsf{letdisc}(d.B, P, a.N); \overrightarrow{\xi_i \hookrightarrow \mathsf{letdisc}(d.B, P_i, a.N)})\end{array}}{\Psi \Vdash \mathsf{letdisc}(d.B, F_s, a.N) = T \in B[F_s/d] \ @ \ \mathsf{par}}$$

Proof. We define a Ψ-relation Elim^{-1} by declaring that $V \approx V' \in \mathit{Elim}^{-1}\langle\psi\rangle$ whenever $\Psi \Vdash \mathsf{letdisc}(d.B\psi, V, a.N\psi) = \mathsf{letdisc}(d.B'\psi, V', a.N'\psi) \in B\psi[V/d] \ @ \ \mathsf{par}$ and $\Vdash V = V' \in \mathsf{Disc}(A)\psi \ @ \ \mathsf{par}$ hold. By Lemma 3.1.38, we have that $\Psi \Vdash \mathsf{letdisc}(d.B\psi, P, a.N\psi) = \mathsf{letdisc}(d.B'\psi, P', a.N'\psi) \in B\psi[P/d] \ @ \ \mathsf{par}$ for $P \approx P' \in \Downarrow \mathit{Elim}^{-1}\psi$. To prove the first rule, it therefore suffices to show that $[\![\mathsf{Disc}(A)]\!] \subseteq \mathit{Elim}^{-1}$, which by universal property of $[\![\mathsf{Disc}(A)]\!]$ means showing that $\mathit{Mod}_{\mathsf{cc}}([\![A]\!]) \subseteq \mathit{Elim}^{-1}$ and $\mathit{Fhcom}(\mathit{Elim}^{-1}) \subseteq \mathit{Elim}^{-1}$.

To show that $\mathit{Mod}_{\mathsf{cc}}([\![A]\!]) \subseteq \mathit{Elim}^{-1}$, we observe that the second rule above holds immediately by coherent head expansion. It follows that any equal values in $\mathit{Mod}_{\mathsf{cc}}([\![A]\!])$ are also equal in Elim^{-1}.

To show that $Fhcom(Elim^{-1}) \subseteq Elim^{-1}$, suppose we are given a pair of formal composites $\mathsf{fhcom}^{t\to u}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \approx \mathsf{fhcom}^{t\to u}(M'; \overrightarrow{\xi_i \hookrightarrow y.N'_i}) \in Fhcom(Elim^{-1})\langle\psi\rangle$. When we apply the eliminator to these values, the results reduce to composites of eliminations in the target family B, which are well-typed because the arguments to the formal composites belong to $\Downarrow Elim^{-1}$. It is straightforward to check that we can then apply coherent expansion to see that these reductions induce equalities, and we thereby deduce that the formal composites belong to $Elim^{-1}$ as required. □

14.4.3 Splitting

Before we finish, there is one last construct we need to make proper use of bridge endpoint assumptions, an operator split that performs endpoint case analysis. Its operational semantics are included in Figure 14.4. This operator is easily seen to satisfy the following rules; note that in a interval context Ψ, any endpoint term is either $\mathbf{0}$ or $\mathbf{1}$.

Rules 14.4.14 (Splitting).

$$\frac{\Psi \Vdash A\ \mathsf{type}\ @\ m \qquad \Psi \Vdash \boldsymbol{r} \in \mathbf{2}\ @\ m \qquad \Psi, \boldsymbol{r} \equiv \mathbf{0} \Vdash M_0 = M'_0 \in A\ @\ m \qquad \Psi, \boldsymbol{r} \equiv \mathbf{1} \Vdash M_1 = M'_1 \in A\ @\ m}{\Psi \Vdash \mathsf{split}_{\boldsymbol{r}}(M_0, M_1) = \mathsf{split}_{\boldsymbol{r}}(M'_0, M'_1) \in A}$$

$$\frac{\Psi \Vdash A\ \mathsf{type}\ @\ m \qquad \Psi \Vdash M_0 \in A\ @\ m}{\Psi \Vdash \mathsf{split}_{\boldsymbol{r}}(M_0, M_1) = M_0 \in A} \qquad \frac{\Psi \Vdash A\ \mathsf{type}\ @\ m \qquad \Psi \Vdash M_1 \in A\ @\ m}{\Psi \Vdash \mathsf{split}_{\boldsymbol{r}}(M_0, M_1) = M_1 \in A}$$

Proof. Immediate by coherent expansion. □

Chapter 15

Programming in cohesive parametric type theory

Having established a basic suite of rules governing the modal context operators and modal types, we now apply the theory. As described in Chapter 13, our overarching goal is to show that the free theorems that hold of terms defined in the parametric mode can be used to obtain results in the pointwise mode.

We begin in Section 15.1 with a few lemmas for conveniently reasoning about the discrete embedding type Disc. In Section 15.2 we return to the example of Church booleans from Section 10.1: we show that any pointwise Church boolean that arises from a parametric Church boolean is "true" or "false". Section 15.3 revisits the concept of bridge-discreteness introduced in Section 10.3; we show in particular that types of the form $\mathrm{Disc}(A)$ are bridge-discrete. Finally, Section 15.4 shows that we can apply our characterization of parametrically polymorphic functions between smash products from Section 10.5 to obtain algebraic laws and coherences for the pointwise smash product.

15.1 Properties of the discrete embedding

Before getting into concrete examples, it is useful to derive a few basic properties of the discrete type, which plays the central role in transferring parametricity results.

First, in addition to the ordinary discrete eliminator, the presence of the codiscrete type allows us to derive an eliminator for inhabiting *pointwise* families indexed by a modal hypothesis $(\mathsf{dsc} \mid d : \mathrm{Disc}(A))$ of discrete type. This is analogous to Shulman's derivation of "crisp $\flat$-induction" [Shu18, Lemma 5.1] in his own cohesive type theory; our modal hypotheses under dsc play the role of his crisp hypotheses, while Disc-types play the role of $\flat$-types.

Lemma 15.1.1 (Pointwise elimination). We have a term $\mathsf{letdisc}_{\mathsf{pt}}(d.B, P, a.N)$ validating the following for any $\Gamma \gg A$ type @ pt, family $\Gamma, (\mathsf{dsc} \mid d : \mathsf{Disc}(A)) \gg B$ type @ pt, and $\Gamma, a : A \gg N \in B[\mathsf{mod}(a)/d]$ @ pt.

$$\frac{\Gamma.\mathsf{dsc} \gg P \in \mathsf{Disc}(A) \text{ @ par}}{\Gamma \gg \mathsf{letdisc}_{\mathsf{pt}}(d.B, P, a.N) \in B[P/d] \text{ @ pt}}$$

$$\frac{\Gamma \gg M \in A \text{ @ pt}}{\Gamma \gg \mathsf{letdisc}_{\mathsf{pt}}(d.B, \mathsf{mod}(M), a.N) = N[M/a] \in B[\mathsf{mod}(M)/d] \text{ @ pt}}$$

Proof. We define pointwise elimination as ordinary elimination into the codiscrete embedding of B.

$$\mathsf{letdisc}_{\mathsf{pt}}(d.B, P, a.N) \coloneqq \mathsf{unmod}(\mathsf{letdisc}(d.\mathsf{Codisc}(B), P, a.\mathsf{mod}(N)))$$

We aim to show this term has type $B[P/d]$. By the projection rule for Codisc, it suffices to show that $\Gamma.\mathsf{dsc} \gg \mathsf{letdisc}(d.\mathsf{Codisc}(B), P, a.\mathsf{mod}(N)) \in \mathsf{Codisc}(B[P/d])$.

To show $\Gamma.\mathsf{dsc}, d : \mathsf{Disc}(A) \gg \mathsf{Codisc}(B)$ type @ par, it suffices by Codisc-formation to check that $(\Gamma.\mathsf{dsc}, d : \mathsf{Disc}(A)).\mathsf{glo} \gg B$ type @ pt. We have $(\Gamma.\mathsf{dsc}, d : \mathsf{Disc}(A)).\mathsf{glo} = \Gamma.\mathsf{dsc}.\mathsf{glo}, (\mathsf{dsc} \mid d : \mathsf{Disc}(A))$ by definition. It follows from $\Gamma, (\mathsf{dsc} \mid d : \mathsf{Disc}(A)) \gg B$ type @ pt and the counit substitution of the discrete-global adjunction that we have $\Gamma.\mathsf{dsc}.\mathsf{glo}, (\mathsf{dsc} \mid d : \mathsf{Disc}(A)) \gg B$ type @ pt.

To show $\Gamma.\mathsf{dsc}, (\mathsf{cc} \mid a : A) \gg \mathsf{mod}(N) \in \mathsf{Codisc}(B[\mathsf{mod}(a)/d])$ @ par, it suffices by Codisc-introduction to show $(\Gamma.\mathsf{dsc}, (\mathsf{cc} \mid a : A)).\mathsf{glo} \gg N \in B[\mathsf{mod}(a)/d]$ @ pt. Again we compute the action of context modality.

$$(\Gamma.\mathsf{dsc}, (\mathsf{cc} \mid a : A)).\mathsf{glo} = \Gamma.\mathsf{dsc}.\mathsf{glo}, (\mathsf{dsc}, \mathsf{cc} \mid a : A)$$

We deduce $\Gamma.\mathsf{dsc}.\mathsf{glo}, (\mathsf{dsc}, \mathsf{cc} \mid a : A) \gg N \in B[\mathsf{mod}(a)/d]$ @ pt from the assumption $\Gamma, a : A \gg N \in B[\mathsf{mod}(a)/d]$ @ pt using the counit substitution of the discrete-global adjunction and the unit of the components-discrete adjunction.

Combining these with $\Gamma.\mathsf{dsc} \gg P \in \mathsf{Disc}(A)$ @ par, we apply parametric elimination to see that $\Gamma.\mathsf{dsc} \gg \mathsf{letdisc}(d.\mathsf{Codisc}(B), P, a.\mathsf{mod}(N)) \in \mathsf{Codisc}(B[P/d])$ @ par. The projection rule for the codiscrete type now gives the conclusion of the first rule. The second rule follows analogously by the reduction rules for the discrete eliminator and codiscrete projection. □

In truth, we will use this elimination principle only to define the following construction for *projecting* the underlying element of A from a hypothesis $(\mathsf{dsc} \mid d : \mathsf{Disc}(A))$.

Lemma 15.1.2. Given A type and $(\mathsf{dsc} \mid d : \mathsf{Disc}(A))$, there is some $\mathsf{undisc}(d) \in A$ with the following properties.

- For any $a : A$, we have $\mathsf{undisc}(\mathsf{mod}(a)) = a \in A$.
- For any $(\mathsf{dsc} \mid d : \mathsf{Disc}(A))$, we have a path as follows.

$$\mathsf{undisc\text{-}uniq}(d) \in \mathsf{Glo}(\mathsf{Path}(\mathsf{Disc}(A), \mathsf{mod}(\mathsf{undisc}(d)), d))$$

Proof. Set $\mathsf{undisc}(d) \coloneqq \mathsf{letdisc}_{\mathsf{pt}}(\_.A, d, a.a)$. The first property follows from the reduction rule for the pointwise eliminator. For the second, we construct the path by pointwise elimination into $B \coloneqq \mathsf{Glo}(\mathsf{Path}(\mathsf{Disc}(A), \mathsf{mod}(\mathsf{undisc}(d)), d))$.

$$\mathsf{undisc\text{-}uniq}(d) \coloneqq \mathsf{letdisc}_{\mathsf{pt}}(d.B, d, a.\mathsf{mod}(\lambda^{\mathbb{I}}\_.\, \mathsf{mod}(a))) \qquad \square$$

From this point forward, we will use the following syntactic sugar for the discrete eliminator, mimicking our higher inductive type pseudocode.

$$\left[\begin{array}{l} \textbf{case } P \textbf{ of} \\ \mid \mathsf{mod}(a) \mapsto N \end{array}\right] \coloneqq \mathsf{letdisc}(d.B, P, a.N)$$

The type argument B is implicit here, but should be straightforward for the reader to infer in concrete cases. Again as with HITs, we will also collapse iterated case analyses into a single block branching on two or more terms, as in the following definition.

Proposition 15.1.3 (Action of the discrete embedding). For any $(\mathsf{cc} \mid A, B : \mathsf{U})$, we have a term $\mathsf{map\text{-}disc} \in \mathsf{Disc}(A \to B) \to \mathsf{Disc}(A) \to \mathsf{Disc}(B)\ @\ \mathsf{par}$ defined as follows.

$$\mathsf{map\text{-}disc} \coloneqq \lambda f.\, \lambda d. \left[\begin{array}{l} \textbf{case } f, d \textbf{ of} \\ \mid \mathsf{mod}(g), \mathsf{mod}(a) \mapsto \mathsf{mod}(f\ a) \end{array}\right]$$

De-sugared, this is $\lambda f.\, \lambda d.\, \mathsf{letdisc}(\_.\mathsf{Disc}(B), f, g.\mathsf{letdisc}(\_.\mathsf{Disc}(B), d, a.\mathsf{mod}(f\ a)))$.

15.2 Church booleans

To demonstrate how we can apply parametricity results in the pointwise fragment, let us revisit the Church boolean example presented in Section 10.1.

$$\mathbb{B} \coloneqq (A : \mathsf{U}) \to A \to A \to A$$

Built from a universe and function types, the type of Church booleans exists in both the pointwise and parametric modes: we have both $\mathbb{B}$ type @ par and $\mathbb{B}$ type @ pt. Note that while these types are syntactically identical, they are interpreted as different relations: the elements of $\mathbb{B}$ in the parametric mode must have an action on bridges, while the elements of $\mathbb{B}$ in the pointwise mode need only act on paths. (Indeed, it is merely a

coincidence that we have used the same syntax for the parametric and pointwise type formers.) As such, we cannot expect to show that any element of the pointwise $\mathbb{B}$ is path-equal to a $\mathbb{t}$ or $\mathbb{f}$. However, we *can* expect that any pointwise Church boolean that arises from a parametric Church boolean can be so characterized.

Within the pointwise mode, we have access to the type of parametric Church booleans via the global type $\mathsf{Glo}(\mathbb{B})$. Such a term is a polymorphic function defined for all elements of the parametric universe; by restricting it to discrete types, we can access its "underlying" pointwise function.

Lemma 15.2.1. We have a function $\mathsf{shadow} \in \mathsf{Glo}(\mathbb{B}) \to \mathbb{B}\ @\ \mathsf{pt}$ defined as follows.

$$\mathsf{shadow}\, c \coloneqq \lambda A.\, \lambda t.\, \lambda f.\, \mathsf{undisc}(\mathsf{unmod}(c)\ (\mathsf{Disc}(A))\ (\mathsf{mod}(t))\ (\mathsf{mod}(f)))$$

Proof. It is instructive to go through a typing derivation for the above term, working our way inward from the outside. By the introduction rule for functions, we must type the inner term in the context $\Gamma \coloneqq (c : \mathsf{Glo}(\mathbb{B}), A : \mathsf{U}, t : A, f : A)$. Next we come to undisc. To apply Lemma 15.1.2, we must show the following.

$$\Gamma.\mathsf{dsc} \gg \mathsf{unmod}(c)\ (\mathsf{Disc}(A))\ (\mathsf{mod}(t))\ (\mathsf{mod}(f)) \in \mathsf{Disc}(A)\ @\ \mathsf{par}$$

First, we have $\Gamma \gg c \in \mathsf{Glo}(\mathbb{B})\ @\ \mathsf{pt}$. As $\Gamma = \Gamma.\mathsf{dsc.cc}$, we can apply the projection for the global type to see that $\Gamma.\mathsf{dsc} \gg \mathsf{unmod}(c) \in \mathbb{B}\ @\ \mathsf{par}$. Next, we have $\Gamma \gg A\ \mathsf{type}\ @\ \mathsf{pt}$; again using $\Gamma = \Gamma.\mathsf{dsc.cc}$, we can apply the formation rule for the discrete type to learn that $\Gamma.\mathsf{dsc} \gg \mathsf{Disc}(A)\ \mathsf{type}\ @\ \mathsf{par}$. Similarly, we use $\Gamma.\mathsf{dsc.cc} \gg t \in A\ @\ \mathsf{pt}$ and $\Gamma.\mathsf{dsc.cc} \gg f \in A\ @\ \mathsf{pt}$ to derive $\Gamma.\mathsf{dsc} \gg \mathsf{mod}(t) \in \mathsf{Disc}(A)\ @\ \mathsf{par}$ and $\Gamma.\mathsf{dsc} \gg \mathsf{mod}(f) \in \mathsf{Disc}(A)\ @\ \mathsf{par}$. Applying $\mathsf{unmod}(c)$ at these arguments gives $\Gamma.\mathsf{dsc} \gg \mathsf{unmod}(c)\ (\mathsf{Disc}(A))\ (\mathsf{mod}(t))\ (\mathsf{mod}(f)) \in \mathsf{Disc}(A)\ @\ \mathsf{par}$ as required. □

In particular, if we take the "shadows" of the canonical parametric elements $\mathbb{t}, \mathbb{f} \in \mathbb{B}\ @\ \mathsf{par}$, we obtain their pointwise equivalents.

Lemma 15.2.2. We have the following equations.

$$\mathsf{shadow}\ (\mathsf{mod}(\mathbb{t})) = \mathbb{t} \in \mathbb{B}\ @\ \mathsf{pt} \qquad \mathsf{shadow}\ (\mathsf{mod}(\mathbb{f})) = \mathbb{f} \in \mathbb{B}\ @\ \mathsf{pt}$$

Proof. By the reduction equation for undisc. □

Using the action of unmod on paths, we can then say that the shadow of any parametric Church boolean is equal to one of the canonical pointwise elements.

Theorem 15.2.3. For any $c : \mathsf{Glo}(\mathbb{B})$, we have either a path $(\mathsf{shadow}\, c) \rightsquigarrow \mathbb{t}$ or a path $(\mathsf{shadow}\, c) \rightsquigarrow \mathbb{f}$.

Proof. By the projection rule, we have $(c : \mathrm{Glo}(\mathbb{B})).\mathrm{dsc} \gg \mathrm{unmod}(c) \in \mathbb{B}\ @\ \mathrm{par}$. Per the argument in Section 10.1—which we can apply in the parametric mode—we either have some path $(c : \mathrm{Glo}(\mathbb{B})).\mathrm{dsc} \gg P \in \mathrm{Path}(\mathbb{B}, \mathrm{unmod}(c), \mathbb{t})\ @\ \mathrm{par}$ or some path $(c : \mathrm{Glo}(\mathbb{B})).\mathrm{dsc} \gg P \in \mathrm{Path}(\mathbb{B}, \mathrm{unmod}(c), \mathbb{f})\ @\ \mathrm{par}$. Without loss of generality, let us suppose the former is the case. Applying P pointwise, we have $(c : \mathrm{Glo}(\mathbb{B}), x : \mathbb{I}).\mathrm{dsc} \gg P\,x \in \mathbb{B}\ @\ \mathrm{par}$, here using that dsc—like all of our modalities—commutes with path interval hypotheses. We can then apply the global introduction rule and shadow to derive $c : \mathrm{Glo}(\mathbb{B}), x : \mathbb{I} \gg \mathrm{shadow}\,(\mathrm{mod}(P\,x)) \in \mathrm{Glo}(\mathbb{B})\ @\ \mathrm{pt}$, followed by path abstraction for $c : \mathrm{Glo}(\mathbb{B}) \gg \lambda^{\mathbb{I}} x.\, \mathrm{shadow}\,(\mathrm{mod}(P\,x)) \in \mathrm{Path}(\mathrm{Glo}(\mathbb{B}), \mathrm{shadow}\,c, \mathbb{t})\ @\ \mathrm{pt}$. □

Put another way, any pointwise $K \in \mathrm{Bool}$ in the image of shadow is either $\mathbb{t}$ or $\mathbb{f}$ up to a path. This is perhaps not so useful in the case of Church booleans. However, the same technique applies more generally: if we can show that a pointwise term is the "shadow" of some parametric term, then we can deduce that it satisfies parametricity theorems. We apply this technique to the case of the smash product in Section 15.4.

Codiscrete shadow In the particular example of Church booleans, it is also possible to define the shadow of a Church boolean by instantiation at *codiscrete* types.

$$\mathrm{shadow}'\,c \coloneqq \lambda A.\, \lambda t.\, \lambda f.\, \mathrm{unmod}(\mathrm{unmod}(c)\,(\mathrm{Codisc}(A))\,(\mathrm{mod}(t))\,(\mathrm{mod}(f)))$$

One obtains the same results: the shadow′ of any Church boolean is $\mathbb{t}$ or $\mathbb{f}$ up to a path. However, this route fails to generalize to type expressions containing inductive types. Unlike the discrete embedding, the codiscrete embedding does not commute with such type formers. For example, we have $\mathrm{Codisc}(A + B) \not\simeq \mathrm{Codisc}(A) + \mathrm{Codisc}(B)$ in general, as only the former contains bridges between inl and inr elements. In categorical terms, this reflects that the codiscrete type is only a right adjoint, not a left adjoint, and so need not preserve colimits.

The limits of shadowing We saw in Section 10.4 that internal parametricity suffices to refute the (weak) law of the excluded middle; that is, we have some term of the following type.

$$((A : \mathsf{U}) \to (b : \mathrm{Bool}) \times \mathrm{elim}_{\mathrm{Bool}}(\_.\mathsf{U}; b; \neg A, \neg\neg A)) \to \mathrm{Void}$$

Given the methodology just outlined, one may wonder if this result can be transferred to the pointwise setting as well. In this case, however, the polarity is wrong. In the Church boolean type $(A : \mathsf{U}) \to A \to A \to A$, the universe U occurs in a *negative* position (to the left of an odd number of function arrows). We can therefore exploit the fact that the pointwise universe is embedded in the parametric universe via Disc, restricting a function defined on parametric types to one defined on pointwise types. In the type of the

refutation of $\mathsf{LEM}_{\neg}$, on the other hand, U occurs in a *positive* position. In sum, although we cannot decide the inhabitation of parametric types, this does not imply we cannot decide the inhabitation of the "smaller" class of pointwise types.[1]

15.3 Bridge-discreteness

Before introducing cohesive parametric type theory, we already had a notion of discreteness: the concept of *bridge-discrete type* introduced in Section 10.3. These play an important role in parametricity theorems that involve external type parameters, such as the characterization of $(B : \mathsf{U}) \to (A \to B) \to B$ for bridge-discrete A given in that section. We would therefore hope that pointwise types brought to the parametric fragment by Disc are bridge-discrete. This is indeed the case. (We do not, on the other hand, expect to be able to show that every bridge-discrete type is isomorphic to one of the form $\mathsf{Disc}(A)$.)

Theorem 15.3.1. For any $(\mathsf{cc} \mid A : \mathsf{U})$, $\mathsf{Disc}(A)$ is bridge-discrete.

Proof. Per Lemma 10.3.3, it suffices to show that $\mathsf{Bridge}(\mathsf{Disc}(A), d_0, d_1)$ is a retract of $\mathsf{Path}(\mathsf{Disc}(A), d_0, d_1)$ for every $d_0, d_1 : \mathsf{Disc}(A)$. We take an approach similar to our proof of boolean bridge-discreteness (Theorem 10.3.7), first defining a function into the Gel type $G_{\boldsymbol{x}} \coloneqq \mathsf{Gel}_{\boldsymbol{x}}(\mathsf{Disc}(A), \mathsf{Disc}(A), \mathsf{Path}(\mathsf{Disc}(A), -, -))$ as follows.

$$F_{\boldsymbol{x}} \coloneqq \lambda d. \left[\begin{array}{l} \textbf{case } d \textbf{ of} \\ \mid \mathsf{mod}(a) \mapsto \mathsf{gel}_{\boldsymbol{x}}(\mathsf{mod}(a), \mathsf{mod}(a), \lambda^{\mathbb{I}}\_. \mathsf{mod}(a)) \end{array} \right] \in \mathsf{Disc}(A) \to G_{\boldsymbol{x}}$$

For this term to be well-typed according to Rules 14.4.13, we must show that we have $(\mathsf{cc} \mid A : \mathsf{U}), \boldsymbol{x} : \mathbf{I}, (\mathsf{cc} \mid a : A) \gg \mathsf{gel}_{\boldsymbol{x}}(\mathsf{mod}(a), \mathsf{mod}(a), \lambda^{\mathbb{I}}\_. \mathsf{mod}(a)) \in G_{\boldsymbol{x}}$ @ par. We want to apply Gel introduction and Disc introduction in each argument. Looking at the first, we then need $((\mathsf{cc} \mid A : \mathsf{U}), \boldsymbol{x} : \mathbf{I}, (\mathsf{cc} \mid a : A) \setminus \boldsymbol{x}).\mathsf{cc} \gg a \in A$ @ pt. This follows by computing the effect of restriction and connected components on the context: we have $((\mathsf{cc} \mid A : \mathsf{U}), \boldsymbol{x} : \mathbf{I}, (\mathsf{cc} \mid a : A) \setminus \boldsymbol{x}).\mathsf{cc} = A : \mathsf{U}, a : A$. The same argument allows us to type the other two arguments. The key here is that we can guarantee a is apart from the interval variable $\boldsymbol{x}$, because it is hypothesized under cc: it only depends on the connected components of its predecessors in the context. Using $F_{\boldsymbol{x}}$, we obtain a function from bridges in $\mathsf{Disc}(A)$ to paths in $\mathsf{Disc}(A)$.

$$F \coloneqq \lambda p. \mathsf{ungel}(\boldsymbol{x}.F_{\boldsymbol{x}}\ (p\ \boldsymbol{x})) \in \mathsf{Bridge}(\mathsf{Disc}(A), d_0, d_1) \to \mathsf{Path}(\mathsf{Disc}(A), F_{\mathbf{0}}\ d_0, F_{\mathbf{1}}\ d_1)$$

[1] The pointwise excluded middle may of course fail for other reasons. Sattler has claimed that LEM_{-1} is in fact falsified in the Kan cartesian cubical set model of type theory [Sat18], a fact that would presumably carry over to our computational interpretation.

Conversely, we have a function $G_{\boldsymbol{x}} \to \mathsf{Disc}(A)$ given by loosen and extent, as in the case of Bool.

$$L_{\boldsymbol{x}} \coloneqq \lambda g.\, \mathsf{extent}_{\boldsymbol{x}}(g; d_0.d_0, d_1.d_1, \_.\_.q.\mathsf{loosen}_{\mathsf{Disc}(A)}\,(\mathsf{ungel}(\boldsymbol{x}.q\,\boldsymbol{x}))) \in G_{\boldsymbol{x}} \to \mathsf{Disc}(A)$$

We next construct a term $P_{\boldsymbol{x}} \in (d : \mathsf{Disc}(A)) \to \mathsf{Path}(\mathsf{Disc}(A), L_{\boldsymbol{x}}\,(F_{\boldsymbol{x}}\,d), d)$ showing that $F_{\boldsymbol{x}}$ is a right inverse to $L_{\boldsymbol{x}}$. By parametric elimination for the discrete type, it suffices to show $\mathsf{Path}(\mathsf{Disc}(A), L_{\boldsymbol{x}}\,(F_{\boldsymbol{x}}\,\mathsf{mod}(a)), \mathsf{mod}(a))$ for all $(\mathrm{cc} \mid a : A)$. This follows from the following sequence of paths and equations in $\mathsf{Disc}(A)$.

$$\begin{aligned}
L_{\boldsymbol{x}}\,(F_{\boldsymbol{x}}\,(\mathsf{mod}(a))) &= \mathsf{extent}_{\boldsymbol{x}}(F_{\boldsymbol{x}}\,(\mathsf{mod}(a)); d_0.d_0, d_1.d_1, _._.q.\mathsf{loosen}_{\mathsf{Disc}(A)}\,(\mathsf{ungel}(\boldsymbol{x}.q\,\boldsymbol{x})))\\
&= \mathsf{loosen}_{\mathsf{Disc}(A)}(\mathsf{ungel}(\boldsymbol{x}.\mathsf{gel}_{\boldsymbol{x}}(\mathsf{mod}(a), \mathsf{mod}(a), \lambda^{\mathbb{I}}_.\,\mathsf{mod}(a))))\,\boldsymbol{x}\\
&= \mathsf{loosen}_{\mathsf{Disc}(A)}(\lambda^{\mathbb{I}}_.\,\mathsf{mod}(a))\,\boldsymbol{x}\\
&\rightsquigarrow (\lambda^{\mathbf{I}}_.\,\mathsf{mod}(a))\,\boldsymbol{x}\\
&= \mathsf{mod}(a)
\end{aligned}$$

For any $q{:}\mathsf{Bridge}(\mathsf{Bool}, d_0, d_1)$, the term $\lambda q.\, \lambda^{\mathbb{I}} y.\, \lambda^{\mathbf{I}}\boldsymbol{x}.\, P_{\boldsymbol{x}}\,(q\,\boldsymbol{x})\,y$ then has the following type.

$$\mathsf{Path}(y.\mathsf{Bridge}(\mathsf{Bool}, P_{\mathbf{0}}\,d_0\,y, P_{\mathbf{1}}\,d_1\,y), \mathsf{loosen}_{\mathsf{Disc}(A)}\,(F\,q), q)$$

By the same argument used to prove Theorem 10.3.7, we can use singleton contractibility to replace F by some $F' \in \mathsf{Bridge}(\mathsf{Disc}(A), d_0, d_1) \to \mathsf{Path}(\mathsf{Disc}(A), d_0, d_1)$ that satisfies the above with $P_{\mathbf{0}}\,d_0$ and $P_{\mathbf{1}}\,d_1$ replaced by reflexive paths, showing that the bridge type is a retract of the path type. □

We therefore have, for example, that $((B : \mathsf{U}) \to (\mathsf{Disc}(A) \to B) \to B) \simeq \mathsf{Disc}(A)$ as a consequence of Theorem 10.3.4. Polymorphic types like this one—where external pointwise types appear wrapped in Disc—are also amenable to the construction of "shadows", as in the example below. (We leave the type-checking as an exercise to the reader.)

Proposition 15.3.2. For any $A : \mathsf{U}$ and $c : \mathsf{Glo}((B : \mathsf{U}) \to (\mathsf{Disc}(A) \to B) \to B)$, we have an induced function $\mathsf{shadow}_A\,c \in (B : \mathsf{U}) \to (A \to B) \to B\ @\ \mathsf{pt}$ defined as follows.

$$\mathsf{shadow}_A\,c \coloneqq \mathsf{undisc}(\mathsf{unmod}(c)\,(\mathsf{Disc}(B))\,(\mathsf{map\text{-}disc}\,(\mathsf{mod}(f))))$$

The codiscrete type satisfies a complementary property: it is *bridge-codiscrete*, in the sense that its bridge types are contractible. Here we see how the split operator allows us to construct bridges in the codiscrete type.

Theorem 15.3.3. For any type $(\mathrm{glo} \mid A : \mathsf{U})$ and $c_0, c_1 : \mathsf{Codisc}(A)$, the type of bridges $\mathsf{Bridge}(\mathsf{Codisc}(A), c_0, c_1)$ is contractible.

Proof. We have an element P of the bridge type defined as follows.

$$P :\equiv \lambda^{\mathbf{I}} \boldsymbol{x}.\, \mathsf{mod}(\mathsf{split}_{\boldsymbol{x}}(\mathsf{unmod}(c_0), \mathsf{unmod}(c_1))) \in \mathsf{Bridge}(\mathsf{Codisc}(A), c_0, c_1) \;@\; \mathsf{par}$$

Note that we type split in the context $((\mathsf{glo} \mid A : \mathsf{U}), c_0, c_1 : \mathsf{Codisc}(A), \boldsymbol{x} : \mathbf{I}).\mathsf{glo}$, with the application of glo brought about by the introduction rule for the codiscrete type. The global modality transforms the hypothesis $\boldsymbol{x} : \mathbf{I}$ into $\boldsymbol{x} : \mathbf{2}$, enabling us to analyze it with split.

We prove uniqueness of P in a similar fashion. For any $q : \mathsf{Bridge}(\mathsf{Codisc}(A), c_0, c_1)$, we define a path $S \in \mathsf{Path}(\mathsf{Bridge}(\mathsf{Codisc}(A), c_0, c_1), q, P) \;@\; \mathsf{par}$ from q to P as follows.

$$\lambda^{\mathbb{I}} y.\, \lambda^{\mathbf{I}} \boldsymbol{x}.\, \mathsf{mod}(\mathsf{split}_{\boldsymbol{x}}(\lambda^{\mathbb{I}} \_.\, \mathsf{unmod}(c_0), \lambda^{\mathbb{I}} \_.\, \mathsf{unmod}(c_1))\; y)$$

Here we are applying split at a path type like so.

$$\mathsf{split}_{\boldsymbol{x}}(\lambda^{\mathbb{I}} \_.\, \mathsf{unmod}(c_0), \lambda^{\mathbb{I}} \_.\, \mathsf{unmod}(c_1)) \in \mathsf{Path}(A, \mathsf{unmod}(q\,\boldsymbol{x}), \mathsf{unmod}(P\,\boldsymbol{x})) \;@\; \mathsf{par}$$

That is, we know that $\mathsf{unmod}(q\,\boldsymbol{x})$ and $\mathsf{unmod}(P\,\boldsymbol{x})$ agree for any $\boldsymbol{x} : \mathbf{2}$ by virtue of their endpoint equations, and it follows by nature of the codiscrete type that $q\,\boldsymbol{x}$ and $P\,\boldsymbol{x}$ agree for any $\boldsymbol{x} : \mathbf{I}$. □

15.4 Iterated smash products

We now return to the showcase result of Part III, the characterization of polymorphic pointed functions between smash products (Section 10.5). We see that given commutator and associator functions defined in the parametric mode, we can derive pointwise functions that are guaranteed by parametricity to satisfy various coherence properties.

Preliminaries It will pay at this point to develop a more structured toolkit for deriving shadows of parametric functions. First, we introduce shorthand notation for instantiating a parametrically polymorphic function at a discrete type; we also here shift attention from types to pointed types.

Definition 15.4.1. Given a pointwise pointed type $(\mathsf{cc} \mid A_* : \mathsf{U}_*)$, we define its discrete embedding as $\mathsf{Disc}_*(A_*) :\equiv \langle \mathsf{Disc}(A), \mathsf{mod}(a_0) \rangle \in \mathsf{U}_* \;@\; \mathsf{par}$.

Notation 15.4.2. Given $X : \mathsf{U}_* \gg B$ type $@$ par, a function $f : (X : \mathsf{U}_*) \to B$, and a type $(\mathsf{cc} \mid A : \mathsf{U}_*)$, we define $f \triangleleft A_* :\equiv f\,(\mathsf{Disc}_*(A_*)) \in B[\mathsf{Disc}_*(A)/X] \;@\; \mathsf{pt}$.

Let us also explicitly define identity and composition of pointed functions.

Definition 15.4.3 (Identity and composites of pointed functions). Given $A_* : \mathsf{U}_*$ in any mode m, we define the pointed identity function $\mathrm{id}_*(A_*) \in A_* \to A_* \mathbin{@} m$ as follows.

$$\mathrm{id}_*(A_*) \coloneqq \langle \lambda a.\, a, \lambda^{\mathbb{I}} \_.\, a_0 \rangle$$

Given two pointed functions $f : A_* \to B_*$ and $g : B_* \to C_*$, we define their pointed composite $g_* \circ_* f_* \in A_* \to C_* \mathbin{@} m$ as follows

$$g_* \circ_* f_* \coloneqq \langle \lambda a.\, g\,(f\,a), \lambda^{\mathbb{I}} x.\, \mathsf{hcom}_C^{0\to 1}(g\,(f_0\,x); x \equiv 0 \hookrightarrow \_.g\,(f\,a_0), x \equiv 1 \hookrightarrow y.g_0\,y) \rangle$$

The second component here is the composite of the path $\lambda^{\mathbb{I}} x.\, g\,(f_0\,x)$ from $g\,(f\,a_0)$ to $g\,b_0$ with the path g_0 from $g\,b_0$ to c_0.

We take a few basic algebraic properties of pointed functions for granted, namely the unit laws and associativity of pointed composition.

The first essential result for constructing shadows is the following isomorphism, which equates pointwise functions with globally-defined parametric functions between discrete types.

Lemma 15.4.4. Given a pair of pointed types $A_*, B_* : \mathsf{U}_*$ and $f_* : A_* \to B_*$, we have a term $\Diamond_* f_* \in \mathsf{Glo}(\mathsf{Disc}_*(A_*) \to \mathsf{Disc}_*(B_*)) \mathbin{@} \mathsf{pt}$. Conversely, given a global pointed function $u : \mathsf{Glo}(\mathsf{Disc}_*(A_*) \to \mathsf{Disc}_*(B_*))$, we have a term $\blacklozenge_* u \in A_* \to B_* \mathbin{@} \mathsf{pt}$. The two functions $\Diamond_*$ and $\blacklozenge_*$ constitute an isomorphism.

Proof. We define $\Diamond_* f_*$ and $\blacklozenge_* u$ as follows.

$$\Diamond_* f_* \coloneqq \mathsf{mod}(\langle \mathsf{map\text{-}disc}\, f, \lambda^{\mathbb{I}} x.\, \mathsf{mod}(f_0\,x) \rangle)$$
$$\blacklozenge_* u \coloneqq \langle \lambda a.\, \mathsf{undisc}(\mathsf{fst}(\mathsf{unmod}(u))\,(\mathsf{mod}(a))), \lambda^{\mathbb{I}} x.\, \mathsf{undisc}(\mathsf{snd}(\mathsf{unmod}(u))\,x) \rangle$$

One inverse condition holds up to exact equality: we have $\blacklozenge_* \Diamond_* f_* = f_* \in A_* \to B_*$ for any $f_* : A_* \to B_*$. For the other, given any $u : \mathsf{Glo}(\mathsf{Disc}_*(A_*) \to \mathsf{Disc}_*(B_*))$, we have a path $\Diamond_* \blacklozenge_* u \rightsquigarrow u$ defined as follows. First we construct an auxiliary family of paths as follows.

$$H \in \mathsf{Glo}((d : \mathsf{Disc}(A)) \to \mathsf{Path}(\mathsf{Disc}(B), \mathsf{fst}(\mathsf{unmod}(\Diamond_* \blacklozenge_* u))\,d, \mathsf{fst}(\mathsf{unmod}(u))\,d))$$
$$H \coloneqq \mathsf{mod}\left(\lambda d.\left[\begin{array}{l}\textbf{case}\ d\ \textbf{of}\\ \mid \mathsf{mod}(a) \mapsto \mathsf{unmod}(\mathsf{undisc\text{-}uniq}(\mathsf{unmod}(u)\,(\mathsf{mod}(a))))\end{array}\right]\right)$$

Then we use this to define a path P from $\Diamond_* \blacklozenge_* u$ to u.

$$P \coloneqq \lambda^{\mathbb{I}} y.\, \mathsf{mod}(\langle \lambda d.\, \mathsf{unmod}(H)\,d\,y, \lambda^{\mathbb{I}} x.\, \mathsf{undisc\text{-}uniq}(f_0\,x)\,y \rangle) \qquad \square$$

The second key lemma is more specific to the smash product: we must know that it commutes with the discrete embedding. From a categorical perspective, this follows from the fact that Disc is both a left and right adjoint. Because it is a left adjoint (to Glo), it commutes with colimits, here in the guise of a higher inductive type; because it is a right adjoint (to cc), it commutes with products.

Lemma 15.4.5. For any pointwise pointed types $(\mathsf{cc} \mid A_*, B_* : \mathsf{U}_*)$, we have a pointed isomorphism $\wedge\text{-disc} \in \mathsf{Disc}_*(A_*) \wedge_* \mathsf{Disc}_*(B_*) \simeq \mathsf{Disc}_*(A_* \wedge_* B_*)$ @ par.

Proof. For the forward function $F \in \mathsf{Disc}_*(A_*) \wedge \mathsf{Disc}_*(B_*) \to \mathsf{Disc}(A_* \wedge B_*)$ @ par, we go by induction on the smash product input. To cover the pair case, we define a map $F_{\mathsf{pair}} \in \mathsf{Disc}(A) \to \mathsf{Disc}(B) \to \mathsf{Disc}(A_* \wedge B_*)$ @ par.

$$F_{\mathsf{pair}} \coloneqq \lambda u.\, \lambda v. \left[\begin{array}{l} \textbf{case } u, v \textbf{ of} \\ \mid \mathsf{mod}(a), \mathsf{mod}(b) \mapsto \mathsf{mod}(\langle\!\langle a, b \rangle\!\rangle) \end{array}\right]$$

Next we have $F_{\mathsf{L}} \in (v : \mathsf{Disc}(B)) \to \mathsf{Path}(\mathsf{Disc}(A_* \wedge B_*), \mathsf{mod}(\circledast^{\mathsf{L}}), F_{\mathsf{pair}}\,(\mathsf{mod}(a_0))\,v)$ @ par.

$$F_{\mathsf{L}} \coloneqq \lambda v. \left[\begin{array}{l} \textbf{case } u, v \textbf{ of} \\ \mid \mathsf{mod}(b) \mapsto \lambda^{\mathbb{I}} y.\, \mathsf{mod}(\mathsf{spoke}^{\mathsf{L}}(b, y)) \end{array}\right]$$

The symmetric $F_{\mathsf{R}} \in (u : \mathsf{Disc}(A)) \to \mathsf{Path}(\mathsf{Disc}(A_* \wedge B_*), \mathsf{mod}(\circledast^{\mathsf{R}}), F_{\mathsf{pair}}\,u\,(\mathsf{mod}(b_0)))$ @ par is likewise definable. We then assemble these to construct the inverse map F.

$$F \coloneqq \lambda s. \left[\begin{array}{l} \textbf{case } s \textbf{ of} \\ \mid \langle\!\langle u, v \rangle\!\rangle \mapsto F_{\mathsf{pair}}\,u\,v \\ \mid \circledast^{\mathsf{L}} \mapsto \mathsf{mod}(\circledast^{\mathsf{L}}) \\ \mid \mathsf{spoke}^{\mathsf{L}}(b, y) \mapsto F_{\mathsf{L}}\,b\,y \\ \mid \circledast^{\mathsf{R}} \mapsto \mathsf{mod}(\circledast^{\mathsf{R}}) \\ \mid \mathsf{spoke}^{\mathsf{R}}(a, x) \mapsto F_{\mathsf{R}}\,a\,x \end{array}\right]$$

Note that $F\,\langle\!\langle \mathsf{mod}(a_0), \mathsf{mod}(b_0) \rangle\!\rangle = \mathsf{mod}(\langle\!\langle a_0, b_0 \rangle\!\rangle) \in \mathsf{Disc}(A_* \wedge B_*)$, so F is a pointed function.

In the reverse direction, we make use of the adjunction between Disc and Glo, defining first an auxiliary $A_*, B_* : \mathsf{U}_* \gg G' \in A_* \wedge B_* \to \mathsf{Glo}(\mathsf{Disc}_*(A_*) \wedge \mathsf{Disc}_*(B_*))$ @ pt.

$$G' \coloneqq \lambda s. \left[\begin{array}{l} \textbf{case } s \textbf{ of} \\ \mid \langle\!\langle a, b \rangle\!\rangle \mapsto \mathsf{mod}(\langle\!\langle \mathsf{mod}(a), \mathsf{mod}(b) \rangle\!\rangle) \\ \mid \circledast^{\mathsf{L}} \mapsto \mathsf{mod}(\circledast^{\mathsf{L}}) \\ \mid \mathsf{spoke}^{\mathsf{L}}(b, y) \mapsto \mathsf{mod}(\mathsf{spoke}^{\mathsf{L}}(\mathsf{mod}(b), y)) \\ \mid \circledast^{\mathsf{R}} \mapsto \mathsf{mod}(\circledast^{\mathsf{R}}) \\ \mid \mathsf{spoke}^{\mathsf{R}}(a, x) \mapsto \mathsf{mod}(\mathsf{spoke}^{\mathsf{R}}(\mathsf{mod}(a), x)) \end{array}\right]$$

We then transpose to define $G \in \mathsf{Disc}(A_* \wedge B_*) \to \mathsf{Disc}_*(A_*) \wedge \mathsf{Disc}_*(B_*)$ @ par as follows.

$$G \coloneqq \lambda w. \left[\begin{array}{l} \textbf{case } w \textbf{ of} \\ \mid \mathsf{mod}(s) \mapsto \mathsf{unmod}(G'\, s) \end{array} \right]$$

We leave detailed proofs of the inverse conditions as an exercise to the reader. The proofs predictably follow the structure of the functions themselves. Briefly, to define a map

$$(s : \mathsf{Disc}_*(A_*) \wedge \mathsf{Disc}_*(B_*)) \to \mathsf{Path}(\mathsf{Disc}_*(A_*) \wedge \mathsf{Disc}_*(B_*), G\,(F\,s), s)$$

we use smash product induction followed by discrete induction in each case; to show

$$(w : \mathsf{Disc}(A_* \wedge B_*)) \to \mathsf{Path}(\mathsf{Disc}(A_* \wedge B_*), F\,(G\,w), w)$$

we first prove

$$(s : A_* \wedge B_*) \to \mathsf{Glo}(\mathsf{Path}(\mathsf{Disc}(A_* \wedge B_*), F\,(\mathsf{unmod}(G'\,s)), \mathsf{mod}(s)))$$

by smash product induction and then transpose. □

Commutativity For our first concrete application, we show that any parametric commutator that behaves correctly on Bool induces a pointwise commutator that is an isomorphism.

Assumption 15.4.6. We assume given a global commutator as follows.

$$\mathsf{comm} \in \mathsf{Glo}((A_*, B_* : \mathsf{U}_*) \to A_* \wedge_* B_* \to B_* \wedge_* A_*) \text{ @ } \mathsf{pt}$$

We assume moreover that this term satisfies the following path equality.

$$\mathsf{comm}\ \mathsf{Bool}_*\ \mathsf{Bool}_*\ \langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle \rightsquigarrow \langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle$$

We derive a pointwise commutator by instantiating comm at discrete types and then applying $\blacklozenge_*$, using also that $\wedge$ commutes with Disc.

Definition 15.4.7 (Commutator shadow). Given pointwise types $A_*, B_* : \mathsf{U}_*$, we define $\mathsf{comm}_{\mathsf{pt}}\, A_*\, B_* \in A_* \wedge_* B_* \to B_* \wedge_* A_*$ @ pt as follows.

$$\mathsf{comm}_{\mathsf{pt}}\, A_*\, B_* \coloneqq \blacklozenge_*(\mathsf{mod}(\wedge\text{-}\mathsf{disc} \circ_* (\mathsf{unmod}(\mathsf{comm}) \triangleleft A_* \triangleleft B_*) \circ_* \wedge\text{-}\mathsf{disc}^{-1}))$$

That is, we first apply the parametric comm at the discrete embeddings of A_* and B_*, then adjust by $\wedge$-disc to obtain the following dashed composite function.

$$\begin{array}{ccc} \mathsf{Disc}_*(A_*) \wedge_* \mathsf{Disc}_*(B_*) & \xrightarrow[\wedge\text{-disc}]{\simeq} & \mathsf{Disc}_*(A_* \wedge_* B_*) \\ \Big\downarrow{\scriptstyle \mathsf{unmod}(\mathsf{comm}) \triangleleft A_* \triangleleft B_*} & & \vdots \\ \mathsf{Disc}_*(B_*) \wedge_* \mathsf{Disc}_*(A_*) & \xrightarrow[\wedge\text{-disc}]{\simeq} & \mathsf{Disc}_*(B_* \wedge_* A_*) \end{array}$$

Applying $\blacklozenge_*(\mathsf{mod}(-))$ then yields a map $A_* \wedge_* B_* \to B_* \wedge_* A_*$.

We show that $\mathsf{comm}_{\mathsf{pt}}\, A_*\, B_*$ is an isomorphism by checking that $\mathsf{comm}_{\mathsf{pt}}\, B_*\, A_*$ is its inverse, *i.e.*, that $\mathsf{comm}_{\mathsf{pt}}\, B_*\, A_* \circ_* \mathsf{comm}_{\mathsf{pt}}\, A_*\, B_*$ for any A_* and B_*. The aim is to reduce this condition to the corresponding condition on the parametric commutator. To do so, we need to know how $\blacklozenge_*$ interacts with function identity and composition.

Proposition 15.4.8 (Functoriality of $\blacklozenge_*$). Given $A_*, B_*, C_* : \mathsf{U}_*$ and a pair of functions $u : \mathsf{Glo}(\mathsf{Disc}_*(A_*) \to \mathsf{Disc}_*(B_*))$ and $v : \mathsf{Glo}(\mathsf{Disc}_*(B_*) \to \mathsf{Disc}_*(A_*))$, we have paths of the following types.

$$\mathsf{id}_*(A_*) \rightsquigarrow \blacklozenge_*(\mathsf{mod}(\mathsf{id}_*(\mathsf{Disc}_*(A_*)))) \in A_* \to A_*$$
$$\blacklozenge_* v \circ_* \blacklozenge_* u \rightsquigarrow \blacklozenge_*(\mathsf{mod}(\mathsf{unmod}(v) \circ_* \mathsf{unmod}(u))) \in A_* \to C_*$$

Proof (sketch). The first of these two paths holds up to exact equality. The second takes more work to establish; its proof involves undisc-uniq and, to relate the basepoint preservation paths, the fact that mod (as a constructor for Disc) commutes with hcom. □

This work can be avoided by instead *defining* composition in the pointwise mode as the shadow of parametric composition.

$$g_* \circ_*^{\mathsf{pt}} f_* \coloneqq \blacklozenge_*(\mathsf{mod}(\mathsf{unmod}(\lozenge_* g_*) \circ_* \mathsf{unmod}(\lozenge_* f_*)))$$

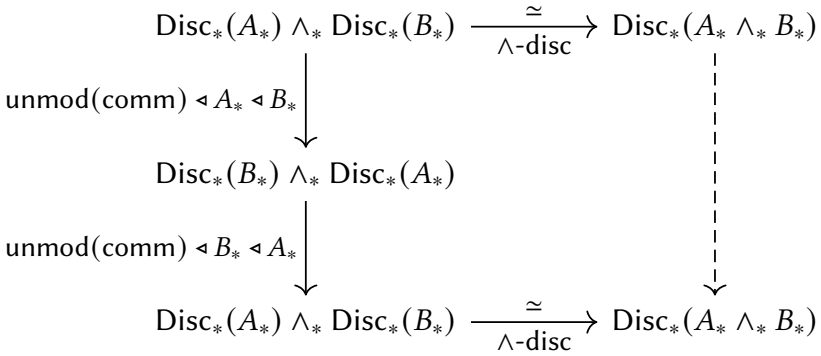
The path $\blacklozenge_* v \circ_*^{\mathsf{pt}} \blacklozenge_* u \rightsquigarrow \blacklozenge_*(\mathsf{mod}(\mathsf{unmod}(v) \circ_* \mathsf{unmod}(u)))$ then follows as a corollary of the inverse conditions of the $\lozenge_*$-$\blacklozenge_*$ isomorphism.

Theorem 15.4.9. $\mathsf{comm}_{\mathsf{pt}}$ is a family of isomorphisms.

Proof. It suffices to show that for any $A_*, B_* : \mathsf{U}_*$, we have a path in $A_* \wedge_* B_* \to B_* \wedge_* A_*$ as follows.

$$\mathsf{comm}_{\mathsf{pt}}\, B_*\, A_* \circ_* \mathsf{comm}_{\mathsf{pt}}\, A_*\, B_* \rightsquigarrow \mathsf{id}_*(A_* \wedge_* B_*)$$

By functoriality of $\blacklozenge_*$ and the path $\wedge\text{-disc}^{-1} \circ_* \wedge\text{-disc} \rightsquigarrow \mathsf{id}_*(B_* \wedge_* A_*)$, the left-hand side is path-equal to the image by $\blacklozenge_*(\mathsf{mod}(-))$ of the following dashed composite.



The composite map on the left is an instance of a parametrically polymorphic function:

$$\lambda X_*.\, \lambda Y_*.\, \mathsf{unmod}(\mathsf{comm})\, Y_*\, X_* \circ_* \mathsf{unmod}(\mathsf{comm})\, X_*\, Y_*$$
$$\pitchfork$$
$$(X_*, Y_* : \mathsf{U}_*) \to X_* \wedge_* Y_* \to_* X_* \wedge_* Y_*$$

By assumption, this function sends $\langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle$ to $\langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle$ when instantiated at Bool_* and Bool_*. By our characterization of such polymorphic functions in Part III, namely Theorem 10.5.11, we can conclude it is path-equal to the identity function. Thus we have

$$\begin{aligned} \mathsf{comm}_{\mathsf{pt}}\, B_*\, A_* \circ_* \mathsf{comm}_{\mathsf{pt}}\, A_*\, B_* &\rightsquigarrow \blacklozenge_*(\wedge\text{-disc} \circ_* \wedge\text{-disc}^{-1}) \\ &\rightsquigarrow \blacklozenge_*(\mathsf{id}_*(A_* \wedge_* B_*)) \\ &\rightsquigarrow \mathsf{id}_*(A_* \wedge_* B_*) \end{aligned}$$

as required. □

Associativity and the pentagon We can apply the same chain of reasoning to the associator, obtaining not only the isomorphism inverse conditions but also the pentagon by parametricity.

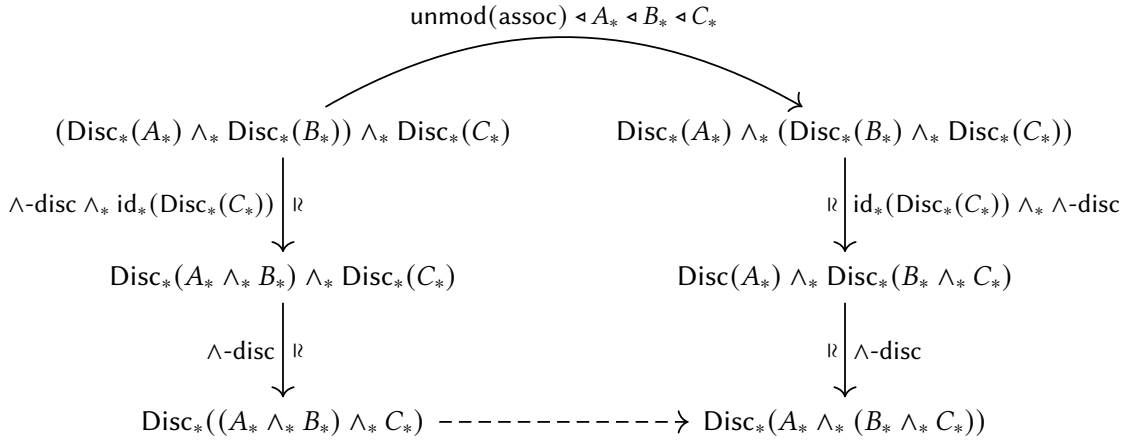
Assumption 15.4.10. We assume given a global associator and candidate inverse as follows.

$$\mathsf{assoc} \in \mathsf{Glo}((A_*, B_*, C_* : \mathsf{U}_*) \to (A_* \wedge_* B_*) \wedge_* C_* \to A_* \wedge_* (B_* \wedge_* C_*)) \ @\ \mathsf{pt}$$
$$\mathsf{assoc}^{-1} \in \mathsf{Glo}((A_*, B_*, C_* : \mathsf{U}_*) \to A_* \wedge_* (B_* \wedge_* C_*) \to (A_* \wedge_* B_*) \wedge_* C_*) \ @\ \mathsf{pt}$$

We assume moreover that these terms satisfy the following path equalities.

$$\mathsf{assoc}\ \mathsf{Bool}_*\ \mathsf{Bool}_*\ \mathsf{Bool}_*\ \langle\!\langle \langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle, \mathsf{ff} \rangle\!\rangle \rightsquigarrow \langle\!\langle \mathsf{ff}, \langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle \rangle\!\rangle$$
$$\mathsf{assoc}^{-1}\ \mathsf{Bool}_*\ \mathsf{Bool}_*\ \mathsf{Bool}_*\ \langle\!\langle \mathsf{ff}, \langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle \rangle\!\rangle \rightsquigarrow \langle\!\langle \langle\!\langle \mathsf{ff}, \mathsf{ff} \rangle\!\rangle, \mathsf{ff} \rangle\!\rangle$$

Definition 15.4.11 (Associator shadow). Given pointwise types $A_*, B_*, C_* : \mathsf{U}_*$, we define $\mathsf{assoc}_{\mathsf{pt}}\, A_*\, B_*\, C_* \in (A_* \wedge_* B_*) \wedge_* C_* \to A_* \wedge_* (B_* \wedge_* C_*)\ @\ \mathsf{pt}$ as in the case of the commutator, as the image under $\blacklozenge_*(\mathsf{mod}(-))$ of the following composite.



We likewise define $\mathsf{assoc}^{-1}_{\mathsf{pt}}\, A_*\, B_*\, C_* \in A_* \wedge_* (B_* \wedge_* C_*) \to (A_* \wedge_* B_*) \wedge_* C_*\ @\ \mathsf{pt}$.

The statement of the pentagon identity involves the action of the smash product on pointed functions. As with function identity and composition, we therefore need that the parametric and pointwise action correspond across the $\blacklozenge_*$ isomorphism. As our objective is to *avoid* reasoning about the smash product, we take the way out suggested in our discussion of Proposition 15.4.8, defining the pointwise action so that the correspondence is immediate.

Definition 15.4.12. Given pointwise pointed functions $f_* : A_* \to C_*$ and $g_* : B_* \to D_*$, we define the map $f_* \wedge^{\mathsf{pt}}_* g_* \in (A_* \wedge B_*) \to_* (C_* \wedge D_*)$ as the "shadow" of its parametric analogue (defined in Definition 10.5.6).

$$g_* \wedge^{\mathsf{pt}}_* f_* :\equiv \blacklozenge_*(\mathsf{mod}(\wedge\text{-disc} \circ_* (\mathsf{unmod}(\lozenge_* g_*) \wedge_* \mathsf{unmod}(\lozenge_* f_*)) \circ_* \wedge\text{-disc}^{-1}))$$

Now we have the main theorem, which proceeds exactly as with commutativity.

Theorem 15.4.13. $\mathsf{assoc}_{\mathsf{pt}}$ is a family of isomorphisms and satisfies the pentagon identity.

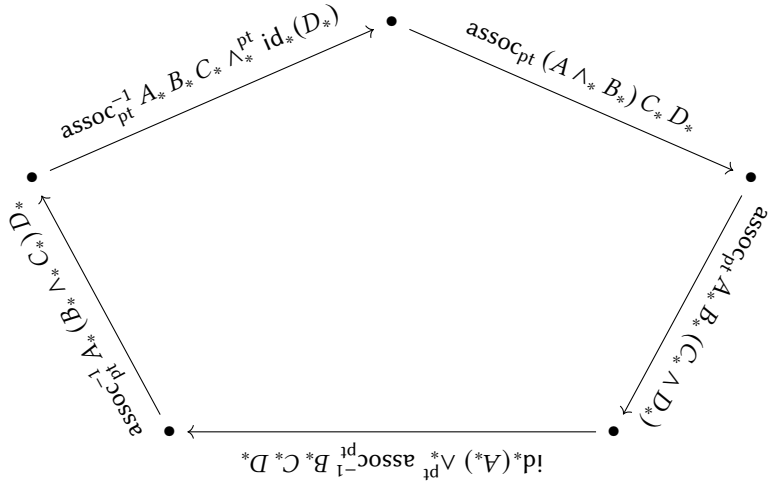
Proof. To show that $\mathsf{assoc}_{\mathsf{pt}}$ is an isomorphism, it suffices to construct two paths as follows for every $A_*, B_*, C_* : \mathsf{U}$.

$$\mathsf{assoc}^{-1}_{\mathsf{pt}}\, A_*\, B_*\, C_* \circ_* \mathsf{assoc}_{\mathsf{pt}}\, A_*\, B_*\, C_* \rightsquigarrow \mathsf{id}_*((A_* \wedge_* B_*) \wedge_* C_*)$$
$$\mathsf{assoc}_{\mathsf{pt}}\, A_*\, B_*\, C_* \circ_* \mathsf{assoc}^{-1}_{\mathsf{pt}}\, A_*\, B_*\, C_* \rightsquigarrow \mathsf{id}_*(A_* \wedge_* (B_* \wedge_* C_*))$$

Without loss of generality, we restrict attention to the first. As in the proof of Theorem 15.4.9, the functoriality of $\blacklozenge_*$ allows us to reduce our goal to proving that the following parametrically polymorphic function is the identity.

$$\lambda X_*.\, \lambda Y_*.\, \lambda Z_*.\, \mathsf{unmod}(\mathsf{assoc}^{-1})\, X_*\, Y_*\, Z_* \circ_* \mathsf{unmod}(\mathsf{assoc})\, X_*\, Y_*\, Z_*$$
$$\pitchfork$$
$$(X_*, Y_*, Z_* : \mathsf{U}_*) \to (X_* \wedge_* Y_*) \wedge_* Z_* \to_* (X_* \wedge_* Y_*) \wedge_* Z_*$$

We finish by applying Theorem 10.5.11. Finally, the pentagon identity asserts that the following round-trip composite is the identity function on $((A_* \wedge_* B_*) \wedge_* C_*) \wedge_* D_*$.



Using the fact that $\blacklozenge_*$ commutes with identities, composition, and the action of the smash product on pointed functions (that is, converts $\wedge_*$ to $\wedge^{\mathsf{pt}}_*$), we again reduce this to an equation on a composite of parametric functions and apply Theorem 10.5.11. □

These first few coherences give a sense of the effectiveness and limitations of our approach. The method is easiest to apply when all the constructions involved in the coherence are induced from parametric constructions. In the statement of the pentagon identity, for example, we use id_*, $\circ_*$, $\wedge^{\mathsf{pt}}_*$, $\mathsf{assoc}_{\mathsf{pt}}$, and $\mathsf{assoc}^{-1}_{\mathsf{pt}}$. We have defined the latter three terms as the shadows of parametric constructions, making the relationship between the parametric and pointwise equivalents obvious. For id_* and $\circ_*$, we instead require a lemma (Proposition 15.4.8) connecting the naive pointwise definition to the shadow of some parametric term. For low-dimensional constructions like these two, the latter is feasible; for a term like the associator, on the other hand, it would be much more difficult to relate the "naive" pointwise definition to the shadow of $\mathsf{assoc}_{\mathsf{pt}}$. On the other hand, the exact definition of $\mathsf{assoc}_{\mathsf{pt}}$ is less likely to be important to future "non-free" theorems

than the definition of $\circ_*$. There is thus an intuitive trade-off: while we can easily exploit "free" theorems by using the shadows of parametric definitions directly, such definitions are more difficult to reason with in the pointwise mode.

Chapter 16

Formalism

Building on the parametric formalism and presheaf model developed in Chapter 11, we sketch an extension to a modal parametric type theory. Following the framework developed in Licata and Shulman's *adjoint logic* [LS16] and used in Gratzer et al.'s **MTT** [GKNB20], we express the properties of the context modalities compactly by formulating a *mode theory*.[1] This consists of the two judgments m mode and $\mu : m \to n$ we have already encountered as well as a *2-cell* judgment $\alpha :: \mu \Rightarrow \nu : m \to n$ specifying maps between modalities, which together constitute a definition of a strict 2-category [JY20, §2.3]. Each 2-cell $\alpha :: \mu \Rightarrow \nu : m \to n$ will induce a transformation $\Gamma'.\mu \vdash \gamma \otimes \mu : \Gamma.\mu \mathbin{@} m$ between modal contexts. We also annotate each of the previously-existing judgments with a mode.

Judgment	Presuppositions	Reading
m mode		m is a mode
$\mu : m \to n$	(m, n mode)	μ is a modality from m to n
$\alpha :: \mu \Rightarrow \mu' : m \to n$	($\mu, \mu' : m \to n$)	α is a 2-cell from μ to μ'
Γ ctx $@\ m$	(m mode)	Γ is a context at mode m
$\vdots$	$\vdots$	$\vdots$

The logic of modalities we use—the set of rules mediating $-.\mu$ and modal hypotheses—is a mechanical specialization of the **MTT** framework. The negative treatment of the modal type operators Disc and Glo is novel, though it might be viewed as an algebraicization of a Fitch-style calculus [Clo18; BCMEPS20]. The discrete type we use is a restricted version of **MTT**'s formulation of modal types, as discussed in Section 14.4.2.

[1] In those works, the aim is to define a general formalism that can be instantiated at various mode theories, while we are interested only in the specific mode theory describing the cohesive relationship between pointwise and parametric modes. Nevertheless, specification in terms of mode theory judgments is convenient for expressing the functoriality of modalities and naturality of transformations between them concisely.

Mode theory As always, we have two modes, the pointwise and the parametric.

$$\frac{}{\mathsf{par\ mode}} \qquad \frac{}{\mathsf{pt\ mode}}$$

We seed the modality judgment with the three basic modalities, then generate a category structure by including an identity id and composition $- \otimes -$.

$$\frac{}{\mathsf{cc} : \mathsf{pt} \to \mathsf{par}} \qquad \frac{}{\mathsf{dsc} : \mathsf{par} \to \mathsf{pt}} \qquad \frac{}{\mathsf{glo} : \mathsf{pt} \to \mathsf{par}} \qquad \frac{}{\mathsf{id} : m \to m}$$

$$\frac{\nu : n \to p \qquad \mu : m \to n}{\nu \otimes \mu : m \to p} \qquad \frac{\mu : m \to n}{\mu \otimes \mathsf{id} = \mu : m \to n} \qquad \frac{\mu : m \to n}{\mathsf{id} \otimes \mu = \mu : m \to n}$$

$$\frac{\pi : p \to q \qquad \nu : n \to p \qquad \mu : m \to n}{(\pi \otimes \nu) \otimes \mu = \pi \otimes (\nu \otimes \mu) : m \to q}$$

The 2-cell judgment, $\alpha :: \mu \Rightarrow \mu' : m \to n$, codifies the adjunctions between the three basic modalities and the fact that cc and glo both cancel dsc up to isomorphism. To avoid some repetition, we encapsulate the basic adjoint relationships by an auxiliary judgment $m : \mu \dashv \nu : n$ (presupposing $\mu : m \to n$ and $\nu : n \to m$).

$$\frac{}{\mathsf{pt} : \mathsf{cc} \dashv \mathsf{dsc} : \mathsf{par}} \qquad \frac{}{\mathsf{par} : \mathsf{dsc} \dashv \mathsf{glo} : \mathsf{pt}}$$

The 2-cell judgment is then required to support the following morphisms: unit and counit transformations for each adjunction, inverses for these in the $\mathsf{dsc} \otimes \mathsf{cc}$ and $\mathsf{dsc} \otimes \mathsf{glo}$ cases, and vertical and horizontal composition operations endowing the mode theory with the structure of a (strict) 2-category. (The former is "ordinary" composition of 2-cells, the latter the action of $- \otimes -$ on 2-cells).

$$\frac{m : \mu \dashv \nu : n}{\mathsf{unit} :: \mu \otimes \nu \Rightarrow \mathsf{id} : n \to n} \qquad \frac{m : \mu \dashv \nu : n}{\mathsf{cou} :: \mathsf{id} \Rightarrow \nu \otimes \mu : m \to m}$$

$$\frac{}{\mathsf{cou}^{-1} :: \mathsf{dsc} \otimes \mathsf{cc} \Rightarrow \mathsf{id} : \mathsf{pt} \to \mathsf{pt}} \qquad \frac{}{\mathsf{unit}^{-1} :: \mathsf{id} \Rightarrow \mathsf{dsc} \otimes \mathsf{glo} : \mathsf{pt} \to \mathsf{pt}}$$

$$\frac{\alpha' :: \mu' \Rightarrow \mu'' : m \to n \qquad \alpha :: \mu \Rightarrow \mu' : m \to n}{\alpha' \circ \alpha :: \mu \Rightarrow \mu'' : m \to n}$$

$$\frac{\beta :: \nu \Rightarrow \nu' : n \to p \qquad \alpha :: \mu \Rightarrow \mu' : m \to n}{\beta \otimes \alpha :: \nu \otimes \mu \Rightarrow \nu' \otimes \mu' : m \to p}$$

These are required to satisfy various equations we will not list explicitly: triangle equalities relating the unit and counit of each adjunction, the fact that cou^{-1} and unit^{-1} provide the inverses suggested by the notation, and the laws of a strict 2-category [JY20, Proposition 2.3.4].

Contexts The collection of context formers is now extended with the application of modalities, modal hypotheses, and endpoint hypotheses. Like other operations defined by recursion in the computational framework—substitution, interval restriction—the application of a modality here becomes a primitive context former.

$$\frac{\Gamma\ \mathsf{ctx} \mathbin{@} n \qquad \mu : m \to n}{\Gamma.\mu\ \mathsf{ctx} \mathbin{@} m} \qquad \frac{\mu : m \to n \qquad \Gamma.\mu \vdash A\ \mathsf{type} \mathbin{@} m}{\Gamma.(\mu \mid A)\ \mathsf{ctx} \mathbin{@} n} \qquad \frac{\Gamma\ \mathsf{ctx} \mathbin{@} m}{\Gamma.\mathbf{2}\ \mathsf{ctx} \mathbin{@} m}$$

$$\frac{}{\Gamma.\mathsf{id} = \Gamma\ \mathsf{ctx} \mathbin{@} m} \qquad \frac{\nu : n \to p \qquad \mu : m \to n}{\Gamma.(\nu \otimes \mu) = \Gamma.\nu.\mu\ \mathsf{ctx} \mathbin{@} m}$$

Modalities in substitutions Each modality has a functorial action on substitutions, and each 2-cell moreover induces a substitution between modal contexts. These are required to preserve the 2-categorical structure—for example, we require $\Gamma' \vdash \gamma \otimes \mathsf{id} = \gamma : \Gamma \mathbin{@} m$—and we ask that each substitution $\{\alpha\}$ satisfies a naturality condition as shown below.

$$\frac{\mu : m \to n \qquad \Gamma' \vdash \gamma : \Gamma \mathbin{@} n}{\Gamma'.\mu \vdash \gamma \otimes \mu : \Gamma.\mu \mathbin{@} m} \qquad \frac{\alpha :: \mu \Rightarrow \nu : m \to n}{\Gamma.\nu \vdash \{\alpha\} : \Gamma.\mu \mathbin{@} m}$$

$$\frac{\alpha :: \mu \Rightarrow \nu : m \to n \qquad \Gamma' \vdash \gamma : \Gamma}{\Gamma'.\nu \vdash \{\alpha\} \circ (\gamma \otimes \nu) = (\gamma \otimes \mu) \circ \{\alpha\} : \Gamma.\mu \mathbin{@} m}$$

The rule for forming substitutions into a modal hypothesis matches the computational definition, and the variable rule is as in Theorem 14.3.15.

$$\frac{\mu : m \to n \qquad \Gamma' \vdash \gamma : \Gamma \mathbin{@} n \qquad \Gamma.\mu \vdash A\ \mathsf{type} \mathbin{@} m \qquad \Gamma'.\mu \vdash M : A[\gamma \otimes \mu] \mathbin{@} m}{\Gamma' \vdash \gamma.M : \Gamma.(\mu \mid A) \mathbin{@} n}$$

$$\frac{\mu : m \to n \qquad \Gamma.\mu \vdash A\ \mathsf{type} \mathbin{@} m}{\Gamma.(\mu \mid A).\mu \vdash \mathsf{v} : A[\mathsf{p} \otimes \mu] \mathbin{@} m}$$

Endpoints and interval The bridge endpoint object is populated with the two endpoints, and is included in the bridge interval by way of a "boundary" substitution ∂.

$$\frac{}{\Gamma \vdash \mathbf{0}_2 : \Gamma.\mathbf{2} \mathbin{@} m} \qquad \frac{}{\Gamma \vdash \mathbf{1}_2 : \Gamma.\mathbf{2} \mathbin{@} m} \qquad \frac{\Gamma \text{ ctx} \mathbin{@} \text{par}}{\Gamma.\mathbf{2} \vdash \partial : \Gamma.\mathbf{I} \mathbin{@} \text{par}}$$

We express the interaction between the interval and action of modalities by a series of isomorphisms (up to judgmental equality) corresponding to clauses of Figure 14.1. (While these isomorphisms are equalities in the computational interpretation, we do not expect this to be the case in all models.) For example, endpoint hypotheses (and path interval hypotheses) commute with all modalities, while cc collapses interval hypotheses and glo turns them into endpoint hypotheses.

$$\frac{\Gamma \text{ ctx} \mathbin{@} n \qquad \mu : m \longrightarrow n}{\Gamma.\mathbf{2}.\mu \vdash \mathsf{ex}_{\mu 2} : \Gamma.\mu.\mathbf{2} \mathbin{@} m} \qquad \frac{\Gamma \text{ ctx} \mathbin{@} n \qquad \mu : m \longrightarrow n}{\Gamma.\mu.\mathbf{2} \vdash \mathsf{ex}_{2\mu} : \Gamma.\mathbf{2}.\mu \mathbin{@} m}$$

$$\frac{\Gamma \text{ ctx} \mathbin{@} \text{par}}{\Gamma.\mathsf{cc} \vdash \mathsf{cc}_{\mathbf{I}} : \Gamma.\mathbf{I}.\mathsf{cc} \mathbin{@} \text{pt}} \qquad \frac{\Gamma \text{ ctx} \mathbin{@} \text{par}}{\Gamma.\mathbf{I}.\mathsf{glo} \vdash \mathsf{glo}_{\mathbf{I}} : \Gamma.\mathsf{glo}.\mathbf{2} \mathbin{@} \text{pt}}$$

We ask that the two exchange substitutions $\mathsf{ex}_{\mu 2}$ and $\mathsf{ex}_{\mu 2}$ are mutually inverse (as are their equivalents for paths) and that $\mathsf{cc}_{\mathbf{I}}$ and $\mathsf{glo}_{\mathbf{I}}$ invert the substitutions $\mathsf{p}_{\mathbf{I}} \otimes \mathsf{cc}$ and $(\partial \otimes \mathsf{glo}) \circ \mathsf{ex}_{2\mathsf{glo}}$ respectively. We also ask each substitution to be natural in Γ and μ if applicable and interact correctly with interval-related substitutions; for example, we should have $\Gamma.\mu \vdash \mathsf{ex}_{\mu 2} \circ (\mathbf{0}_2 \otimes \mu) = \mathbf{0}_2 : \Gamma.\mu.\mathbf{2} \mathbin{@} m$.

We do *not* impose any additional substitutions specifying the action of the modalities on term hypotheses. It already follows from the existing rules for modalities and term hypotheses that we have the following isomorphisms.

$$\begin{aligned} \Gamma.(\mu \mid A).\mathsf{dsc} &\cong \Gamma.\mathsf{dsc}.(\mathsf{cc} \otimes \mu \mid A[\{\mathsf{cou}\} \otimes \mu]) \\ \Gamma.(\mu \mid A).\mathsf{glo} &\cong \Gamma.\mathsf{glo}.(\mathsf{dsc} \otimes \mu \mid A[\{\mathsf{cou}\} \otimes \mu]) \\ \Gamma.(\mathsf{cc} \otimes \mu \mid A).\mathsf{cc} &\cong \Gamma.\mathsf{cc}.(\mu \mid A) \end{aligned}$$

That cc completely removes hypotheses not typed under cc, meanwhile, is not something we want to require in all models (and indeed fails in the cubical set model described below).

Negative modal types The two negative modal types—Glo and Codisc—are specified by rules following those we proved in Section 14.4.1. We show the rules for the global type here, and leave it to the reader to infer the rules for the codiscrete type. With substitutions now explicit, we see how the reduction and uniqueness equations involve the unit and

counit substitutions. We note again the similarity between these rules and the rules for the bridge type, with dsc playing the rule of $-.\mathbf{I}$ and cc the role of $-.\backslash \boldsymbol{r}$.

$$\frac{\Gamma.\mathsf{dsc} \vdash A \ \mathsf{type} \ @ \ \mathsf{par}}{\Gamma \vdash \mathsf{Glo}(A) \ \mathsf{type} \ @ \ \mathsf{pt}} \qquad \frac{\Gamma.\mathsf{dsc} \vdash M : A \ @ \ \mathsf{par}}{\Gamma \vdash \mathsf{mod}(M) : \mathsf{Glo}(A) \ @ \ \mathsf{pt}}$$

$$\frac{\Gamma.\mathsf{cc}.\mathsf{dsc} \vdash A \ \mathsf{type} \ @ \ m \qquad \Gamma.\mathsf{cc} \vdash M : \mathsf{Glo}(A) \ @ \ n}{\Gamma \vdash \mathsf{unmod}(M) : A[\{\mathsf{unit}\}] \ @ \ m}$$

$$\frac{\Gamma.\mathsf{cc}.\mathsf{dsc} \vdash A \ \mathsf{type} \ @ \ m \qquad \Gamma.\mathsf{cc}.\mathsf{dsc} \vdash M : A \ @ \ m}{\Gamma \vdash \mathsf{unmod}(\mathsf{mod}(M)) = M[\{\mathsf{unit}\}] : A[\{\mathsf{unit}\}] \ @ \ m}$$

$$\frac{\Gamma.\mathsf{dsc} \vdash A \ \mathsf{type} \ @ \ m \qquad \Gamma \vdash M : \mathsf{Glo}(A) \ @ \ n}{\Gamma \vdash M = \mathsf{mod}(\mathsf{unmod}(M[\{\mathsf{cou}\}])) : \mathsf{Glo}(A) \ @ \ n}$$

Discrete type Finally, our formal rules for the discrete type likewise mimic the suite of rules proven in Section 14.4.2.

$$\frac{\Gamma.\mathsf{cc} \vdash A \ \mathsf{type} \ @ \ \mathsf{par}}{\Gamma \vdash \mathsf{Disc}(A) \ \mathsf{type} \ @ \ \mathsf{pt}} \qquad \frac{\Gamma.\mathsf{cc} \vdash M : A \ @ \ \mathsf{par}}{\Gamma \vdash \mathsf{mod}(M) : \mathsf{Disc}(A) \ @ \ \mathsf{pt}}$$

$$\frac{\begin{array}{c}\Gamma.\mathsf{cc} \vdash A \ \mathsf{type} \ @ \ \mathsf{pt} \qquad \Gamma.\mathsf{Disc}(A) \vdash B \ \mathsf{type} \ @ \ \mathsf{par} \\ \Gamma \vdash P : \mathsf{Disc}(A) \ @ \ \mathsf{par} \qquad \Gamma.(\mathsf{cc} \mid A) \vdash N : B[\mathsf{p}.\mathsf{mod}(\mathsf{v})] \ @ \ \mathsf{par}\end{array}}{\Gamma \vdash \mathsf{letdisc}(B, P, N) : B[\mathsf{id}.P] \ @ \ \mathsf{par}}$$

$$\frac{\begin{array}{c}\Gamma.\mathsf{cc} \vdash A \ \mathsf{type} \ @ \ \mathsf{pt} \qquad \Gamma.\mathsf{Disc}(A) \vdash B \ \mathsf{type} \ @ \ \mathsf{par} \\ \Gamma.\mathsf{cc} \vdash M : A \ @ \ \mathsf{par} \qquad \Gamma.(\mathsf{cc} \mid A) \vdash N : B[\mathsf{p}.\mathsf{mod}(\mathsf{v})] \ @ \ \mathsf{par}\end{array}}{\Gamma \vdash \mathsf{letdisc}(B, \mathsf{mod}(M), N) = N[\mathsf{id}.M] : B[\mathsf{id}.\mathsf{mod}(M)] \ @ \ \mathsf{par}}$$

16.1 Cubical set model

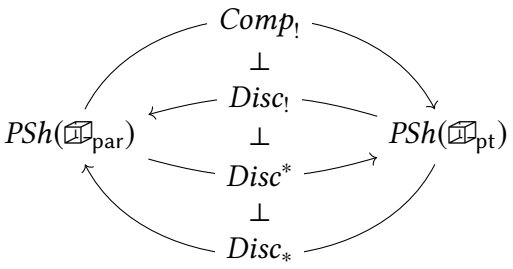
To construct a model in cubical sets, we combine the pointwise and parametric models described in Sections 3.3.1 and 11.1 respectively. We interpret judgments in the pointwise mode as statements about the category $PSh(\square_c)$ of cartesian cubical sets and judgments in the parametric mode as statements about the category $PSh(\square_{c\times a})$ of cartesian-affine bicubical sets. Henceforth we rename $\square_c$ and $\square_{c\times a}$ to $\square_{\mathsf{pt}}$ and $\square_{\mathsf{par}}$ respectively to reflect their roles.

Modal operators As with the interpretation of bridge interval context extension and restriction, we derive the modal context operators from functors between the two cube categories. We have two such functors, the connected components functor $Comp : \square_{\text{par}} \to \square_{\text{pt}}$ and discrete embedding $Disc : \square_{\text{pt}} \to \square_{\text{par}}$, both obtained by assembling the operations defined in Figures 14.1 and 14.2.

$$Comp(\Psi) \coloneqq \Psi.\text{cc} \qquad Disc(\Psi) \coloneqq \Psi.\text{dsc}$$
$$Comp(\Psi' \Vdash \psi \in \Psi) \coloneqq (\psi : \Psi) \otimes \text{cc} \qquad Disc(\Psi' \Vdash \psi \in \Psi) \coloneqq (\psi : \Psi) \otimes \text{dsc}$$

Per Proposition 14.2.14, *Comp* is left adjoint to *Disc*. Note that the global sections operator cannot be defined as a map between the cube categories: the category $\square_{\text{pt}}$ contains no "endpoint object".

As described in Section 11.1, a functor $F : \mathcal{C} \to \mathcal{D}$ between index categories induces an adjoint triple $F_! \dashv F^* \dashv F_*$ between the presheaf categories $PSh(\mathcal{C})$ and $PSh(\mathcal{D})$, with the central functor $F^* : PSh(\mathcal{D}) \to PSh(\mathcal{C})$ given by precomposition—$F^*(P)(c) = G(P(c))$—and $F_!, F_* : PSh(\mathcal{C}) \to PSh(\mathcal{D})$ by left and right Kan extension respectively. Applying with *Comp* and *Disc*, we have in particular the following adjoint quadruple, our cohesion situation. Here we also use the fact that $Comp \dashv Disc$ implies $Comp_! \dashv Disc_!$.



We interpret contexts in the pointwise mode as objects of $PSh(\square_{\text{pt}})$ and contexts in the parametric mode as objects of $PSh(\square_{\text{par}})$. The first three functors of the quadruple above accordingly implement the three modal operators on contexts: we interpret $-.\text{cc}$ by $Comp_!$, $-.\text{dsc}$ by $Disc_!$, and $-.\text{glo}$ by $Disc^*$. Recall again that $F_!(\text{ш}(c)) \cong \text{ш}(F(c))$, so we know the connected components and discrete functors have the desired behavior on interval hypotheses: $Comp_!(\mathbf{I}) \cong \text{ш}(\,\cdot\,)$, $Disc_!(\mathbb{I}_{\text{pt}}) \cong \mathbb{I}_{\text{par}}$, and so on. (We henceforth use pt and par subscripts to disambiguate between objects in $PSh(\square_{\text{pt}})$ and $PSh(\square_{\text{par}})$ when necessary.) We can also quickly check that the connected components and global sections functors cancel the discrete embedding, using formal properties of $(-)_!$ and $(-)^*$.

$$Comp_! \circ Disc_! \cong (Comp \circ Disc)_! \cong (id_{\square_{\text{pt}}})_! \cong id_{PSh(\square_{\text{pt}})}$$
$$Disc^* \circ Disc_! \cong Disc^* \circ Comp^* \cong (Comp \circ Disc)^* \cong (id_{\square_{\text{pt}}})^* \cong id_{PSh(\square_{\text{pt}})}$$

Endpoints We may define the endpoint objects in each mode by the judgment defined in Chapter 14: we set $\mathbf{2}_m(\Psi) \coloneqq \{\boldsymbol{r} \mid \Psi \Vdash \boldsymbol{r} \in \mathbf{I} \mathbin{@} m\}$. We then interpret endpoint context extension by product: $[\![\Gamma.\mathbf{2}]\!] \coloneqq [\![\Gamma]\!] \times \mathbf{2}_m$ for Γ ctx $@\ m$, and terms $\Gamma \vdash \boldsymbol{r} : \mathbf{2} \mathbin{@} m$ by maps $[\![\boldsymbol{r}]\!] \in [\![\Gamma]\!] \to \mathbf{2}_m$. We can check that the global sections functor $Disc^*$ takes $\mathbf{I}$ to $\mathbf{2}_{\mathsf{pt}}$ as follows.

$$Disc^*(\mathbf{I})(\Psi) = \mathbf{I}(\Psi.\mathsf{dsc}) \cong \{\boldsymbol{r} \mid \Psi.\mathsf{dsc} \Vdash \boldsymbol{r} \in \mathbf{I} \mathbin{@} \mathsf{par}\} \cong \{\boldsymbol{r} \mid \Psi \Vdash \boldsymbol{r} \in \mathbf{2} \mathbin{@} \mathsf{pt}\} = \mathbf{2}_{\mathsf{pt}}(\Psi)$$

We can also characterize each endpoint object as the coproduct $\mathbf{2}_m \cong \mathbf{1}_m + \mathbf{1}_m$ of two copies of the terminal presheaf $\mathbf{1}_m(\Psi) = \{\star\}$: there are two endpoints in any interval context. Being left adjoints, $Comp_!$, $Disc_!$, and $Disc^*$ all preserve coproducts. $Disc_!$ and $Disc^*$ are also right adjoints and thus preserve terminal objects, and we can manually check that $Comp_!(\mathbf{1}_{\mathsf{pt}}) \cong Comp_!(ﾖ(\,\cdot\,)) \cong ﾖ(\,\cdot\,) \cong \mathbf{1}_{\mathsf{par}}$. It follows that the three preserve the endpoint object as required.

By interpreting the three basic modalities as above and composites by composition of functors, we can interpret each $\mu : m \to n$ by a functor $[\![\mu]\!] : PSh(\square_n) \to PSh(\square_m)$, and so define $[\![\Gamma.\mu]\!] \coloneqq [\![\mu]\!]([\![\Gamma]\!])$.

Modal hypotheses Given a modality $\mu : m \to n$, a semantic context $G \in PSh(\square_n)$, and a semantic pretype T over a $[\![\mu]\!](G)$, we will define a new semantic pretype $(\mu \mid T)$ over G. Let us first consider the special case $\mu = \mathsf{cc}$. We may define $(\mathsf{cc} \mid T)$ as follows.

$$(\mathsf{cc} \mid T)(\Psi, g) \coloneqq T(\Psi.\mathsf{cc}, [\![\mathsf{cc}]\!](g))$$
$$(\mathsf{cc} \mid T)(\psi, g) \coloneqq T((\psi : \Psi) \otimes \mathsf{cc}, [\![\mathsf{cc}]\!](g))$$

Here we make implicit use of the Yoneda lemma [Mac98, §III.2]: for any presheaf $G \in PSh(C)$, the elements of $G(c)$ are in (natural) correspondence with morphisms $ﾖ(c) \to G$, with any $g \in G(c)$ inducing $\alpha : ﾖ(c) \to G$ defined by $\alpha(d)(f) \coloneqq G(f)(g)$ and any $\alpha : ﾖ(c) \to G$ inducing $\alpha(c)(id_c) \in G(c)$. In the above, we first regard $g \in G(\Psi)$ as a morphism $g : ﾖ(\Psi) \to G$, apply the functorial action of cc to obtain a morphism $[\![\mathsf{cc}]\!](g) : ﾖ(\Psi.\mathsf{cc}) \cong [\![\mathsf{cc}]\!](ﾖ(\Psi)) \to [\![\mathsf{cc}]\!](G)$, then apply the Yoneda lemma once more to regard this as an element $[\![\mathsf{cc}]\!](g) \in [\![\mathsf{cc}]\!](G)(\Psi.\mathsf{cc})$. The effect, analogously to the computational setting, is that an element of $(\mathsf{cc} \mid T)$ in some context instantiation g is an element of T over the connected component of G to which g belongs.

This definition relies on the fact that cc takes interval contexts to interval contexts; this is not the case for all modalities, thanks to the presence of glo. In the general case, we compensate by quantifying over all closing substitutions using a categorical limit.

$$(\mu \mid T)(\Psi, g) \coloneqq \lim \left(T(\Psi', [\![\mu]\!](g) \circ h) \;\middle|\; \Psi' \in \square_m, h : ﾖ(\Psi') \to [\![\mu]\!](ﾖ(\Psi))\right)$$

Explicitly, an element of $(\mu \mid T)(\Psi, g)$ is a family of terms $t_{\Psi',h} \in T(\Psi', [\![\mu]\!](g) \circ h)$ indexed by contexts Ψ' and closing substitutions $h : ⅄(\Psi') \to [\![\mu]\!](⅄(\Psi))$ and satisfying the property that $T(\psi, h)(t_{\Psi',h}) = t_{\Psi'',h \circ ⅄(\psi)}$ for every $\Psi'' \Vdash \psi \in \Psi' \mathbin{@} m$.

Finally, the extension $G.(\mu \mid T) \in PSh(\square_n)$ of a context G by a modal hypotheses T over $[\![\mu]\!](G)$ is defined as the ordinary context extension by $(\mu \mid T)$.

$$(G.(\mu \mid T))(\Psi) \coloneqq \sum_{g \in G(\Psi)} (\mu \mid T)(\Psi, g)$$
$$(G.(\mu \mid T))(\psi)(g, t) \coloneqq (G(\psi)(g), (\mu \mid T)(\psi, g)(t))$$

Modal types We can interpret the two right adjoint modal types using the modal hypothesis pretypes already defined. Given a semantic type T over $[\![\mathrm{dsc}]\!](G)$, we define the semantic type $\mathrm{Glo}(T) \coloneqq (\mathrm{dsc} \mid T)$; given T over $[\![\mathrm{glo}]\!](G)$, we likewise define $\mathrm{Codisc}(T) \coloneqq (\mathrm{glo} \mid T)$. We leave it to the reader to reconstruct the interpretations of the introduction and elimination rules and Kan operators following their computational definitions.

For the discrete type, we must close $(\mathrm{cc} \mid T)$ under formal homogeneous composites. Just as we construct the value relation for the computational type $\mathrm{Disc}(A)$ as the least fixed-point of a process adding formal composite values, we can arrive at $\mathrm{Disc}(T)$ as a sequential colimit of presheaves beginning with $(\mathrm{cc} \mid T)$ and adding a layer of formal composites in each step. Alternatively, a second method of constructing cubical sets with formal composites can be found in [CHM18, §2.4] in the context of constructing higher inductive types.

Chapter 17

Conclusions

17.1 Related work

Cohesive type theory Lawvere's axiomatic cohesion [Law07] defines an abstract, categorical setting in which the objects of one category may be regarded as "spaces" whose "points" are drawn from another category. This framework was first applied in type theory by Schreiber and Shulman [SS12], in the form of an extension of **HoTT** by axioms capturing some consequences of a cohesive situation, in pursuit of synthetic quantum field theory. Shulman [Shu18] proceeded to develop a second theory, this one extending homotopy type theory by a combination of axioms and modal judgmental structure, to more precisely capture the axioms of cohesion.

Shulman's aim is to address **HoTT**'s inability to reason about non-homotopy-invariant constructions, *i.e.*, constructions that do not support coercion. His theory combines the homotopical structure of **HoTT** with a second layer of topological structure, the two layers interacting via cohesion. This enables the use of **HoTT**-style synthetic homotopy theory in the service of topological theorems, Brouwer's fixed-point theorem being the showcase example. Extensions to Shulman's theory incorporating additional modalities have been further used to capture *differential* topological structure [GLNPRSW17; Wel18] building on ideas of Schreiber [Sch13]. On a different note, Kavvos has studied connections between cohesion and calculi for information flow [Kav19].

A major difference between our work and Shulman's is that our cohesion is defined around an explicit judgmental (bridge) interval: the connected components functor collapses bridges, the global sections functor returns the type of elements in an empty bridge context, and so on. In contrast, Shulman's judgmental structure only includes modal features; the connection to topology is established via axioms relating the modal operators to the type of real numbers.

A more mundane difference is that we explicitly include two modes (pt and par). Shul-

man's theory instead takes place entirely in the "cohesive mode", our par, taking the composite operators $\flat(-) :\equiv \mathsf{Disc}(\mathsf{Glo}(-))$ and $\sharp(-) :\equiv \mathsf{Codisc}(\mathsf{Glo}(-))$ as the primitive modal types. It is still possible to speak of "pointwise" types in such a theory: roughly, they are the types A such that a canonical map $\flat A \to A$ is an isomorphism [Shu18, Definition 6.12]. Given that our objective is to use parametricity in the service of pointwise theorems, however, we consider it clearer to give the pointwise world the status of a full-fledged mode. Shulman's formulation of the two type formers $\flat$ and $\sharp$ nevertheless bears clear similarities to our own modal types: $\sharp$ is specified by a negative elimination rule, while $\flat$ is positive and relies on a notion of *crisp hypothesis* paralleling our modal hypotheses.

In addition to the $\flat$ and $\sharp$ modalities, Shulman's theory also includes a third *shape modality*, written $\int$, which corresponds to a composite $\mathsf{Disc}(\mathsf{CComp}(-))$ not available as a type former in our theory. In Shulman's theory, $\int A$ is defined as the *localization* of A at the map $\mathbb{R} \to \mathsf{Unit}$, where $\mathbb{R}$ is type of the Dedekind real numbers; this is to say that $\int A$ is defined by contracting all images of $\mathbb{R}$ in A (all "topological" paths in A), an operation that may be effected by a higher inductive type [Shu18, Definition 9.6; RSS20, Definition 2.14]. It is then postulated, indirectly, that this operation is left adjoint to $\flat$ [Shu18, Axiom C0, Theorem 9.15]. If our schema for higher inductive types were extended to allow bridge types in recursive arguments, it might be possible to similarly obtain a connected component type former in our theory by localizing at "$\mathbf{I} \to \mathsf{Unit}$"; however, it is not clear that this would relate correctly to the primitive cohesive type formers.

There is, in any case, a major difference in focus. We concentrate on providing a computational justification for our theory, which is largely orthogonal to Shulman's goals; he uses axioms freely in the specification of his theory, although he makes an effort to be constructive whenever possible. The direction of application is also reversed: Shulman uses homotopical methods to prove topological results, *i.e.*, results in the cohesive mode, whereas we use cohesive methods (parametricity) to prove results in the pointwise mode.

Modal type theory Modalities more generally have a storied history in type theory and logic. The question of how to represent connectives like our cohesive operators that enact a change of context has long been a source of consternation among proof theorists; only relatively recently have well-behaved and general techniques for specifying modal logics and type theories begun to appear.

Early modal logics, emanating from Lewis and Langford's seminal axiomatizations [LL32], are typically concerned with capturing *necessity* and *possibility*, introducing connectives $\Box A$ and $\Diamond A$ to represent the propositions "A is necessary" and "A is possible". Another canonical application is to temporal reasoning, for example with connectives representing "A holds later" or "A holds at time t". We refer to [Sim94; PGM04; Kav16] for detailed surveys of constructive modal logics in particular. Modal *dependent* type theories

are much more recent [PR15; Shu18; GSB19; Zwa19; BCMEPS20], as the more complex structure of a dependent context naturally complicates the treatment of modalities. Similar issues arise in efforts to define dependent *substructural* type theories [CP02; Vák14; KPB15], which, like modal type theories, place restrictions on how variables can be accessed from the context.

Fortunately for us, frameworks for defining modal type theories have recently begun to crop up. Licata and Shulman [LS16] introduced the concept of a *mode theory*, a 2-category with modes as objects, modalities as morphisms, and maps between modalities as 2-cells, as a way of specifying a system of modalities. Their work builds on that of Reed [Ree09], who considers the special case of preorder mode theories. The generalization was motivated in particular by cohesive type theory, which requires the two parallel modalities $\mathsf{cc}, \mathsf{glo} : \mathsf{pt} \to \mathsf{par}$. We use such a mode theory in the specification of our formalism in Chapter 16. Licata, Shulman, and Riley [LSR17] further generalize the Licata-Shulman framework to capture substructural phenomena.

The mode theory machinery was picked up by Gratzer, Kavvos, Nuyts, and Birkedal [GKNB20] in their *multimodal type theory* (**MTT**), a framework for *dependent* modal type theories. While our theory takes advantage of various simplifications appropriate to our special case, **MTT** has been tremendously useful as a template. Our formulation of modal hypotheses in particular is taken directly from **MTT**.

Our eliminator for discrete types is a restricted version of the **MTT** eliminator, which would additionally permit the principal argument (P below) to be supplied beneath an auxiliary modality.

MTT-Elim

$$\frac{\nu : \mathsf{par} \to m \qquad \Gamma.\nu.\mathsf{cc} \gg A\ \mathsf{type} \mathbin{@} \mathsf{pt} \qquad \Gamma, (\nu \mid d : \mathsf{Disc}(A)) \gg B\ \mathsf{type} \mathbin{@} m \\ \Gamma.\nu \gg P \in \mathsf{Disc}(A) \mathbin{@} \mathsf{par} \qquad \Gamma, (\nu, \mathsf{cc} \mid a : A) \gg N \in B[\mathsf{mod}(a)/d] \mathbin{@} m}{\Gamma \gg \mathsf{letdisc}_\nu(d.B, P, a.N) \in B[P/d] \mathbin{@} m}$$

This parameter is necessary in general to take advantage of interactions between cc and other modalities; note how ν and cc are combined in the hypotheses of N. In the particular case of cohesion, however, the only essential property of modalities of the form (ν, cc) is the equation $\Gamma.(\mathsf{dsc}, \mathsf{cc}) = \Gamma$, and the $\nu = \mathsf{dsc}$ instance of the **MTT** eliminator is *derivable* (Lemma 15.1.1) by use of the codiscrete type. (A similar derivability was observed by Shulman for $\flat$-types [Shu18, Lemma 5.1].) This is fortunate, as the general rule would seriously complicate the computational interpretation. Consider that when the ambient context is an interval context Ψ, the principal argument P is typed in context $\Psi.\nu$. This may not be an interval context: if $\Psi = (\boldsymbol{x} : \mathbf{I})$ and $\nu = (\mathsf{glo}, \mathsf{dsc})$, for example, then we have $\Psi.\nu = (\boldsymbol{x} : 2)$. We would therefore need to be able to evaluate terms in *extended* interval contexts (possibly containing endpoints). We conjecture that a system could be designed in which endpoint hypotheses can appear in genuine interval contexts and split becomes a sheaf condition imposed on types, but our approach seems much simpler.

The type formers Glo and Codisc diverge further from **MTT**, which gives positive eliminators for all modal types. A similar projection rule was at some point considered by Gross et al. [GLNPRSW17] for ♯-types, but to our knowledge this formulation of cohesion is otherwise novel. Similar situations—where a type former has two adjoints to its left, or more generally has a left adjoint that is a *parametric right adjoint*—are analyzed in detail in [GCKGB21]. Such structure has previously appeared implicitly in several type theories of modal character [BGM17; BCMEPS20], as well as in nominal type theory [Che12] and our own treatment of affine interval variables.

The definitions of the modal context operators −.dsc and −.glo in our computational interpretation perform what Nuyts et al. [NVD17] call *left division* on modal term hypotheses, adjusting the modality of each such hypothesis. They trace this style of definition to Pfenning [Pfe01] and Abel (as "inverse application") [Abe08], while the broader idea of marking the hypotheses in a context by different modes or modalities goes back to Avron et al. [AHMP98]. Others have used a *split context* representation wherein the hypotheses at each mode or modality are listed in their own context—see for example [PD01; Kav20].

Internal parametricity à la Nuyts et al. Nuyts, Vezzosi, and Devriese [NVD17], already mentioned in Chapter 12, also use modalities to formulate their type theory for internal parametricity. We follow here the convention introduced in Chapter 12 of referring to their paths as "0-bridges" and bridges as "1-bridges"; we write $\mathbf{I}_0$ and $\mathbf{I}_1$ for the two intervals.

Their system is interpreted in a presheaf category $\mathit{PSh}(\mathsf{BPCube})$ (*bridge-path cubical sets*), which is cohesive over the category $\mathit{PSh}(\square_c)$ of cartesian cubical sets. (In fact there is a string of *five* adjoint functors between them.) Here the objects of $\square_c$ should be thought of as contexts of hypotheses $\boldsymbol{x} : \mathbf{I}_0$, while the objects of BPCube are contexts containing both $\boldsymbol{x} : \mathbf{I}_0$ and $\boldsymbol{x} : \mathbf{I}_1$ hypotheses. BPCube is not the equivalent to the product $\square_c \times \square_c$, however, as there is a map from the 1-bridge interval to the 0-bridge interval. That is, there is some $(\Psi, \boldsymbol{x} : \mathbf{I}_1) \Vdash \psi \in (\Psi, \boldsymbol{x} : \mathbf{I}_0)$. The cohesive situation arising from the inclusion $\square_c \hookrightarrow \mathsf{BPCube}$ then includes functors that send 0-bridges to 1-bridges and vice versa, enabling the expression of parametric functions (which take 1-bridges to 0-bridges) as elements of modal function types. Like Shulman, Nuyts et al. also design their type theory around a single mode, corresponding to our par. The modal operators, like our −.dsc and −.glo, are defined on term hypotheses by left division.

It may be possible, following this setup, to design a version of our theory where there is a map $\mathbf{I} \to \mathbb{I}$. In such a system, the map loosen from paths to bridges could be defined using the judgmental interval structure rather than the Kan operations, which is intuitively appealing. However, complications arise from the affine nature of $\mathbb{I}$. In particular, the restriction operator $- \setminus \boldsymbol{r}$ would need to convert bridge variables to path variables rather than deleting them, with consequences for the formulations of extent and Gel.

Modalities and intervals As mentioned in Section 12.1, Nuyts [Nuy20] shows that V, Gel, and similar types can be derived from the presence of a *transpension type* that is available in all presheaf categories. His system is an instantiation **MTT**; the transpension in particular is the modal type corresponding to a context operator $🔒_{◊r}$ indexed by an interval term. This operator is right adjoint to (possibly affine) context extension $(-, x : \mathbb{I})$, making the transpension type former $\langle ◊\ x \mid - \rangle$ right adjoint to interval quantification $\langle \forall (x : \mathbb{I}) \mid - \rangle$. Nuyts analyzes, among other things, how the properties of assumptions $x : \mathbb{I}$ (such as structurality or affinity) affect the behavior of transpension. In the affine cubical setting, the transpension of a type A is analogous to the type $\mathsf{Gel}_{\boldsymbol{r}}(\mathsf{U}, \mathsf{Unit}, \_.\_.A)$. Nuyts recovers the general Gel-type (there called the Ψ-*combinator*) as a combination of transpension and quantification over the boundary of $\boldsymbol{r}$ [Nuy20, §7.8]. The extent operator (there the Φ-*combinator*) is also obtainable using $◊\ x$.

In our system, the Gel-type resembles, at least informally, a modal type corresponding to the context operator $-.\backslash \boldsymbol{r}$, which is *left* adjoint to context extension by an interval. We speculate that these two views of Gel—as deriving from a left adjoint $- \backslash \boldsymbol{r}$ or right adjoint $🔒_{◊r}$ to context extension—are dual views of the *equivalence* between bridges and types in an interval context.

Gratzer et al. [GCKGB21] also analyze affine interval variables as a modal phenomenon, drawing out the commonalities between our rules for bridge types and the modal type formers Glo and Codisc.

17.2 Outlook

We have now truly fulfilled the promise put forth at the start of Part III: that we can apply parametricity to the analysis of higher inductive types in cubical type theory. We synthesize the recently well-developed body of work on type-theoretic cohesion to relate a model with logical-relational structure to a "basic" model, allowing us to access parametricity results in the pointwise world and thereby translate Reynolds' methodology to Bernardy and Moulin's internal parametricity. We find that while additional effort is required to drag free theorems from the parametric to the pointwise mode, we are still able to take effective advantage of parametricity, as exemplified by our coherence theorems for the smash product.

One question that remains—beyond those already discussed in Section 12.2—is whether formal internal parametricity is able to prove results about a pure cubical formalism. In our presheaf model, the judgments of the pointwise mode are interpreted exactly as they are in their plain cubical equivalents introduced in Part I. The same is nearly the case in the computational interpretation, with the minor exception that the inductively generated universes must now contain Glo types. It is therefore reasonable to say that we can use internal parametricity as a tool to prove results in pointwise settings. We do not

know whether this is true on the level of formalisms: is the modal parametric cubical formalism conservative over the cubical formalism, in the sense that any theorem of the pointwise mode expressible in the extended formalism is already provable in the basic cubical formalism? We conjecture that this is indeed the case. However, one factor that may inhibit a translation from cohesive parametric type theory to pure cubical type theory is the pointwise type $\mathsf{Glo}(\mathsf{U})$. Naively, one might attempt to translate this to a type of *affine cubical types*, *i.e.*, families of types indexed by bridge interval context, but it is unclear if the type of such structures is internally definable. The similar question of whether *(semi-)simplicial types* can be defined in homotopy type theory is a long-standing open problem.

Recent work of Sterling and Harper [SH20] and Sterling and Angiuli [SA21] lays out a system of *synthetic Tait computability*, suggesting that the internalization of logical relations is a technique with broader applications. In these works, the syntactic category of a type theory is related by modalities to a *gluing model*, in which each type is equipped with some computability structure. Our addition of cohesion to relate parametric and pointwise theories brings an analogy between their work and internal parametricity closer within reach.

Bibliography

[ABCFHL19] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. "Syntax and Models of Cartesian Cubical Type Theory". Unpublished draft. Feb. 2019. URL: https://github.com/dlicata335/cart-cube.

[Abe08] Andreas Abel. "Polarised subtyping for sized types". In: *Math. Struct. Comput. Sci.* 18.5 (2008), pp. 797–822. DOI: 10.1017/S0960129508006853. URL: https://doi.org/10.1017/S0960129508006853.

[Abe13] Andreas Abel. "Normalization by Evaluation: Dependent Types and Impredicativity". Habilitation. Ludwig-Maximilians-Universität München, 2013.

[ABFJ20] Mathieu Anel, Georg Biedermann, Eric Finster, and André Joyal. "A generalized Blakers-Massey theorem". In: *Journal of Topology* 13.4 (2020), pp. 1521–1553. DOI: 10.1112/topo.12163.

[ACC93] Martn Abadi, Luca Cardelli, and Pierre-Louis Curien. "Formal Parametric Polymorphism". In: *Theor. Comput. Sci.* 121.1&2 (1993), pp. 9–58. DOI: 10.1016/0304-3975(93)90082-5.

[ACCL91] Martn Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. "Explicit Substitutions". In: *J. Funct. Program.* 1.4 (1991), pp. 375–416. DOI: 10.1017/S0956796800000186.

[ACDKN18] Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. "Quotient Inductive-Inductive Types". In: *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*. 2018, pp. 293–310. DOI: 10.1007/978-3-319-89366-2_16.

[ACFHS18] Carlo Angiuli, Evan Cavallo, Kuen-Bang Hou (Favonia), Robert Harper, and Jonathan Sterling. "The RedPRL Proof Assistant (Invited Paper)". In: *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2018, Oxford, UK, 7th July 2018*. Ed. by Frédéric Blanqui and Giselle Reis. Vol. 274. EPTCS. 2018, pp. 1–10. DOI: 10.4204/EPTCS.274.1.

[ACMZ21] Carlo Angiuli, Evan Cavallo, Anders Mörtberg, and Max Zeuner. "Internalizing representation independence with univalence". In: *Proc. ACM Program. Lang.* 5.POPL (2021), pp. 1–30. DOI: 10.1145/3434293.

[ADK17] Thorsten Altenkirch, Nils Anders Danielsson, and Nicolai Kraus. "Partiality, Revisited - The Partiality Monad as a Quotient Inductive-Inductive Type". In: *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. Ed. by Javier Esparza and Andrzej S. Murawski. Vol. 10203. Lecture Notes in Computer Science. 2017, pp. 534–549. DOI: 10.1007/978-3-662-54458-7_31.

[AFH18] Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. "Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities". In: *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*. 2018, 6:1–6:17. DOI: 10.4230/LIPIcs.CSL.2018.6.

[Agda] The Agda Development Team. *The Agda Programming Language*. 2020. URL: http://wiki.portal.chalmers.se/agda/pmwiki.php.

[AGJ14] Robert Atkey, Neil Ghani, and Patricia Johann. "A relationally parametric model of dependent type theory". In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. 2014, pp. 503–516. DOI: 10.1145/2535838.2535852.

[AGS12] Steve Awodey, Nicola Gambino, and Kristina Sojakova. "Inductive Types in Homotopy Type Theory". In: *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. 2012, pp. 95–104.

[AHMP98] Arnon Avron, Furio Honsell, Marino Miculan, and Cristian Paravano. "Encoding Modal Logics in Logical Frameworks". In: *Stud Logica* 60.1 (1998), pp. 161–208. DOI: 10.1023/A:1005060022386. URL: https://doi.org/10.1023/A:1005060022386.

[AK16] Thorsten Altenkirch and Ambrus Kaposi. "Type theory in type theory using quotient inductive types". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. 2016, pp. 18–29. DOI: 10.1145/2837614.2837638.

[All87] Stuart Allen. "A Non-Type-Theoretic Definition of Martin-Löf's Types". In: *Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987*. 1987, pp. 215–221.

[AMS07] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. "Observational equality, now!" In: *Proceedings of the ACM Workshop Programming Languages meets Program Verification, PLPV 2007, Freiburg, Germany, October 5, 2007*. 2007, pp. 57–68. DOI: 10.1145/1292597.1292608.

[Ang19] Carlo Angiuli. "Computational Semantics of Cartesian Cubical Type Theory". PhD thesis. Carnegie Mellon University, 2019. URL: http://reports-archive.adm.cs.cmu.edu/anon/2019/abstracts/19-127.html.

[AW09] Steve Awodey and Michael A. Warren. "Homotopy theoretic models of identity types". In: *Math. Proc. Cambridge Philos. Soc.* 146.1 (2009), pp. 45–55. DOI: 10.1017/S0305004108001783.

[Awo18] Steve Awodey. "A cubical model of homotopy type theory". In: *Ann. Pure Appl. Logic* 169.12 (2018), pp. 1270–1294. DOI: 10.1016/j.apal.2018.08.002.

[Bar91] Henk Barendregt. "Introduction to generalized type systems". In: *Journal of Functional Programming* 1.2 (1991), pp. 122–154. DOI: 10.1017/S0956796800020025.

[BCH13] Marc Bezem, Thierry Coquand, and Simon Huber. "A Model of Type Theory in Cubical Sets". In: *19th International Conference on Types for Proofs and Programs, TYPES 2013, April 22-26, 2013, Toulouse, France*. 2013, pp. 107–128. DOI: 10.4230/LIPIcs.TYPES.2013.107.

[BCH19] Marc Bezem, Thierry Coquand, and Simon Huber. "The Univalence Axiom in Cubical Sets". In: *J. Autom. Reasoning* 63.2 (2019), pp. 159–171. DOI: 10.1007/s10817-018-9472-6.

[BCM15] Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. "A Presheaf Model of Parametric Type Theory". In: *Electr. Notes Theor. Comput. Sci.* 319 (2015), pp. 67–82. DOI: 10.1016/j.entcs.2015.12.006.

[BCMEPS20] 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.

[BCP15] Marc Bezem, Thierry Coquand, and Erik Parmann. "Non-Constructivity in Kan Simplicial Sets". In: *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*. Ed. by Thorsten Altenkirch. Vol. 38. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 92–106. DOI: 10.4230/LIPIcs.TLCA.2015.92.

[BELS16] Auke Bart Booij, Martn Hötzel Escardó, Peter LeFanu Lumsdaine, and Michael Shulman. "Parametricity, Automorphisms of the Universe, and Excluded Middle". In: *22nd International Conference on Types for Proofs and Programs, TYPES 2016, May 23-26, 2016, Novi Sad, Serbia*. 2016, 7:1–7:14. DOI: 10.4230/LIPIcs.TYPES.2016.7.

[Ben19] Bruno Bentzen. *Naive cubical type theory*. 2019. arXiv: 1911.05844 [cs.LO]. URL: http://arxiv.org/abs/1911.05844.

[BG11] Benno van den Berg and Richard Garner. "Types are weak ω-groupoids". In: *Proceedings of the London Mathematical Society* 102.2 (2011), pp. 370–394. DOI: 10.1112/plms/pdq026.

[BG12] Benno van den Berg and Richard Garner. "Topological and Simplicial Models of Identity Types". In: *ACM Trans. Comput. Log.* 13.1 (2012), 3:1–3:44. DOI: 10.1145/2071368.2071371.

[BGM17] Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. "The clocks are ticking: No more delays!" In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 2017, pp. 1–12. DOI: 10.1109/LICS.2017.8005097. URL: https://doi.org/10.1109/LICS.2017.8005097.

[BGW17] Henning Basold, Herman Geuvers, and Niels van der Weide. "Higher Inductive Types in Programming". In: *J. UCS* 23.1 (2017), pp. 63–88.

[BHN13] Nick Benton, Martin Hofmann, and Vivek Nigam. "Proof-Relevant Logical Relations for Name Generation". In: *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*. Ed. by Masahito Hasegawa. Vol. 7941. Lecture Notes in Computer Science. Springer, 2013, pp. 48–60. DOI: 10.1007/978-3-642-38946-7_6.

[BHN14] Nick Benton, Martin Hofmann, and Vivek Nigam. "Abstract effects and proof-relevant logical relations". In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*. 2014, pp. 619–632. DOI: 10.1145/2535838.2535869.

[BJP10] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. "Parametricity and dependent types". In: *ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*. 2010, pp. 345–356.

[BL11] Jean-Philippe Bernardy and Marc Lasson. "Realizability and Parametricity in Pure Type Systems". In: *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. Ed. by Martin Hofmann. Vol. 6604. Lecture Notes in Computer Science. Springer, 2011, pp. 108–122. DOI: 10.1007/978-3-642-19805-2_8.

[BM12] Jean-Philippe Bernardy and Guilhem Moulin. "A Computational Interpretation of Parametricity". In: *LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. 2012, pp. 135–144. DOI: 10.1109/LICS.2012.25.

[BM13] Jean-Philippe Bernardy and Guilhem Moulin. "Type-theory in color". In: *ICFP 2013, Boston, MA, USA - September 25 - 27, 2013*. 2013, pp. 61–72. DOI: 10.1145/2500365.2500577.

[Bru16] Guillaume Brunerie. "On the homotopy groups of spheres in homotopy type theory". PhD thesis. Université de Nice Sophia Antipolis, 2016.

[Bru18] Guillaume Brunerie. "Computer-generated proofs for the monoidal structure of the smash product". *Homotopy Type Theory Electronic Seminar Talks*. Nov. 2018. URL: https://www.uwo.ca/math/faculty/kapulkin/seminars/hottest.html.

[BS91] Ulrich Berger and Helmut Schwichtenberg. "An Inverse of the Evaluation Functional for Typed lambda-calculus". In: *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*. 1991, pp. 203–211. DOI: 10.1109/LICS.1991.151645.

[Car86] John Cartmell. "Generalised algebraic theories and contextual categories". In: *Annals of Pure and Applied Logic* 32 (1986), pp. 209–243. ISSN: 0168-0072. DOI: 10.1016/0168-0072(86)90053-9.

[Cav15] Evan Cavallo. "Synthetic cohomology in homotopy type theory". MA thesis. Carnegie Mellon University, 2015.

[Cav19] Evan Cavallo. *Stable factorization from a fibred algebraic weak factorization system*. 2019. arXiv: 1910.03121 [math.CT].

[CCHM15] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom". In: *21st International Conference on Types for Proofs and Programs, TYPES 2015, May 18-21, 2015, Tallinn, Estonia*. 2015, 5:1–5:34. DOI: 10.4230/LIPIcs.TYPES.2015.5.

[CH19a] Evan Cavallo and Robert Harper. "Higher inductive types in cubical computational type theory". In: *PACMPL* 3.POPL (2019), 1:1–1:27. DOI: 10.1145/3290314.

[CH19b] Evan Cavallo and Robert Harper. *Parametric Cubical Type Theory*. 2019. arXiv: 1901.00489 [cs.LO].

[CH20] Evan Cavallo and Robert Harper. "Internal Parametricity for Cubical Type Theory". In: *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*. 2020, 13:1–13:17. DOI: 10.4230/LIPIcs.CSL.2020.13.

[Che12] James Cheney. "A dependent nominal type theory". In: *Logical Methods in Computer Science* 8.1 (2012). DOI: 10.2168/LMCS-8(1:8)2012.

[CHM18] Thierry Coquand, Simon Huber, and Anders Mörtberg. "On Higher Inductive Types in Cubical Type Theory". In: *LICS 2018, Oxford, UK, July 9-12, 2018*. 2018. DOI: 10.1145/3209108.3209197.

[CHS19] Thierry Coquand, Simon Huber, and Christian Sattler. "Homotopy Canonicity for Cubical Type Theory". In: *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*. 2019, 11:1–11:23. DOI: 10.4230/LIPIcs.FSCD.2019.11.

[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.

[CMS20] Evan Cavallo, Anders Mörtberg, and Andrew W. Swan. "Unifying Cubical Models of Univalent Type Theory". In: *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*. 2020, 14:1–14:17. DOI: 10.4230/LIPIcs.CSL.2020.14.

[Con+86] Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, Todd B. Knoblock, N. P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing mathematics with the Nuprl proof development system*. Prentice Hall, 1986.

[Con09] R. L. Constable. "Computational type theory". In: *Scholarpedia* 4.2 (2009). revision #130876, p. 7618. DOI: 10.4249/scholarpedia.7618.

[Coq] The Coq Development Team. *The Coq Proof Assistant*. URL: https://www.coq.inria.fr.

[Coq19] Thierry Coquand. "Canonicity and normalization for dependent type theory". In: *Theor. Comput. Sci.* 777 (2019), pp. 184–191. DOI: 10.1016/j.tcs.2019.01.015.

[Coq93] Thierry Coquand. "Infinite Objects in Type Theory". In: *Types for Proofs and Programs, International Workshop TYPES'93, Nijmegen, The Netherlands, May 24-28, 1993, Selected Papers*. Ed. by Henk Barendregt and Tobias Nipkow. Vol. 806. Lecture Notes in Computer Science. Springer, 1993, pp. 62–78. DOI: 10.1007/3-540-58085-9_72. URL: https://doi.org/10.1007/3-540-58085-9%5C_72.

[CP02] Iliano Cervesato and Frank Pfenning. "A Linear Logical Framework". In: *Inf. Comput.* 179.1 (2002), pp. 19–75. DOI: 10.1006/inco.2001.2951. URL: https://doi.org/10.1006/inco.2001.2951.

[CP88] Thierry Coquand and Christine Paulin. "Inductively defined types". In: *COLOG-88, International Conference on Computer Logic, Tallinn, USSR, December 1988, Proceedings*. 1988, pp. 50–66.

[CubAg] The Agda Development Team. *Cubical Agda Library*. 2020. URL: https://github.com/agda/cubical.

[Dij17] Gabe Dijkstra. "Quotient inductive-inductive definitions". PhD thesis. University of Nottingham, UK, 2017. URL: http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.728471.

[DM17] Peter Dybjer and Hugo Moeneclaey. "Finitary Higher Inductive Types in the Groupoid Model". In: *Mathematical Foundations of Programming Semantics, 33rd International Conference, Ljubljana, Slovenia*. 2017.

[Doo16] Floris van Doorn. "Constructing the propositional truncation using non-recursive HITs". In: *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016*. 2016, pp. 122–129. DOI: 10.1145/2854065.2854076.

[Doo18] Floris van Doorn. "On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory". PhD thesis. Carnegie Mellon University, 2018.

[DP02] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order, Second Edition*. Cambridge University Press, 2002. ISBN: 978-0-521-78451-1. DOI: 10.1017/CBO9780511809088.

[Dyb00] Peter Dybjer. "A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory". In: *J. Symb. Log.* 65.2 (2000), pp. 525–549. DOI: 10.2307/2586554.

[Dyb94] Peter Dybjer. "Inductive Families". In: *Formal Aspects of Computing* 6.4 (1994), pp. 440–465.

[EK66] Samuel Eilenberg and G. Max Kelly. "Closed categories". In: *Proceedings of the Conference on Categorical Algebra* (1966), pp. 421–562. DOI: 10.1007/978-3-642-99902-4_22.

[FFLL16] Kuen-Bang Hou (Favonia), Eric Finster, Daniel R. Licata, and Peter LeFanu Lumsdaine. "A Mechanization of the Blakers-Massey Connectivity Theorem in Homotopy Type Theory". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*. 2016, pp. 565–574. DOI: 10.1145/2933575.2934545.

[FPS20] Marcelo P. Fiore, Andrew M. Pitts, and S. C. Steenkamp. "Constructing Infinitary Quotient-Inductive Types". In: *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*. 2020, pp. 257–276. DOI: 10.1007/978-3-030-45231-5_14.

[FXG20] Fredrik Nordvall Forsberg, Chuangjie Xu, and Neil Ghani. "Three equivalent ordinal notation systems in cubical Agda". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*. Ed. by Jasmin Blanchette and Catalin Hritcu. ACM, 2020, pp. 172–185. DOI: 10.1145/3372885.3373835.

[GCKGB21] Daniel Gratzer, Evan Cavallo, G.A. Kavvos, Adrien Guatto, and Lars Birkedal. "Modalities and Parametric Adjoints". In submission. 2021.

[GG08] Nicola Gambino and Richard Garner. "The identity type weak factorisation system". In: *Theor. Comput. Sci.* 409.1 (2008), pp. 94–109. DOI: 10.1016/j.tcs.2008.08.030.

[Gir72] Jean-Yves Girard. "Interprétation fonctionnelle et élimination des coupures de l'arithmetique d'ordre supérieur". PhD thesis. Université Paris VII, 1972.

[GKNB20] 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.

[GLNPRSW17] 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.

[GM03] Marco Grandis and Luca Mauri. "Cubical sets and their site". In: *Theory and Applications of Categories* 11.8 (2003), pp. 185–211. URL: http://www.tac.mta.ca/tac/volumes/11/8/11-08abs.html.

[GNO16] Neil Ghani, Fredrik Nordvall Forsberg, and Federico Orsanigo. "Proof-Relevant Parametricity". In: *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*. Ed. by Sam Lindley, Conor McBride, Philip W. Trinder, and Donald Sannella. Vol. 9600. Lecture Notes in Computer Science. Springer, 2016, pp. 109–131. DOI: 10.1007/978-3-319-30936-1_6.

[GSB19] Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. "Implementing a modal dependent type theory". In: *Proc. ACM Program. Lang.* 3.ICFP (2019), 107:1–107:29. DOI: 10.1145/3341711.

[Har92] Robert Harper. "Constructing Type Systems over an Operational Semantics". In: *J. Symb. Comput.* 14.1 (1992), pp. 71–84. DOI: 10.1016/0747-7171(92)90026-Z.

[Has94] Ryu Hasegawa. "Categorical Data Types in Parametric Polymorphism". In: *Math. Struct. Comput. Sci.* 4.1 (1994), pp. 71–109. DOI: 10.1017/S0960129500000372.

[Hof95] Martin Hofmann. "Extensional concepts in intensional type theory". PhD thesis. Edinburgh: University of Edinburgh, Jan. 1995.

[HS98] Martin Hofmann and Thomas Streicher. "The groupoid interpretation of type theory". In: *Twenty-five years of constructive type theory (Venice, 1995)*. Vol. 36. Oxford Logic Guides. New York: Oxford Univ. Press, 1998, pp. 83–111.

[Hub19] Simon Huber. "Canonicity for Cubical Type Theory". In: *J. Autom. Reason.* 63.2 (2019), pp. 173–210. DOI: 10.1007/s10817-018-9469-1.

[Hug19] Jasper Hugunin. "Constructing Inductive-Inductive Types in Cubical Type Theory". In: *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*. Ed. by Mikolaj Bojanczyk and Alex Simpson. Vol. 11425. Lecture Notes in Computer Science. Springer, 2019, pp. 295–312. DOI: 10.1007/978-3-030-17127-8_17.

[JS17] Patricia Johann and Kristina Sojakova. "Cubical Categories for Higher-Dimensional Parametricity". arXiv:1701.06244. Jan. 2017.

[JY20] Niles Johnson and Donald Yau. *2-Dimensional Categories*. 2020. arXiv: 2002.06055 [math.CT].

[Kan55] Daniel M. Kan. "Abstract Homotopy. I". In: *Proceedings of the National Academy of Sciences of the United States of America* 41.12 (1955), pp. 1092–1096. ISSN: 00278424.

[Kap17] Ambrus Kaposi. "Type theory in a type theory with quotient inductive types". PhD thesis. University of Nottingham, UK, 2017. URL: http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.713896.

[Kav16] G.A. Kavvos. *The Many Worlds of Modal λ-calculi: I. Curry-Howard for Necessity, Possibility and Time*. 2016. arXiv: 1605.08106 [cs.LO].

[Kav19] G. A. Kavvos. "Modalities, cohesion, and information flow". In: *Proc. ACM Program. Lang.* 3.POPL (2019), 20:1–20:29. DOI: 10.1145/3290333. URL: https://doi.org/10.1145/3290333.

[Kav20] G. A. Kavvos. "Dual-Context Calculi for Modal Logic". In: *Log. Methods Comput. Sci.* 16.3 (2020). URL: https://lmcs.episciences.org/6722.

[KD13] Neelakantan R. Krishnaswami and Derek Dreyer. "Internalizing Relational Parametricity in the Extensional Calculus of Constructions". In: *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*. 2013, pp. 432–451.

[KHS19] Ambrus Kaposi, Simon Huber, and Christian Sattler. "Gluing for Type Theory". In: *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*. 2019, 25:1–25:19. DOI: 10.4230/LIPIcs.FSCD.2019.25.

[KK18] Ambrus Kaposi and András Kovács. "A syntax for higher inductive-inductive types". In: *3nd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*. 2018.

[KK20a] Ambrus Kaposi and András Kovács. "Signatures and Induction Principles for Higher Inductive-Inductive Types". In: *Log. Methods Comput. Sci.* 16.1 (2020). DOI: 10.23638/LMCS-16(1:10)2020.

[KK20b] András Kovács and Ambrus Kaposi. "Large and Infinitary Quotient Inductive-Inductive Types". In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller. ACM, 2020, pp. 648–661. DOI: 10.1145/3373718.3394770.

[KL12a] Chris Kapulkin and Peter LeFanu Lumsdaine. *The Simplicial Model of Univalent Foundations (after Voevodsky)*. 2012. arXiv: 1211.2851 [math.LO].

[KL12b] Chantal Keller and Marc Lasson. "Parametricity in an Impredicative Sort". In: *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*. 2012, pp. 381–395.

[KL20] Chris Kapulkin and Peter LeFanu Lumsdaine. "The Law of Excluded Middle in the Simplicial Model of Type Theory". Unpublished note. 2020. URL: https://www.uwo.ca/math/faculty/kapulkin/papers/LEM_in_sSet.pdf.

[KPB15] Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton. "Integrating Linear and Dependent Types". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. Ed. by Sriram K. Rajamani and David Walker. ACM, 2015, pp. 17–30. DOI: 10.1145/2676726.2676969. URL: https://doi.org/10.1145/2676726.2676969.

[Kra16] Nicolai Kraus. "Constructions with Non-Recursive Higher Inductive Types". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*. 2016, pp. 595–604. DOI: 10.1145/2933575.2933586.

[Law07] F. William Lawvere. "Axiomatic Cohesion". In: *Theory and Applications of Categories* 19.3 (2007), pp. 31–49. URL: http://www.tac.mta.ca/tac/volumes/19/3/19-03abs.html.

[LB15] Daniel R. Licata and Guillaume Brunerie. "A Cubical Approach to Synthetic Homotopy Theory". In: *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. 2015, pp. 92–103. DOI: 10.1109/LICS.2015.19.

[LH11] Daniel R. Licata and Robert Harper. "2-Dimensional Directed Type Theory". In: *Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011*. 2011, pp. 263–289. DOI: 10.1016/j.entcs.2011.09.026.

[LH12] Daniel R. Licata and Robert Harper. "Canonicity for 2-dimensional type theory". In: *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*. 2012, pp. 337–348. DOI: 10.1145/2103656.2103697.

[LL32] C.I. Lewis and C.H. Langford. *Symbolic Logic*. New York: The Century Co, 1932.

[LOPS18] 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, July 9-12, 2018, Oxford, UK*. Ed. by Hélène Kirchner. Vol. 108. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 22:1–22:17. DOI: 10.4230/LIPIcs.FSCD.2018.22.

[LS13] Daniel R. Licata and Michael Shulman. "Calculating the Fundamental Group of the Circle in Homotopy Type Theory". In: *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. 2013, pp. 223–232. DOI: 10.1109/LICS.2013.28.

[LS16] Daniel R. Licata and Michael Shulman. "Adjoint Logic with a 2-Category of Modes". In: *Logical Foundations of Computer Science - International Symposium, LFCS 2016, Deerfield Beach, FL, USA, January 4-7, 2016. Proceedings*. Ed. by Sergei N. Artëmov and Anil Nerode. Vol. 9537. Lecture Notes in Computer Science. Springer, 2016, pp. 219–235. DOI: 10.1007/978-3-319-27683-0_16.

[LS20] Peter LeFanu Lumsdaine and Michael Shulman. "Semantics of higher inductive types". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 169.1 (2020), pp. 159–208. DOI: 10.1017/S030500411900015X.

[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, September 3-9, 2017, Oxford, UK*. Ed. by Dale Miller. Vol. 84. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 25:1–25:22. DOI: 10.4230/LIPIcs.FSCD.2017.25. URL: https://doi.org/10.4230/LIPIcs.FSCD.2017.25.

[Mac98] Saunders Mac Lane. *Categories for the Working Mathematician*. 0072-5285. Graduate Texts in Mathematics. New York: Springer, 1998.

[Mai91] Harry G. Mairson. "Outline of a Proof Theory of Parametricity". In: *Functional Programming Languages and Computer Architecture, 5th ACM Conference, Cambridge, MA, USA, August 26-30, 1991, Proceedings*. Ed. by John Hughes. Vol. 523. Lecture Notes in Computer Science. Springer, 1991, pp. 313–327. DOI: 10.1007/3540543961_15.

[Mai98] Maria Emilia Maietti. "About Effective Quotients in Constructive Type Theory". In: *Types for Proofs and Programs, International Workshop TYPES '98, Kloster Irsee, Germany, March 27-31, 1998, Selected Papers*. Ed. by Thorsten Altenkirch, Wolfgang Naraschewski, and Bernhard Reus. Vol. 1657. Lecture Notes in Computer Science. Springer, 1998, pp. 164–178. DOI: 10.1007/3-540-48167-2_12.

[Mar75] Per Martin-Löf. "An intuitionistic theory of types: predicative part". In: *Logic Colloquium '73*. Ed. by H.E. Rose and J.C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. North-Holland, 1975, pp. 73–118. DOI: 10.1016/S0049-237X(08)71945-1.

[Mar82] Per Martin-Löf. "Constructive Mathematics and Computer Programming". In: *Logic, Methodology and Philosophy of Science*. Ed. by L.J. Cohen, J. o, H. Pfeiffer, and K.-P. Podewski. Vol. VI. 1982, pp. 153–175.

[Mou16] Guilhem Moulin. "Internalizing Parametricity". PhD thesis. Chalmers University of Technology, Gothenburg, Sweden, 2016. URL: https://research.chalmers.se/en/publication/235758.

[MP20] Anders Mörtberg and Loc Pujet. "Cubical Synthetic Homotopy Theory". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 158–171. ISBN: 9781450370974. DOI: 10.1145/3372885.3373825. URL: https://doi.org/10.1145/3372885.3373825.

[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 2018, Oxford, UK, July 09-12, 2018*. 2018, pp. 779–788. DOI: 10.1145/3209108.3209119.

[NS10] Fredrik Nordvall Forsberg and Anton Setzer. "Inductive-Inductive Definitions". In: *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*. Ed. by Anuj Dawar and Helmut Veith. Vol. 6247. Lecture Notes in Computer Science. Springer, 2010, pp. 454–468. DOI: 10.1007/978-3-642-15205-4_35.

[Nuy20] Andreas Nuyts. "Contributions to Multimode and Presheaf Type Theory". PhD thesis. KU Leuven, Leuven, Belgium, 2020. URL: https://lirias.kuleuven.be/3065223.

[NVD17] Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. "Parametric quantifiers for dependent type theory". In: *PACMPL* 1.ICFP (2017), 32:1–32:29. DOI: 10.1145/3110276.

[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.

[PA93] Gordon D. Plotkin and Martn Abadi. "A Logic for Parametric Polymorphism". In: *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*. 1993, pp. 361–375.

[PD01] Frank Pfenning and Rowan Davies. "A judgmental reconstruction of modal logic". In: *Math. Struct. Comput. Sci.* 11.4 (2001), pp. 511–540. DOI: 10.1017/S0960129501003322.

[Pfe01] Frank Pfenning. "Intensionality, Extensionality, and Proof Irrelevance in Modal Type Theory". In: *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*. IEEE Computer Society, 2001, pp. 221–230. DOI: 10.1109/LICS.2001.932499. URL: https://doi.org/10.1109/LICS.2001.932499.

[PGM04] Valeria de Paiva, Rajeev Goré, and Michael Mendler. "Modalities in Constructive Logics and Type Theories". In: *Journal of Logic and Computation* 14.4 (2004), pp. 439–446.

[Plo04] Gordon D. Plotkin. "A structural approach to operational semantics". In: *J. Log. Algebraic Methods Program.* 60-61 (2004), pp. 17–139.

[PR15] 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, Natal, Brazil, August 31 - September 1, 2015*. Ed. by Mario R. F. Benevides and René Thiemann. Vol. 323. Electronic Notes in Theoretical Computer Science. Elsevier, 2015, pp. 143–161. DOI: 10.1016/j.entcs.2016.06.010. URL: https://doi.org/10.1016/j.entcs.2016.06.010.

[RedPRL] The RedPRL Development Team. *RedPRL – the People's Refinement Logic*. URL: http://www.redprl.org/.

[redtt] The RedPRL Development Team. `redtt`. URL: https://github.com/RedPRL/redtt.

[Ree09] Jason Reed. "A Judgmental Deconstruction of Modal Logic". Unpublished note. 2009. URL: www.cs.cmu.edu/~jcreed/papers/jdml.pdf.

[Rey83] John C. Reynolds. "Types, Abstraction and Parametric Polymorphism". In: *IFIP Congress*. 1983, pp. 513–523.

[Rie14] Emily Riehl. *Categorical Homotopy Theory*. New Mathematical Monographs. Cambridge University Press, 2014. DOI: 10.1017/CBO9781107261457.

[Rie18] Emily Riehl. "On the directed univalence axiom". Talk slides, AMS Special Session on Homotopy Type Theory, Joint Mathematics Meetings. Jan. 2018. URL: http://www.math.jhu.edu/~eriehl/JMM2018-directed-univalence.pdf.

[Rij17] Egbert Rijke. *The join construction*. Jan. 2017. arXiv: 1701.07538 [math.CT].

[Rij18] Egbert Rijke. "Classifying Types: Topics in synthetic homotopy theory". PhD thesis. Carnegie Mellon University, 2018. URL: https://arxiv.org/abs/1906.09435.

[RS17] Emily Riehl and Michael Shulman. "A type theory for synthetic ∞-categories". In: *Higher Structures* 1.1 (2017), pp. 116–193. URL: https://journals.mq.edu.au/index.php/higher_structures/article/view/36.

[RSS20] Egbert Rijke, Michael Shulman, and Bas Spitters. "Modalities in homotopy type theory". In: *Log. Methods Comput. Sci.* 16.1 (2020). DOI: 10.23638/LMCS-16(1:2)2020. URL: https://doi.org/10.23638/LMCS-16(1:2)2020.

[SA21] Jonathan Sterling and Carlo Angiuli. *Normalization for Cubical Type Theory*. 2021. arXiv: 2101.11479 [cs.LO].

[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, June 24-30, 2019, Dortmund, Germany*. 2019, 31:1–31:25. DOI: 10.4230/LIPIcs.FSCD.2019.31.

[Sat18] Christian Sattler. *Do cubical models of type theory also model homotopy types?* Talk at the Types, Homotopy Type theory, and Verification Workshop at the Hausdorff Research Institute for Mathematics. June 2018. URL: https://www.youtube.com/watch?v=wkPDyIGmEoA.

[Sch13] Urs Schreiber. *Differential cohomology in a cohesive infinity-topos*. 2013. arXiv: 1310.7930 [math-ph].

[Sco72] Dana Scott. "Continuous lattices". In: *Toposes, Algebraic Geometry and Logic*. Ed. by F. W. Lawvere. Berlin, Heidelberg: Springer, 1972, pp. 97–136. ISBN: 978-3-540-37609-5.

[SH20] Jonathan Sterling and Robert Harper. *Logical Relations as Types: Proof-Relevant Parametricity for Program Modules*. 2020. arXiv: 2010.08599 [cs.LO].

[Shu15] Michael Shulman. "Univalence for inverse diagrams and homotopy canonicity". In: *Math. Struct. Comput. Sci.* 25.5 (2015), pp. 1203–1277. DOI: 10.1017/S0960129514000565.

[Shu18] Michael Shulman. "Brouwer's fixed-point theorem in real-cohesive homotopy type theory". In: *Math. Struct. Comput. Sci.* 28.6 (2018), pp. 856–941. DOI: 10.1017/S0960129517000147.

[Sim94] Alex K. Simpson. "The proof theory and semantics of intuitionistic modal logic". PhD thesis. University of Edinburgh, UK, 1994. URL: http://hdl.handle.net/1842/407.

[SJ18] Kristina Sojakova and Patricia Johann. "A General Framework for Relational Parametricity". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. 2018, pp. 869–878. DOI: 10.1145/3209108.3209141.

[Soj14] Kristina Sojakova. "Higher Inductive Types as Homotopy-Initial Algebras". arXiv:1402.0761. Feb. 2014.

[Soj15] Kristina Sojakova. "Higher Inductive Types as Homotopy-Initial Algebras". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. 2015, pp. 31–42.

[Soj16] Kristina Sojakova. "Higher Inductive Types as Homotopy-Initial Algebras". PhD thesis. Carnegie Mellon University, 2016.

[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.

[Str67] Christopher Strachey. *Fundamental Concepts in Programming Languages*. Copenhagen: Lecture notes, International Summer School in Computer Programming, 1967.

[Swa18a] Andrew W Swan. *Identity Types in Algebraic Model Structures and Cubical Sets*. 2018. arXiv: 1808.00915 [math.CT].

[Swa18b] Andrew W Swan. *Separating Path and Identity Types in Presheaf Models of Univalent Type Theory*. 2018. arXiv: 1808.00920 [math.LO].

[Tar55] Alfred Tarski. "A lattice-theoretical fixpoint theorem and its applications." In: *Pacific Journal of Mathematics* 5.2 (1955), pp. 285–309. DOI: pjm/1103044538. URL: https://doi.org/.

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

[VAG+20] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. *UniMath — a computer-checked library of univalent mathematics*. 2020. URL: https://github.com/UniMath/UniMath.

[Vák14] Matthijs Vákár. *Syntax and Semantics of Linear Dependent Types*. 2014. arXiv: 1405.0033 [cs.LO].

[VMA19] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. "Cubical Agda: a dependently typed programming language with univalence and higher inductive types". In: *Proc. ACM Program. Lang.* 3.ICFP (2019), 87:1–87:29. DOI: 10.1145/3341691.

[Voe14] Vladimir Voevodsky. *The equivalence axiom and univalent models of type theory*. Talk at CMU on February 4, 2010. 2014. arXiv: 1402.5556 [math.LO].

[Voe15] Vladimir Voevodsky. "An experimental library of formalized Mathematics based on the univalent foundations". In: *Math. Struct. Comput. Sci.* 25.5 (2015), pp. 1278–1294. DOI: 10.1017/S0960129514000577. URL: https://doi.org/10.1017/S0960129514000577.

[Wad07] Philip Wadler. "The Girard-Reynolds isomorphism (second edition)". In: *Theor. Comput. Sci.* 375.1-3 (2007), pp. 201–226. DOI: 10.1016/j.tcs.2006.12.042. URL: https://doi.org/10.1016/j.tcs.2006.12.042.

[Wad89] Philip Wadler. "Theorems for Free!" In: *FPCA 1989, London, UK, September 11-13, 1989*. 1989, pp. 347–359. DOI: 10.1145/99370.99404.

[Wad90] Philip Wadler. "Recursive types in polymorphic second-order lambda-calculus". Draft, University of Glasgow. 1990.

[War08] Michael Alton Warren. "Homotopy Theoretic Aspects of Constructive Type Theory". PhD thesis. Carnegie Mellon University, 2008. URL: http://mawarren.net/papers/phd.pdf.

[Wel18] Felix Wellen. *Cartan Geometry in Modal Homotopy Type Theory*. 2018. arXiv: 1806.05966 [math.DG].

[WL20] Matthew Z. Weaver and Daniel R. Licata. "A Constructive Model of Directed Univalence in Bicubical Sets". In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller. ACM, 2020, pp. 915–928. DOI: 10.1145/3373718.3394794.

[Zwa19] Colin Zwanziger. "Natural Model Semantics for Comonadic and Adjoint Type Theory: Extended Abstract". In: *Preproceedings of Applied Category Theory Conference 2019*. 2019.