Cubical modal type theories

Magnus Baunsgaard Kristensen

PhD dissertation submitted to
Department of Computer Science
IT University of Copenhagen
Denmark
2022

ABSTRACT

The goal of this thesis is to combine Cubical Type Theory (CTT) and modal type theory towards obtaining a type theory suitable for the verification of programs relying on features such as recursion. For this purpose the main advantage of CTT is that extensionality principles such as function extensionality can be shown as theorems, and as such can be given computational content. Additionally CTT supports the powerful univalence axiom as well as the highly useful Higher Inductive Types (HITs). Another extension to type theory is that of modalities which can be used to encode the primitives needed for programming with and reasoning about recursion, staged computation, and information flow control. Combining these two approaches offers the promise of a type theory with the necessary amount of structure to encode e.g., reactive programs with productivity guarantees and programs relying on control of information flow between types with non-interference guarantees.

The thesis contains two papers. The first paper presents a type theory with a family of Fitch-style later modalities indexed over an object of clocks, called Clocked Cubical Type Theory (CCTT). Guarded recursion with multiple clocks has been used in earlier work to encode coinductive types. The primary novelty in this theory is induction under clocks for HITs. The encoding of coinductive types requires functors which commute with clock quantification and a large class of these can be produced by applying induction under clocks. One example of interest is the finite powerset functor, which allows for the encoding of nondeterministic processes. The theory is shown to be sound by the construction of a presheaf model.

In the second paper a more general framework called Cubical Modal Type Theory ($\text{MTT}_{\square}$) is presented. This extends the existing framework of Multimodal Type Theory (MTT) with the primitives of CTT. MTT is a modal type theory indexed by a 2-category of modes which is flexible enough to encode S4, cohesive type theory, and guarded recursion. The new $\text{MTT}_{\square}$ extends this theory with the computationally well understood path types. The primary contribution of the paper is a method for the quick production of models in the style of Orton and Pitts. One of the technical challenges in achieving this is the construction of composition structures on modal types, which requires the cubical structure to be coherent with respect to the indexing 2-category.

RESUMÉ

Målet med denne afhandling er at kombinere Cubical Type Theory (CTT) med modal type teori for at opnå en typeteori som egner sig til verifikationen af programmer der benytter redskaber som rekursion. I den henseende er hovedfordelen ved CTT at ekstensionalitetsprincipper som funktionsekstensionalitet kan bevises og dermed har gode beregningsmæssige egenskaber. Derudover understøtter CTT det kraftulde univalence princip samt det yderst værdifulde højere induktive typer (HITs). En anden udviddelse af type teori er tilføjelsen af modaliteter, som kan bruges til at indkode primitiver som er nødvendige for at programmere med og ræsonnere om rekursion, staged computation og kontrol af informationsstrøm. Kombinationen af disse paradigmer tilbyder løftet om en typeteori med den nødvendige struktur til at indkode for eksempel reaktive programmer med produktivitetsgarantier og programmer som benytter kontrol af informationsstrømme med ikke-inteferensgarantier.

Afhandlingen består af to artikler. Den første artikel præsenterer en typeteori med en familie af Fitch-agtige later modaliteter indekseret over et objekt af ure, som er navngivet Clocked Cubical Type Theory (CCTT). Sikret rekursion med flere ure er blevet brugt i tidligere arbejde til at indkode koinduktive typer. Den primære nyskabelse i denne teori er induktion under ure for HITs. Indkodningsresultatet for koinduktive typer har brug for funktorer som kommuterer med kvantifikation over objektet af ure, og man kan producere en stor klasse af disse ved brug af induktion under ure. Et specielt interessant eksempel på en sådan funktorer er den endelige potensmængde funktor, som muliggør indkodningen af ikke-deterministiske processer. Teorien bliver bevist konsistent ved konstruktionen af en presheaf model.

I den anden artikel præsenteres en mere generel struktur, nemlig Cubical Multimodal Typeteori ($\text{MTT}_{\square}$). Dette er en udviddelse af den eksisterende teori Multimodal Typeteori (MTT) med primitiverne fra CTT. MTT er en modal typeteori indekseret af en 2-kategori af tilstande, som er general nok til at dække over modal S4, kohæsiv typeteori, og sikret rekursion. Den nye $\text{MTT}_{\square}$ tilføjer de beregningsmæssigt velforståede stityper. Det primære bidrag i artiklen er en metode til hurtigt at producere modeller i stil med Orton-Pitts metoden. En af de teksniske udfordringer ved dette er konstruktionen af kompositionsstruktur på modal typer, som kræver at de strukturer som er importeret fra CTT spiller fornuftigt sammen med den indekserende 2-kategori.

ACKNOWLEDGEMENTS

I would like to first and foremost thank my supervisor, Rasmus Ejlers Møgelberg for his encouragement and support throughout my time as a PhD student. His active involvement in my research and help in navigating field of type theory have both been invaluable. PhD studies can be an isolating experience at the best of times, and during COVID lockdowns even more so. I imagine that supervision in such times is a difficult endeavor, and I will be forever grateful for the effort Rasmus invested in my time at ITU.

I would also like to thank my somewhat transient collection of colleagues over the years in the PLS group for the good company and stimulating discussions. In particular I would like to thank Andrea Vezzosi without whom many stones would have gone unturned and many tricks would have gone unlearned.

Lastly I would like to thank Lars Birkedal for hosting me at Aarhus University, and the Aarhus group for making the stay worthwhile even in times where such things are difficult. In particular I would like to thank Philipp Stassen for being a fantastic study- and officemate, and Frederik Lerbjerg Aagard, Daniel Gratzer and Lars for our collaboration.

CONTENTS

1. Introduction 1
1.1 Background 3
1.1.1 Formal verification 3
1.1.2 Reactive programming 4
1.1.3 Guarded recursion 6
1.1.4 Dependent right adjoints and multimodal type theory . . 7
1.1.5 Equality in type theory 9
1.1.6 Cubical type theory 10
1.1.7 Denotational semantics 12
1.2 Contributions 14
1.2.1 Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks 14
1.2.2 Unifying modal and multimodal type theory 16
1.3 Related work 19
1.4 Statement of contributions 22

2. Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks 29

3. Unifying cubical and multimodal type theory 61

1. INTRODUCTION

Type theory is a formalism that can be regarded both as an abstract programming language and a mathematical logic. This duality of type theory is called the *Curry-Howard correspondence*, and it allows one to leverage the power of type theory as a logic to reason about programs written in type theory as a programming language.

In programming language theory, type theory is thus deployed as a tool for providing guarantees about programs in two forms. Programs written in type theory come with certain safety guarantees depending on the language. For the simply typed lambda-calculus this might mean basic type safety of the programs, i.e., the property that a well typed program never gets stuck trying to apply a function to input it is not defined on. In this thesis we focus on dependent type theory à la Martin-Löf (MLTT) [37], which is able to confer positive guarantees. Via Curry-Howard, simply typed languages correspond to propositional logic and dependently typed languages correspond to predicate logic. With dependent types we can thus take advantage of predicate logic in programming to verify that a program adheres to a given specification[1]. Specifically, dependent types allows us to attach various proof obligations to our types, ensuring, e.g., that a function outputs not just any number, but a number with a specified property, or to prevent buffer overflow and null pointer exceptions as in [58]. This is where the full force of the Curry-Howard correspondence is leveraged, since the language we write the specification in and the programming language coincide.

In mathematics, type theory can be viewed as a logic. There is a push for the mathematics formalized in the traditional foundation of set theory to be recast in type theory, in part because type theory is amenable to computer checking of proofs via proof assistants. Concretely, various proof assistants such as Agda [43] and Coq [53] can be used to check the well typedness of terms. Since in type theory types are propositions and terms are proofs, proof assistants allow for the mechanical checking of proof correctness, which is often difficult to do by hand. Many different variants of type theory have models in set theoretic foundations, and often they can be used as the internal language for some class of categories. It is foundational to type theory that the simply typed lambda-calculus can be used as the internal language for Cartesian closed categories [33] and that MLTT can be used as an internal language for locally Cartesian closed categories [25]. Recent advances in homotopical type theories with synthetic descriptions of

[1] From now on we mean dependent type theory when we say type theory unless otherwise specified.

higher structure has continued this story in higher category theory [30, 50], with the long term goal of this program being a reasonable definition of elementary ∞-topoi and a specification of their internal languages as a form of univalent type theory [54].

This thesis is about two extensions to type theory, both of which have ramifications on either side of the correspondence. One extension is by type constructors called *modalities*, which are used to add extra features to type theory. For instance guarded recursion can be added via the "later" modality [42], denoted $\triangleright$, or time-warps [24]. The modality $\triangleright$ is very well-studied: one variant of it has been shown to be sufficient for encoding *coinductive types* [39] and another has been shown to support reasoning about guarded recursive types up to bisimilarity [40]. We note that guarded recursion lies at the heart of recursion in various implementations. For instance modal FRP [32], the variants of RaTT [9], and the implementation of Iris [29].

Other examples of extending type theory by modalities to add expressivity to it are the following: Staged computation can be represented via modal S4 [21], using a variant of the split-context type theory of [45]. A particularly bountiful collection of modalities are those based on Lawvere's axiomatic cohesion [34] which is an axiomatic approach to spaces based on a triple of adjoints. These adjoints can be represented in type theory with modalities with a syntax mirroring [45], and have been employed to connect results from synthetic homotopy theory to topology as formulated in type theory [49] as well as in the study of information flow control [31]. Crisp type theory is a minimal variant of the cohesive setup and has been used in the study of models of cubical type theory CTT [35]. It internalizes the global sections functor in the same way that cohesive type theory internalizes the cohesion triple of adjoints, with the goal of describing global operations.

The other extension is by cubical primitives originating from Cubical Type Theory (CTT) [18, 5]. This type theory was first described as part of an effort to model univalent type theory constructively and in a way that supplies computational content to the eponymous *univalence axiom*, and has since garnered much interest. Univalent type theory or homotopy type theory (HoTT) adds the univalence axiom as a way to obtain equalities between types, or, taking a slightly different view, internalizing the fact that isomorphic types cannot be told apart internally without extra axioms. In CTT equality is represented by functions out of a primitive interval corresponding loosely to paths in topological spaces. Partly due to this specific representation and partly due to results from HoTT it is a powerful type theory for reasoning about equality. We expand on this point and motivate it further in section 1.1.6. The long-term goal of studying cubical modal type theory is to align the practicalities of system design and the desirable theoretical properties offered by the mathematically oriented type theory tradition. Using modalities we add features needed for the desired expressivity of the language in a principled way. The extensionality principles necessary for efficient reasoning in type theory, e.g., function extensionality and univalence, are obtained as theorems in cubical type theory, and moreover we shall see in the second paper how one obtains an appropriate extensionality principle for

modal types.

In section 1.1 we provide context for the thesis in a mostly non-technical fashion. We discuss first the context of the work from the point of view of programming language theory and then as a mathematical logic. The contributions of each paper is discussed in section 1.2, with related work being discussed in section 1.3, and a statement of contribution for each of the papers is available in section 1.4. Chapter 2 contains the first paper as submitted to LICS '22 and chapter 3 contains the second paper as submitted to LMCS.

1.1 Background

1.1.1 Formal verification

To an ever increasing degree, software permeates every part of society. Most of our interaction with governance is either assisted by software or completely digital, our financial system exists mostly in the digital sphere, the factories producing our food and medication relies on various control systems, and so on. Failure of these systems results in everything from minor frustrations to catastrophic losses, so it is important that the software is reliable. Verification of software in general is the process of ensuring that the software behaves in the intended manner and can be done in a number ways, either through testing or some sort of static program analysis. Formal verification is the deployment of formal methods to static program analysis with the goal of mathematically proving that systems do not behave in unintended ways or that they implement a precisely stated specification, bringing mathematical certainty to verification of software. Type theory can be used as a tool for both of these. The exclusion of unintended behavior is essentially a part of the metatheory of type theory: statements like canonicity for instance ensures that programs with output type $\mathbb{N}$ eventually produce a numeral, meaning that it terminates. The strength of the guarantees we can give in this regard is inversely correlated to the expressivity of the language, as Pierce writes in the introduction of [46]:

> For example, research on typed lambda-calculi is usually concerned with systems in which every well-typed computation is guaranteed to terminate, whereas most programming languages sacrifice this property for the sake of features like recursive function definitions.

In functional programming languages, general recursion in some form is needed for the language to be Turing complete[2]. Adding it to the language is equivalent to adding a fix-point operator, given by a function $Y : (A \to A) \to A$. Added without any checks this operator allows for many pathological definitions, such as programs depending on later unfoldings of themselves. To introduce recursion to our type theory we must thus be careful to verify that the self-reference is well-founded, since if it is not, the result is a program that will not terminate,

[2] From now on we will mean general recursion when we say recursion, as opposed to structural or primitive recursion, unless otherwise specified.

and indeed will not even be productive [19]. In section 1.1.2 we will see a concrete argument for why the expressivity offered by recursion is necessary and we will discuss various different approaches to the problem in detail in the later sections.

The other mode of formal verification via type theory involves the Curry-Howard correspondence directly. We can use the logic of type theory to write specifications about our programs and pass proofs of needed properties around in the code[3]. For instance, one might imagine an implementation of arrays where trying to access an element of the array we have to provide the desired index and a proof that this index is smaller than the length of the array. Conversely we might imagine functions which output not just data, but also a proof that we have some desired property. This could be a function for finding roots of a input polynomial which outputs a list of real numbers as well as proofs that these are indeed roots as desired. In both of these examples we require the expressivity offered by dependent types since we are in each case manipulating pairs of numbers and proofs that these numbers satisfy certain properties. Composing programs written in this style allows for a static check that they implement a given specification, with the specification itself being written in type theory. Returning to recursion, we mentioned that the unrestricted fixpoint combinator is problematic in that it no longer excludes certain modes of unintended behavior. In addition to this, it makes the logic inconsistent. Spelling the point out, an unrestricted fixpoint operator would allow one to produce via the Curry-Howard correspondence a proof of the false proposition in type theory, collapsing the logic and thus the ability to write and check program specifications.

1.1.2 Reactive programming

Reactive programs are, as the name suggests, a class of programs with behavior that is to some extent determined by outside input, with the typical case being a two-way interaction with an *environment*. A simple example would be a piece of software controlling an air conditioner. An air conditioner must cool the air until the temperature falls below a certain point, a two-way interaction with the physical temperature in a room. This point must be adjustable via user input, an interaction with a user. While an air conditioner is rarely considered critical, many critical systems are inherently reactive. Like air conditioners, many such systems do not come with a predefined end state; they are designed to run for indeterminate amounts of time. To accommodate this in type theory we reason about these systems as manipulation of fundamentally infinite structures, such as an infinite stream of input temperatures as measured by the air conditioner. Reactive programs are an obvious target for verification efforts for two reasons. Firstly, an increasing number of critical systems such as financial systems, healthcare systems, and systems for the automatic operation of, e.g., airplanes, cars, cooling systems for nuclear reactors, etc., rely on reactive programs. In addition to this, errors in reactive programs might require very long

[3] often the concrete proofs are not actually passed around at runtime for efficiency reasons. Deleting proofs from the runtime program in this way is known as erasure.

runtimes to appear, making it hard to check correctness via testing. Thus they are an obvious candidate target for formal verification.

Streams as a data structure falls under the broader category of coinductive types. One can think of coinductive types in terms of coalgebraic structure dually to how inductive types can be thought of as algebras for certain functors. We can construct elements of inductive types from other elements of the inductive type and some input according to its structure. Dually, we can deconstruct elements of a coinductive type into elements of the coinductive type and some output data determined by the coinductive types structure. The basic example of natural number streams is conceptually an infinite, ordered list of natural numbers, and we denote the type by Str. It is the final coalgebra (or greatest fixpoint) for the functor $\mathbb{N} \times -$, and so by Lambek's theorem we have an isomorphism $\mathsf{Str} \cong \mathbb{N} \times \mathsf{Str}$. This isomorphism is the explanation for the deconstruction procedure described above: a stream can be transferred along this isomorphism, and composing with the projections we obtain either another stream (the tail operation) or a natural number (the head operation). That Str is a terminal coalgebra means that to obtain a map from A to Str for some A it suffices to produce a map $A \to \mathbb{N} \times A$. For instance we can define the constant stream of zeroes as a map $0 : 1 \to \mathsf{Str}$ by simply taking the map $1 \to \mathbb{N}$ which picks out the number 0.

While terminal coalgebras come with a natural mapping-in property in the semantics by virtue of being terminal, coinductive types are often specified via a mapping-out property, e.g., the taking apart of a stream into a head and tail as above. This means that once we are given, e.g., a stream we have a good handle on what data can be extracted from it. On the other hand this gives no a priori way to inhabit such types, since the implied constructor above concatenates a natural number with an already defined stream. The way to get solve this is general recursion, which we can add to type theory via a fixpoint operator, but as discussed earlier this is not consistent. One approach to ensuring that recursion does not result in pathological definitions is to verify syntactically that the recursion follows a pattern known to produce well-founded recursion. This approach is fairly inexpressive compared to the other approaches we will consider. Concretely, the syntactic checks seen in implementations usually force the recursive calls to appear directly under a constructor.

More sophisticated frameworks such as process theory needs more complicated coinductive types. To produce a type theory that it well suited for both programming with and reasoning about processes we need a sensible encoding of processes. One way to do this is via finitely branching labelled transition systems given as terminal coalgebras for $\mathsf{P}_{\mathsf{fin}}(- \times A)$ where $\mathsf{P}_{\mathsf{fin}}(-)$ is the finite powerset and A is the type of labels [28]. That encodings such as this are possible makes coalgebra a tool well-suited for working with infinite data structures.

1.1.3 Guarded recursion

Working with the input streams described as described in the previous section in functional programming requires recursion. As discussed in section 1.1.1 type theory usually corresponds to functional languages which provide termination guarantees, but these evidently cannot be given for reactive programs as described above. While termination guarantees are not possible we can obtain a *productivity* guarantee, i.e., a guarantee that finite stages of the program terminate in finite time. Both this and a promise of *causality* are present in the guarded recursive systems employing the modality $\triangleright$, and more concretely adding a fixpoint operator $\mathsf{löb} : (\triangleright A \to A) \to A$. This approach was pioneered by Nakano [42]. While traditionally recursion was subject to various syntactic checks, guarded recursion allows this to be handled as part of the type-checking. Many different variants of guarded recursion have been conceived [6, 8, 11, 17] and implemented [9, 29, 32, 56]. We think of the modality $\triangleright$ as an abstract step-indexing, with $\triangleright A$ being the type of elements in A available in the next time step. Data available now, i.e., elements of A, will be available later, meaning that we have a function $\mathsf{next} : A \to \triangleright A$. Since the input to $\mathsf{löb}$ is not an endofunction, strictly speaking it provides a fixpoint to the composition with next. As mentioned in the previous section, reactive programming with coinductive types can be achieved through $\triangleright$ and $\mathsf{löb}$. We start by describing the encoding of guarded streams, and then give an account of coinduction via guarded recursion.

We can encode streams with the later modality by ditching the strict version of the type isomorphism described above in favor of the following: $\mathsf{Str}^g \cong \mathbb{N} \times \triangleright\mathsf{Str}^g$. The type Str^g is a type of guarded streams, which break down into a natural number $n : \mathbb{N}$ and a term $s : \triangleright\mathsf{Str}^g$, the latter of which is a guarded stream available one time step from now. As for streams, we have head and tail operations but now the tail is not given to us immediately. Formally, the type of streams is itself defined by recursion over a type of type, i.e., the universe of type theory which we write U. It is the fixpoint of the map $\triangleright\mathsf{U} \to \mathsf{U}$ given by mapping A to $\mathbb{N} \times \triangleright A$. Likewise we inhabit the type of streams by using the fixpoint operator; for instance the constant zero stream is defined by $\mathsf{löb}(\lambda s.\, 0 :: s)$. While this type of guarded streams has its uses, it is too rigid for some applications. Functions $\mathsf{Str}^g \to \mathsf{Str}^g$ are strictly causal, meaning that, for instance, the stream operation picking out every other element cannot be typed. Apart from this, certain relations (in particular weak bisimilarity [38] as seen in [41]) are not causal, so this fragment of coinduction is inadequate.

In [6] a variant of guarded recursion was introduced with an indexed collection of modalities, each with a fixpoint operator in the style above which provides a solution to this issue. Instead of $\triangleright$ we have a collection of modalities indexed over an object of clocks, Clk, with clocks being denoted $\kappa, \kappa', \ldots$ and the corresponding modalities being $\triangleright^\kappa, \triangleright^{\kappa'}$, and so on. In keeping with the intuition that $\triangleright$ is supposed to internalize a concept of time-steps, we think of $\triangleright^\kappa$ as representing time steps on a particular clock. Replaying the construction above we define a type of guarded streams, Str^κ, satisfying a coinduction principle

which induces an isomorphism $\mathsf{Str}^\kappa \cong \mathbb{N} \times \triangleright^\kappa \mathsf{Str}^\kappa$. This multiclocked version of guarded recursion then adds a new type constructor which behaves like a function space from the object of clocks, with introduction and elimination given by abstraction and application with the expected reductions. We denote this type $\forall\kappa.A$. We add to this the primitive $\mathsf{force} : \forall\kappa.\, \triangleright^\kappa A \to \forall\kappa.A$, which is required to be inverse to the functorial action of $\forall\kappa$ on next^κ. Just as function spaces clock quantification commutes with products, and so we get a string of isomorphisms as follows:

$$\begin{aligned}\forall\kappa.\, \mathsf{Str}^\kappa &\cong \forall\kappa.\, (\mathbb{N} \times \triangleright^\kappa \mathsf{Str}^\kappa) \\ &\cong \forall\kappa.\, \mathbb{N} \times \forall\kappa.\, \triangleright^\kappa\, \mathsf{Str}^\kappa \\ &\cong \forall\kappa.\, \mathbb{N} \times \forall\kappa.\, \mathsf{Str}^\kappa\end{aligned}$$

One wrinkle in the calculation above is the lingering clock quantification on the natural number factor, for this to faithfully implement the coinductive stream type we at least need $\forall\kappa.\, \mathbb{N}$ to be isomorphic to $\mathbb{N}$. Types that satisfy this are called *clock irrelevant* and some instances of guarded type theories with multiple clocks take the property as an axiom for all types [13]. The collection of clock irrelevant types is closed under the standard type formers of type theory, and as such the primary problem to be solved in adding the axiom is the universes.

As mentioned earlier, we need coinduction to encode non-causal behavior and predicates. Passing to the above encoding does, however leave the door open for programming in a way that is guaranteed to be causal. We can introduce terms of, e.g., the coinductive stream type by constructing a guarded stream and then abstracting κ in it. This allows one to carefully isolate non-causal behavior and retain the causality guarantee when needed.

1.1.4 Dependent right adjoints and multimodal type theory

Guarded recursion as described above is an instance of a more general class of modalities, namely a dependent right adjoint [10]. The idea behind these modalities is to have an action on contexts L in addition to the type operation R. These are then essentially required to be adjoint, in some sense. Concretely, they consist of rules as below and an as of yet unspecified elimination principle.

$$\frac{L(\Gamma) \vdash A}{\Gamma \vdash R(A)} \qquad \frac{L(\Gamma) \vdash a : A}{\Gamma \vdash \mathsf{mod}_R(a) : R(A)}$$

While the formation and introduction rules for a type as above are clear enough, the elimination rule is a point of contention in the design of these systems. The simplest choice would be to add a formal dual to the introduction rule, defining an operation $\mathsf{unmod}_R(-)$ which allows extraction of a term under a $\mathsf{mod}_R(-)$, making the rule on the right in the above invertible. Modalities that follow this schema, and variants of it designed to capture the same adjunction-style relationship between L and R, are called *Fitch-style* modalities. Adding an inverse to the operation $\mathsf{mod}_R(-)$ is similar to specifying function spaces with a

λ and un-λ rule, and for similar reasons substitutions become problematic. We cover now the proposed solutions that are relevant to the papers.

In [10] the elimination rule for modal types requires a weakening in the conclusion, but the theory is augmented with a universal property for context of the form $L(\Gamma)$. Together this results in a theory in which the eliminator for modal types commutes with all definable substitutions. The approach taken in the first paper is similar to that of [10] and leans on prior work on clocked type theory [8, 36] in taking this function space analogy as far as it can go and working from there. Here, $\mathsf{mod}_\mu(-)$ is treated as λ abstraction of a variable with a restricted form of application as the elimination rule. In the context of guarded recursion with multiple clocks in particular, these protovariables are called ticks, and the type theories based on this approach are called multiple tick variants of guarded recursion [8]. This approach is well-suited for the particular modal type theory that is clocked type theory, but inherits the issue of dependent right adjunctions regarding the semantics: the elimination rule is closed under all definable substitutions, but it cannot be adequate with respect to the usual presheaf models. A modified version of this approach is described in the first paper, so we defer further discussion of this approach.

Another approach that exists in the literature is that of multimodal type theory (MTT) [23], which has the major advantage that the elimination rule for modal types is faithfully reproduced in the intended models. This is a fact we leverage to reason in the internal MTT of certain structures in the second paper. In MTT one allows for the abstraction of data available under a modality. More concretely, the context extension rule allows for the extension of Γ by types in context $L(\Gamma)$. Modal types are then required to support modal induction, which, following the general flavor of induction principles, can be approximated as saying the to construct an inhabitant of some type motive C which depends on $R(A)$ it suffices to construct it for terms of the form $\mathsf{mod}_R(a)$. The exact implementation of this is a form of let-binding which loosely matches the syntax present in [45]. MTT is typically also augmented by a modal variant of the Π-type allowing for the abstraction of the type assumptions which contexts can be extended by. Apart from providing a solution to the problem of substitutions in systems with Fitch-style modalities, a major advantage of MTT is the flexibility. Instead of having a single modality as sketched above, MTT is indexed by a 2-category $\mathcal{M}$ which we call a *mode theory*. The idea is that MTT consists of a copy of MLTT for each mode (object of $\mathcal{M}$) with a dependent adjunction for each modality (morphism of $\mathcal{M}$). Operations like next can be included as 2-morphisms in the mode theory, as it is essentially a transformation $\mathsf{id} \to \triangleright$. Having a centralized infrastructure for defining modal type theories has the advantage that many properties of modal type theories can be obtained by invoking a general result for MTT. For instance any instantiation of MTT has a model [23] and enjoys normalization [22]. Many modal type theories can be captured in the MTT framework, for instance modal S4, cohesive type theory, certain variants of guarded recursion, and a type theory for parametricity [23].

1.1.5 Equality in type theory

As mentioned earlier, this thesis will largely work in and with CTT. To motivate this choice we must first discuss equality in type theory more generally. In all type theories there is a structural notion of equality, called definitional or judgmental equality and one typically includes a notion of propositional or typal equality. Judgmental equality is, as the name implies, a separate judgement of type theory where typal equality is a (dependent) type. The typal equality is thus the equality we have access to in type theory as a logic. For judgmental equality, the point is to enshrine some chosen reduction rules into the structure of type theory. The constraint it is placed under is that it should have enough reductions to be helpful but not so many that the theory becomes trivial or, more likely, difficult to work with. Typal equality on the other hand is, essentially by definition, the least definable reflexive relation. It has an induction principle, path induction, but for some types we have to reason about equalities in them via extensionality principles. For now an extensionality principle is simply an alternative way to show equality in a type, but later we will need a more precise and stronger notion. The archetypical example is function extensionality, the ability to show equality of functions by showing that they produce equal outputs on equal inputs, or in other words that they have the same extensional behavior. Such principles are usually included in judgmental equality by default.

Judgmental equality, being a structural concept, is allowed to have (silent) structural consequences. The one most relevant to the discussion at hand is *type conversion*:

$$\frac{A \equiv B \qquad a : A}{a : B}$$

It is a desirable property for a type theory to have a decidable procedure for checking whether a particular term a has type A. This puts a cap on how complicated we can make judgmental equality. Since the check that $a : B$ needs to be decidable and one way to show it for $a : A$ is to show that $A \equiv B$ it requires that such proofs to not be overly abundant. It is similarly possible to define a type conversion function from a inhabitant of the identity type between two types using path induction, but this resulting function does not incur ambiguity of typing.

A rule which could be added to type theory is the reflection rule, allowing one to infer from the construction of a term in the identity type that the terms are judgmentally equal. There are two good reasons to do this, with the first being an easy way to obtain extensionality principles. Since these can be included at will for judgmental equality, a system with the reflection rule which collapses the two notions of equality can simply be required to include them. The second reason to use reflection is an inherent cap on the complexity of the identity type which can have at most one element (up to equality) with this rule. Both of these make it quite a bit easier to work inside the type theory. We call systems with the reflection rule extensional type theories and systems without it intensional, or say that the systems have an extensional or intensional identity

type respectively.

The reflection rule allows for type conversion along arbitrary proofs of equality, and the existence or non-existence of such proofs is not generally decidable. While decidability of type checking is desirable, it could conceivably be dropped for sufficient benefit; we will not go further along this line of reasoning here, instead we simply exclude the reflection rule as a possibility. This leaves us in a slightly awkward position of having to obtain extensionality principles such as function extensionality from elsewhere and having to accept a somewhat complicated identity type. The latter of these is useful in its own right, since a more complex type allows for a more expressive system, but the former is a problem we need to solve.

1.1.6 Cubical type theory

This thesis works in the paradigm of cubical type theory. There are many reasons to work in this paradigm: the intensional identity type which we argued in favor of in the preceding section, extensionality principles such as function extensionality and univalence and the added expressivity from *higher inductive types* (HITs). In addition to this, cubical type theory is well-studied, meaning that the most critical metatheoretic questions are settled definitively, specifically canonicity [26] and normalization [51], and that models of CTT are well-understood [18, 35, 44]. In contrast to, e.g., HoTT it does not allow one to introduce non-canonical elements of other types via the univalence axiom.

CTT is a dependent type theory extended with an interval primitive $\mathbb{I}$ with endpoints 0 and 1. We then define a path type given by functions out of this abstract interval, and the intention is to use the type of such maps as an equality type in line with the intuition from HoTT. To be slightly more concrete, equalities between a and b in A are maps from the interval into A which are judgmentally equal to a at 0 and b at 1. It is a slogan of any logic that equality is the least relation that is all three of reflexive, symmetric, and transitive and type theory is no different. Showing that the path relation is reflexive is easy: simply take the constant function. The other properties are more tricky and require some combination of operations in the type and extra structure on the interval. We will work with the most structured interval and the least powerful operations following [18]. This means concretely that we require that the interval is a De Morgan algebra, and as such has connections $\wedge$ and $\vee$ which can be seen as abstract minimum and maximum operations respectively, as well as a reversal. This reversal can be used to show that paths are symmetric. In addition to this, we note that as described in section 3.2 of [18] connections can be used to show that singletons are contractible.

To show that paths are transitive we need *Kan operations*, which are an abstract, cubical version of the horn filling property for simplicial sets as in the description of Kan fibrations. In the tradition of [18] this is bundled into a single operation called *composition*. These operations allows one to extend certain partial elements to total ones, namely those partial elements which are (path) equal to an extensible element and are specified on a formula from the *face*

lattice. The way to think about the face lattice is as a collection of subobjects of geometric cubes, specifically those corresponding to a union of faces and subfaces of a cube. These Kan operations can then be used to show transitivity of the path relation (a proof is reproduced in section 2 of the first paper). Moreover the Kan operations can be used to define transport, a special case of composition where one is extending an empty partial element. Together with the aforementioned contractibility of singletons this can be used to show that the path type supports path induction, and so ticks the boxes one might expect of an identity type.

While the path type cannot support a reflection rule in a sensible way, it does introduce more judgmental equalities into the theory, some directly and some via the supporting composition operation. Directly from the path type we get the required reductions: a path from a to b evaluated at 0 must reduce judgmentally to a. At the same time composition is meant to provide an extension of a partial element and so must reduce judgmentally to the partial element it is extending whenever restricted to a face formula on which the partial element is specified. To ensure that these do not break decidability of type checking out of hand these criteria must themselves be decidable. This puts some pressure on the interval and face lattice, but syntactically it boils down to checking equality in a free De Morgan algebra and inequality in a free distributive lattice, both of which are decidable.

Finally, there was a promise of extensionality principles as theorems. An extensionality principle is at its core simply a description of the identity type of a certain type. More formally, we take extensionality principle for A to mean an equality of type between the identity type of A and some other type[4]. As mentioned earlier, CTT satisfies univalence, so in this theory we can obtain extensionality principles from isomorphisms. The most straight-forward one is perhaps function extensionality, the statement that an equality between functions is an equality between the outputs of the functions at every possible input. In extensional type theory this is derivable but in general it is not so for intensional type theories. CTT validates function extensionality by way of a proof that swaps the input order to a certain function.

The second important extensionality principle validated by CTT as a theorem is the univalence axiom. The proof of univalence is significantly more complicated than the one line proof of function extensionality, but it nevertheless does not necessitate the addition of undecidable judgmental equalities. As mentioned above, univalence is what enables us to derive extensionality principles from, e.g., the type isomorphism indicated in the description function extensionality above. In fact function extensionality can also be obtained directly from univalence, but the proof above is simpler and generalizes better to

[4] In a different setting one might take extensionality principle to mean a principle for inferring equality in a type. We make this stronger statement because we work in intensional type theory which is proof relevant, or more specifically with a homotopical variant of type theory where the (higher) identity types are themselves objects of interest. A slight weakening that would be acceptable in this setting might be to require an equivalence of types instead, but in a univalent context these are the same.

modal types. In addition to this we obtain a slew of extensionality principles, such as the one for Σ-types, coproducts, and many others, from univalence via type equivalences.

Another important feature of CTT (or more generally any implementation of HoTT) are the aforementioned HITs, which are covered in, e.g., chapter 6 of [54]. While inductive types will have constructor only for elements, HITs have in addition constructors for the identity type as a built in quotienting up to path. A simple example of the concept is the circle, an inductive type with a single point and a freely added loop on that point. More generally we might expect to be able to represent any finite (homotopy) colimit internally with the syntax of HITs and we can as a special case construct quotients, which have been challenging to represent internally in type theory. A large class of modalities in type theory (although not the ones we will work with) can be described with HITs as nullification of maps out of some indexing family [47]. A particularly important example of such a modality is truncation, which nullifies maps out of the sphere. The effect of this is to trivialize all higher paths. Types with only trivial paths in dimension n and above are said to be of h-level n. In particular 1-types are called h-sets and have no non-trivial paths. Apart from this, HITs are a useful tool for representing free algebraic structure over a type by freely adding both the operations and equalities as constructors. HITs can be combined, and a particularly important such combination is the set truncation of the free commutative monoid, i.e., the finite powerset functor. In combination with recursion, the finite powerset functor can be used to encode a type of finitely branching labelled transition systems. This structure is useful for process theory, as we will discuss in more detail later.

1.1.7 Denotational semantics

Because model construction is a large part of both papers making up this thesis, we discuss the payoff that one might hope to obtain from such work. Models are useful in many respects but the two main ones are showing consistency of a given type theory and gaining further insights about the theory via its semantics. At a high level, *denotational semantics* is the process of ascribing mathematical meaning to expressions in a programming language or logical theory [48]. More concretely, this means that for some theory $\mathcal{L}$ we define a mathematical structure $\mathcal{M}$ and an interpretation function $[\![-]\!] : \mathrm{Expr}(\mathcal{L}) \longrightarrow \mathcal{M}$ where $\mathrm{Expr}(\mathcal{L})$ is the set of expressions in the theory $\mathcal{L}$ sometimes considered up to some form of equivalence. If such a function enjoys *soundness*, the property that expressions provable in $\mathcal{L}$ are true in $\mathcal{M}$ when we apply $[\![-]\!]$ to them, we call $[\![-]\!]$ a *model* of $\mathcal{L}$ or say that $\mathcal{L}$ is a model of $\mathcal{L}$

Models can be useful for many purposes, we highlight those particularly interesting in the study of cubical modal type theory below. Firstly, it can be used to show consistency of a theory. Let us consider in detail how to prove dependent type theory consistent and what such a proof might buy us in terms of formal verification. Consistency is the property that a theory cannot prove a contradiction, and in particular for type theory it is the inability to derive the

judgement $1 \vdash t : 0$ where 1 is the empty context and 0 is the empty type. One can show that MLTT is consistent by constructing a model in the category of sets. Here a term of the empty type would be interpreted as a map of sets from the singleton set to the empty set, and no such map exists. Any proposition can be proven in an inconsistent logic, so it has obvious value when we consider type theory as a logic. In terms of formal verification, inconsistency only allows one to retain the most basic guarantees, e.g., the fact that functions can never be applied to input they are not defined on, but we lose all possibility of using type theory as a logic for specifications.

In addition to the above, many other uses exists for denotational semantics. The consistency discussion above is an important part of verifying that newly added axioms lead to a sound theory, but this process is often a two-way street. A recent high profile example is that of univalence. In around 2005, Voevodsky produced a model of dependent type theory in Kan simplicial sets and observed that this model satisfied the univalence axiom. The idea that one might be able to produce models of intensional type theory with homotopically flavored identity types was present elsewhere [7], but the univalence axiom which crystalizes this in the theory itself was extracted from a model. While the univalence axiom indeed captures an important property of working with intensional type theory, axioms do not generally have any computational meaning. This sparked a search for models of univalence which deconstructs the axiom into more manageable pieces than a monolithic property, resulting in the development of cubical type theory.

Guarded recursion is also to some extent a result of extracting a theory from a semantic construction. In [11] the topos of trees is shown to model guarded recursion, and is then mined for insights into the theory by working in its internal language. In particular it is used to establish the interaction between $\triangleright$ and the propositional logic of type theory. The multiclocked variants of guarded recursion include a clock synchronization substitution [12] which was not included in the theory until a suitable model was found.

Another example of this is the recent advances in *normalization by evaluation* (NbE) in the form of *synthetic Tait computability* (STC) [1, 52]. The method of STC starts with the observation that the internal language of a topos obtained via Artin gluing admits a useful description in terms of the open and close modalities [47]. One can then work in this language to obtain the NbE result without an obligation to check naturality of the constructions involved, a significant boon for the scalability of NbE. This construction has at this point been used to obtain normalization for both MTT [22] and CTT [51], with the MTT normalization proof obtained this way being significantly shorter than the canonicity proof obtained with traditional methods.

1.2 Contributions

1.2.1 Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks

The goal of the first paper is to define and prove sound an implementable type theory expressive enough for reasoning about infinite structures both in a guarded way and coinductively. As we have seen throughout the previous sections the former is achieved by cubical type theory, and the latter is achieved in the variant of guarded recursion with multiple clocks. The contribution of the first paper is the definition of Clocked Cubical Type Theory (CCTT), which is a combination of the two theories, as well as the construction of a model of this theory. The problems we face in the definition of this theory is how to retain the good computational properties of cubical type theory and the operationalization of axioms such as clock and tick irrelevance from clocked type theory [36] as well as the extension of coinductive reasoning to include the expressivity of HITs. In addition, the theory must support extensionality principles for the type formers inherited from clocked type theory.

For CCTT to fulfill its stated purpose of being suitable for reasoning about coinduction, we need the theory to have a supply of functors which commute with clock quantification and clock irrelevant types. The reason for this was discussed in section 1.1.3. We described an encoding of coinductive types for reactive programming using guarded recursion. In the extensional case this procedure is well understood and in § 2 we extend this to a cubical setting. The general formulation of the encoding result lets one produce final coalgebras for functors F which commute with clock quantification, or, in other words, it allows us to produce solutions to type equations $X \simeq F(X)$ when the canonical map $F(\forall \kappa.\, X) \to \forall \kappa.\, F(X)$ is an equivalence. We have already illustrated how this works for the functor $(-) \times \mathbb{N}$ in a type theory where the natural number type $\mathbb{N}$ is clock irrelevant, but reactive programming is a broader subject than stream operations. In particular, we can encode processes as finitely branching labelled transition systems. To represent these in type theory we need the finite powerset functor. This type former has traditionally been difficult to handle because the construction includes quotients and it is handled in the cubical setting using higher inductive types. For this particular encoding to work we need the functor $\mathsf{P}_{\mathsf{fin}}(- \times A)$, where A is the type of labels, to commute with clock quantification in CCTT.

This is achieved by adding a novel induction principle for HITs, *induction under clocks*. This new induction principle allows one to structurally eliminate out of $\forall \kappa.\, H$ directly where H is a higher inductive type, and it is used to show that many higher inductive type formers, including $\mathsf{P}_{\mathsf{fin}}(- \times A)$, commute with clock quantification. More concretely, HITs support elimination principles similar to those for inductive types allowing one to specify a map out of, e.g., the propositional truncation of A by specifying it on $\mathsf{in}(a)$ and $\mathsf{squash}(x, y, i)$. These definitions are subject to boundary constraints ensuring that the value of the path constructor squash coheres with the recursively obtained value on x and y

on the respective endpoints. Induction under clocks allows for this case analysis under an abstracted clock, so that we can define a map $\forall\kappa.\|A\|$ by defining it on $\lambda\kappa.\,\mathsf{in}(a[\kappa])$ and $\lambda\kappa.\,\mathsf{squash}(x[\kappa], y[\kappa], i)$, with a boundary condition on the second case as for the usual induction principle. Using induction under clocks we define a map α as follows:

$$\begin{aligned}
&\alpha : \forall\kappa.\|A\| \to \|\forall\kappa.\,A\| \\
&\alpha(\lambda\kappa.\mathsf{in}(a[\kappa])) = \mathsf{in}(a) \\
&\alpha(\lambda\kappa.\mathsf{squash}(x[\kappa], y[\kappa], i)) = \mathsf{squash}(\alpha(x), \alpha(y), i)
\end{aligned}$$

One can show that this map is inverse[5] to the canonical map in the opposite direction by separate applications of each of the induction principles available, showing that propositional truncation commutes with clock quantification. As a corollary we obtain the fact that clock irrelevant types are closed under propositional truncation, since for clock irrelevant A we have $\forall\kappa.\|A\| \simeq \|\forall\kappa.A\| \simeq \|A\|$. A similar line of argument is carried out in the paper for more examples, extending the pool of functors that commute with clock quantification by pushouts, higher truncations, and the finite powerset functor. In addition we prove a general theorem extending the closure of clock irrelevant types under higher inductive types to all higher inductive types. This solves the stated problem of obtaining a supply of functors and types which interact favorably with clock quantification.

Now that we have provided a large class of functors commuting with clock quantification and clock irrelevant types, we can leverage the encoding of coinductive types to give a faithful, safe implementation of coinduction in type theory. The coinductive types constructed via the encoding result presented in the paper are conjectured to have an identity type coinciding with bisimilarity in the spirit of [40], supported by a proof of this fact in the case of finitely branching labelled transition systems. In addition to this, and the usual trappings of cubical type theory in the form of univalence and function extensionality, CCTT supports two extensionality principles specific to clocked primitives. To state the extensionality principle properly we must first explicate the $\triangleright$ operator: In this system the official syntax is $\triangleright(\alpha : \kappa).A$, and we say that the tick α on the clock κ is abstracted in the type A. We show in the paper that for any $x, y : \triangleright(\alpha : \kappa).A$ we have an equivalence (and hence equality) of types $\triangleright(\alpha : \kappa).(x[\alpha] =_{A[\alpha]} y[\alpha]) \simeq x =_{\triangleright(\alpha:\kappa).A} y$. Furthermore, clock quantification has an extensionality principle akin to function extensionality.

Obtaining functors which commute with clock quantification and clock irrelevant types via the principle of induction under clocks has the advantage that it does not rule out an implementation of the theory. Clocked type theory needs a second irrelevance priciple that we have only mentioned so far: tick irrelevance. CCTT features an operationally motivated implementation of this axiom which is again compatible with implementability. Had we assumed these irrelevance principles as axioms, as is done in other version of the multiclocked

[5] One could instead say quasi-inverse here, since it is only inverse up to a path.

theory, no such implementation would be within reach. We conjecture that the theory as given in the paper supports a decidable normalization procedure and hence a decidable type checking algorithm. This conjecture is supported by the normalization result for a core calculus of clocked type theory without identity types [8].

The composition operation of cubical type theory is well-behaved in part because it supports reductions to composition in constituent types. In CCTT composition in $\forall\kappa.\, A$ reduces to composition in A similarly to the reduction given for Π-types, and composition in $\triangleright^\kappa A$ supports a reduction to composition in A under certain circumstances, with the reduction being extractible from the construction in the model. That composition reduces in a sensible way is another point for the implementability conjecture. The sum total of these parts is a fully furnished type theory for programming with and reasoning about a large class of coinductive types.

The theory is shown to be sound via the construction of a denotational model in presheaves over the product of the cube category used for modelling cubical type theory and the category of time objects used to model guarded recursion with multiple clocks. The novelties of the model are the description of induction under clocks and the composition structure for $\triangleright^\kappa A$. The higher inductive types included in the theory are given by a schema adapted from [15], with the semantics being a variant of what is described in [20] adapted to this extended presheaf category. Applying a result from [13] characterizing the semantics of clock quantification as a certain limit we then show that induction under clocks is validated in the model. The composition structure for $\triangleright^\kappa A$ is obtained by making requirements on the left adjoint to $\triangleright^\kappa$, in particular that it preserves the interval and other cubical structure.

1.2.2 Unifying modal and multimodal type theory

In section 1.1.4 we pointed out the utility in having a single unified approach to modal type theory in the form of the parameterized MTT. As both an instance and extension of this program, we present in the second paper a cubical variant of MTT, Cubical Multimodal Type Theory ($\mathsf{MTT}_\square$). The motivation for defining this type theory is much the same as for the first paper. There is a tension between the expressivity of the theory as mediated by extensionality principles and the implementability. Cubical type theory is an implementable theory with a good supply of extensionality principles derived from either univalence or directly. Thus one might hope that a combination of multimodal type theory and cubical type theory offers an implementable type theory with both the well-behaved equality types of cubical type theory and the flexibility of modal constructions afforded by multimodal type theory. This should be supported by, in particular, a reduction rule for composition in modal types and an extensionality principle for equality in modal types. While MTT is flexible, it does not cover all axioms of, e.g., guarded recursion, where löb has to be added separately. Therefore we need a method for constructing model of instantiations of $\mathsf{MTT}_\square$ for it to be a useful framework. Both of these problems are resolved

in the paper. To be specific, recall that MTT consists of local copies of MLTT. These will be swapped for local copies of CTT in $\mathsf{MTT}_\Box$. We can derive a modal extensionality principle in the same vein as the one described the first paper, this time for any modal type. In addition we can type a computation rule for composition in modal types.

The primary challenge in the design of $\mathsf{MTT}_\Box$ is in specifying how the cubical primitives in one mode should interact with those in other modes. The machinery that makes the theory work is a collection of *exchange operations* which mediate the interaction of the modalities with the cubical primitives. The guiding principle is that the cubical judgments should not be affected by the modalities. This principle is reified as two operations which each induce a substitution, as well as inverses to these induced substitutions. These operations and substitutions are given in fig. 3 of chapter 3. In the notation of section 1.1.4, the operation allow one to access cubical data available in context Γ when working in $L(\Gamma)$, i.e., interval and face lattice terms.

To describe the contributions in detail we need to recall some notation from CTT and MTT. In cubical type theory we have the interval primitive, which is implemented with a context extension rule and a term construction. The interval terms in context Γ are generated according to the following grammar:

$$\textit{(Abstract interval)} \quad r, s : \mathbb{I} \quad ::= \quad i \mid 0 \mid 1 \mid 1 - r \mid r \vee s \mid r \wedge s$$

The terms constructed from this grammar are subject to the De Morgan algebra equations. From MTT we need the context judgments and the modal action on context. Contexts in MTT are annotated by a mode $m : \mathcal{M}$, written as $\Gamma\ \mathsf{cx} \mathbin{@} m$, and we can move a context from one mode to another via the morphisms of $\mathcal{M}$. Concretely, from $\mu : n \to m$ and $\Gamma\ \mathsf{cx} \mathbin{@} m$ we can obtain $\Gamma, \{\mu\}\ \mathsf{cx} \mathbin{@} n$. This context operation is required to be strictly 2-categorical, meaning, for instance, that we have the following equality of contexts: $\Gamma.\{\mu \circ \nu\} = \Gamma.\{\mu\}.\{\nu\}\ \mathsf{cx} \mathbin{@} m$.

In $\mathsf{MTT}_\Box$ we extend the interval grammar with the an operation $-^\mu$, given by the left rule below. From this rule we can construct a substitution $\Gamma, i : \mathbb{I}_m, \{\mu\} \vdash \sigma_\mu : \Gamma, \{\mu\}, j : \mathbb{I}_n \mathbin{@} m$ defined by extending the weakening substitution under $\{\mu\}$ by i^μ. On the right in the below rules we have the introduction for a substitution σ^μ, which we require to be inverse to σ. Both the operation and the defined substitution are required to satisfy equalities making them lax natural and strictly 2-functorial in μ respectively, just as the action on contexts is strictly 2-functorial. This is necessary to avoid ambiguity in exchange operations, which would otherwise induce infinitely many a priori different endoisomorphisms on any context via exchanging along the identity any number of times. The operation is furthermore required to commute with De Morgan algebra connectives.

$$\frac{\Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma, \{\mu\} \vdash r^\mu : \mathbb{I}_n \mathbin{@} n} \qquad \frac{}{\Gamma, \{\mu\}, i : \mathbb{I}_n \vdash \sigma^\mu : \Gamma, j : \mathbb{I}_m, \{\mu\} \mathbin{@} n}$$

The face lattice restrictions on contexts are similarly required to commute with $\{\mu\}$, which is split into an operation as on the left above for the face lattice and

a derivable substitution to which we add an inverse, with both added primitives required to cohere with the 2-categorical structure of $\mathcal{M}$ as appropriate. Together this data allows us to achieve our stated goals of obtaining a reduction rule for modal composition and an extensionality principle for modal types. We type a reduction rule for composition in $\langle \mu \mid A \rangle$ to composition in A provided that the input system to the composition is obtained by applying $\mathsf{mod}_\mu(-)$ in section 3.2 of the second paper. Additionally we derive a modal extensionality principle. As for previous derivations of such extensionality principles we use the explicit representation of equality given by the path type in this derivation.

Apart from giving a workable setup for the combination of modal and cubical type theory, the primary contribution is a model construction, essentially extending the theory of Orton-Pitts style models to multimodal type theory. This is shown to produce a large class of presheaf models covering many important examples out of the box.

The syntax of $\mathsf{MTT}_\square$ is given as a generalized algebraic theory, meaning in particular that it comes equipped with a notion of model. This model definition can be neatly packaged in the language of 2-categories, giving a sensible notion of categorical model for $\mathsf{MTT}_\square$. Just as directly modelling other type theories, producing models according to this specification is mostly intractable. Our contribution on the semantic side is a method for constructing models from categorical data. Just as $\mathsf{MTT}_\square$ can be thought of as a collection of copies of CTT connected by modalities, so too will the models consist of models of CTT at each mode connected by dependent adjunctions. We define cubical cosmoi, fusing two concepts from earlier work. A cubical cosmos over $\mathcal{M}$ is a pseudofunctor $F : \mathcal{M} \to \mathbf{Cat}$ such that each category $F(m)$ can be given the structure of a model of CTT and each $F(\mu)$ has a left adjoint which interacts appropriately with the cubical structure. More concretely, each $F(m)$ is an elementary topos with a tiny interval and a face lattice subject to certain axioms as in [44, 35] which is sufficient for it to furnish a local model of CTT. In addition, the value of F on morphisms must be functors with lex left adjoints which preserve the interval and face lattices up to chosen isomorphisms, with the data of these isomorphisms assembling into a pseudonatural transformation.

Any cubical cosmos induces a model of $\mathsf{MTT}_\square$, which is the main semantical result of the second paper. The proof of this theorem combines a strictification result to obtain a model of MTT, the LOPS-style approach to model the cubical components of the theory, and direct calculations to model the exchange operations. To produce examples of cosmoi, it is shown that given a strict 2-functor $f : \mathcal{M} \to \mathbf{Cat}$ we obtain a cubical cosmos in the network of presheaf categories $\mathbf{PSh}(f(m) \times \square)$, where $\square$ is the De Morgan cube category, connected by either precomposition by $f(\mu) \times \mathsf{id}$ or right Kan extension of these functors. In particular this is used to produce a model of a variant of guarded recursion with a single clock and a modality which serves roughly the same purpose as clock quantification in the multiclocked setting.

The result is a framework for defining modal type theories with a strong justification in a model construction. This framework encapsulates both the flexibility of modal reasoning present in MTT and the computationally well-

behaved identity type of CTT. We further conjecture that the normalization results based on STC for MTT [22] and CTT [51] can be combined so as to show normalization for $\mathsf{MTT}_{\Box}$.

1.3 Related work

This section will cover the work that we built directly on in the papers that did not find a home in the background section as well as some work that purports to solve the same or similar problems.

Related variants of guarded recursion

The variant of guarded recursion presented in the first paper is a successor to two other variants of guarded recursion, namely ticked cubical type theory [40] and clocked type theory [8, 36]. These theories are the offshoot from guarded recursion that uses named ticks in the later modality, ticks can then be abstracted in the context and an associated elimination rule for $\triangleright^{\kappa}$ given by a controlled form of function application. Ticked cubical type theory is a version of this theory with a single clock in a cubical background theory with clocked type theory being the version with multiple clocks.

On their own, each of these theories enjoy some, but not all, of the desirable properties of clocked cubical type theory. Clocked type theory supports an encoding result similar to the one presented in the first paper, allowing for programming with and reasoning about coinduction. It does not, however, support higher inductive types, and as such the encoding of finitely labelled transition systems is not present in that theory. The presentation of clocked type theory in [8] contained no identity types, but [36] updated the theory with an extensional identity type adding some much needed expressivity. Unfortunately the extensional identity type is not suited for implementation and reasoning about equality in modal types. Ticked cubical type theory on the other hand exists in a cubical setting, meaning that it does not have these defects. Here the problem is in the expressivity of the recursion scheme. Because the theory does not have multiple clocks and clock quantification it is limited to guarded types that do not support true coinduction principles.

Another variant of guarded recursion related to this approach is the one presented in the last section of the second paper. Here we operate with a single later modality, but enhance the theory with a modality $\Box$ which behaves similarly to clock quantification [17]. The crucial bit is the counterpoint to the force primitive, an isomorphism $\Box \triangleright A \simeq \Box A$, which allows for an encoding result in the style of clocked type theory. In terms of expressivity, the cubical variant of this theory presented in the second paper is similar to clocked cubical type theory; it lacks only the ability to represent nested coinductive types which rarely appear in practice. While this theory might seem simpler to work with on its face, the fact that clock quantification is represented as a function space makes it simple to work with in type theory. In the context of cubical versions of the theory, this means that clock quantification supported a composition

structure and an extensionality principle, both obtained by the same arguments as for Π-types and thus well understood from the beginning. There is no clearly better theory between the two; one cannot present clocked type theory as an instance of MTT without losing the simple description of clock quantification and the $\Box$ modality cannot be cast in terms of a function space to our knowledge.

Recall that the arguments for the desirability of coinduction rests on two different applications; firstly the definability of relations such as weak bisimilarity and secondly on the non-causality of certain functions known to be productive, e.g., the function returning every other element of some input stream. One principled solution to the latter problem has been proposed in the form of time-warps [24]. While $\triangleright$ is conceptualized as pushing data one time-step into the future, time-warps allow for the representation of more general time operations. One instance of time-warps allow for the definition that steps twice as fast and applying this in the case of streams we can define the function described above via this warp: the n'th output of the stream is allowed to depend on the $2n$'th input! This allows for the typing of a wide array of functions not typable without appealing to general coinduction in the systems described above. This approach is able to keep track of not just whether some definition is productive, but also exactly how time flows in some defined stream, i.e., *how* it is productive. Time-warps are a promising alternative guarded recursive variant for further research.

Sized types

Sized types is a different paradigm for what amounts to abstract step indexing, and has been deployed to ensure termination of structural recursion over inductive types and productivity of coinductive programs [27]. The core idea is to attach to each term of a type a *size*; for inductive types this size is a natural number bounding from above the depth of the term. For coinductive types such as streams, sizes can be used to track whether functions are unfolding or folding a stream, promising a theory in which recursive depth is tracked systematically.

While sized types purports to solve similar problems to those solved by guarded recursion, the theoretical foundations are quite dissimilar. Both approaches are integrated with dependent types, but the study of their semantics have taken different approaches. As we have seen throughout this thesis, guarded recursion is well-understood through the lens of denotational semantics; sized types have primarily been investigated operationally. Dependent sized types are type safe [3] and enjoy normalization via an NbE algorithm [4]. Furthermore they have been implemented in Agda [2], and this implementation has been used to encode coinductive types. It is formally connected to guarded recursion by an encoding of guarded recursion using sized types in Agda [55]. Unfortunately the current implementation of sized types is inconsistent because of its use of the infinite size[6]. It also does not have a denotational semantics to our knowledge.

[6] This is due to an issue described at https://github.com/agda/agda/issues/2820

Modalities in HoTT

A different conceptualization of modalities in type theory is that of certain idempotent monads as developed in [47]. This work is relevant to the papers both because they concern modalities and to the first in particular because it offers a potential solution to the problem of clock irrelevant universes. The starting point for that investigation is to notice that these modalities admit four equivalent descriptions, each suitable for different proofs. In particular, a class of idempotent monadic modalities are characterized which have the property that the type $\Sigma(A : \mathsf{U}).\mathsf{isModal}_{\bigcirc}(A)$ is itself modal with respect to the modality $\bigcirc$. The modalities they consider include those given by nullification of an object, and under certain circumstances such modalities allow for internal definition of modal universes of modal types.

Clock irrelevance can be thought of as nullification of the clock object, although it is not an object in the theory, the model describes it as one. Thus one might hope that the above universe construction applies in this case. Unfortunately it does not satisfy the assumptions for this construction.

Cohesive type theory

Cohesive type theory is related to the present work for two reasons: firstly it is representable as an instance of MTT with the mode theory spelled out in [14], and secondly it is another instance of a modal type theory in which a modal universe of modal types is constructed. It is helpful to keep the following picture in mind, adapted from [49]:

$$\begin{array}{c} \text{Spaces} \\ p_! \downarrow \dashv \uparrow p^* \dashv \downarrow p_* \dashv \uparrow p^! \\ \text{Sets} \end{array}$$

In terms of the first point, cohesive type theory has been applied as a tool for connecting theories with different axioms. In the above picture, we think of spaces as having more structure and thus fewer admissible rules. There are three recent examples of the mode of application for cohesive type theory. It was conceived in the connection of a theory of topological types with a theory of "sets" to allow for the application of classical axioms such as the law of excluded middle in a controlled fashion [49]. This allows one to connect results from synthetic homotopy theory as derived in HoTT to the theory of topology formalizable in HoTT, with the concrete application given by Shulman in [49] the paper being a proof of Brouwer's fixpoint theorem. While homotopy theory has applications apart from those in topology, this connection is an important addition to the topologists toolbox in HoTT.

Another application is that given by Kavvos in [31]. Here a flavor of cohesion is set up to describe information flow, with the "spaces" in the above picture being sets with an equivalence relation, and the "sets" being actual sets. The total equivalence relation on a set amounts to a secret set, and the minimal

relation amounts to a public set. Kavvos uses an adequate model of this to show that the resulting calculus for information flow control satisfies the critical non-interference property, meaning that public programs cannot distinguish elements of secret sets. The theory as it is extensional, meaning that it is unsuitable for implementation and the results in the second paper is a promising avenue for implementation.

A final example is that of internal parametricity in cubical type theory [16]. Here a type theory with primitives for using parametricity to show "free theorems" [57] and the by now familiar cubical primitives is defined and given a model. In this theory many problematic proofs become manageable, with the most striking example being their treatment of the smash product of types which is a particularly complicated structure in theories with higher equalities. The idea is that using these primitives will let us reason about the smash product in cubical type theory, but a mathematician interested in this smash product might reasonably point out that they are interested in the cubical smash product, not the smash product in cubical type theory with extra primitives! Cavallo endeavors to solve this using cohesion in his thesis [14]. In particular he defines a variant of cohesive type theory where the spaces are parametric types and the sets are regular cubical types. It is shown that the theorems obtained in the parametric mode can be carried over to the ordinary cubical mode, thus completing the program of giving a scalable approach to working with structures that incur many coherence obligations, e.g., the smash product. The model produced for this cohesive theory has some of the same challenges as those present in the second paper, but the solution relies on the specifics of the situation and thus does not scale to other settings where one might want to connect theories with cohesion.

Finally, there is a possibility that one can construct clock irrelevant universes of clock irrelevant types via the methods used to obtain a discrete universe of discrete types in [49]. In cohesive type theory, discrete types can be conceptualized as those types lacking certain structure. The setup that might be helpful for producing clock irrelevant universes would have strictly clock irrelevant types for sets and regular types for spaces. Semantically there does exist a model of such a cohesive type theory, but it is not quite enough. The result we have from [49] does not produce proper universes, and the gap has yet to be closed in the proposed model for cohesive clock irrelevance.

1.4 Statement of contributions

- M. B. Kristensen, R. E. Møgelberg, A. Vezzosi, "Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks". Submitted for publication at LICS 2022.

  I participated in the design of the theory defined in the paper, in particular taking lead on the development and application of induction under clocks. I contributed most of the denotational model given in the appendix to the paper. I participated in writing all parts of the paper and wrote the

section on induction under clocks and the section on semantics.

- F. L. Aagaard, M. B. Kristensen, D. Gratzer, L. Birkedal, "Unifying modal and multimodal type theory". Submitted for publication at LMCS.

  I participated in the design of the theory and the development of the semantics. In particular I contributed the initial definitions and proofs that lead to the notion of cubical cosmoi as models of $\mathsf{MTT}_{\Box}$, as well as the derivation of the composition structure on modal types and the reduction rule this operation satisfies. I participated in writing the paper, taking lead on the section on MTT and CTT, with the sections on $\mathsf{MTT}_{\Box}$ and its semantics following the logical progression laid out in the initial notes written be me.

BIBLIOGRAPHY

[1] Andreas Abel. *Normalization by Evaluation: Dependent Types and Impredicativity*. Habilitation.

[2] Andreas Abel. Miniagda: Integrating sized and dependent types. In Ana Bove, Ekaterina Komendantskaya, and Milad Niqui, editors, *Workshop on Partiality And Recursion in Interative Theorem Provers (PAR 2010), Satellite Workshop of ITP'10 at FLoC 2010*, 2010.

[3] Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns programming infinite structures by observations. volume 48, pages 27–38, 01 2013.

[4] Andreas Abel, Andrea Vezzosi, and Theo Winterhalter. Normalization by evaluation for sized dependent types. *Proc. ACM Program. Lang.*, 1:33:1–33:30, 2017.

[5] Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[6] Robert Atkey and Conor McBride. Productive Coprogramming with Guarded Recursion. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 197–208. Association for Computing Machinery, 2013.

[7] Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, 2009.

[8] Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. The clocks are ticking: No more delays! In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2017.

[9] Patrick Bahr, Christian Uldal Graulund, and Rasmus Ejlers Møgelberg. Simply ratt: A fitch-style modal calculus for reactive programming without space leaks. *Proc. ACM Program. Lang.*, 3:109:1–109:27, 2019.

[10] Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118–138, 2020.

[11] Lars Birkedal, Rasmus Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4), 2012.

[12] Aleš Bizjak and Rasmus Ejlers Møgelberg. A model of guarded recursion with clock synchronisation. *Electronic Notes in Theoretical Computer Science*, 319:83 – 101, 2015. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

[13] Aleš Bizjak and Rasmus Ejlers Møgelberg. Denotational semantics for guarded dependent type theory. *Mathematical Structures in Computer Science*, 30(4):342–378, 2020.

[14] Evan Cavallo. *Higher inductive types and internal parametricity for cubical type theory*. PhD thesis, Carnegie Mellon University, 2021.

[15] Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[16] Evan Cavallo and Robert Harper. Internal Parametricity for Cubical Type Theory. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[17] Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. Programming and reasoning with guarded recursion for coinductive types. In Andrew Pitts, editor, *Foundations of Software Science and Computation Structures*, pages 407–421. Springer Berlin Heidelberg, 2015.

[18] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:34, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[19] Thierry Coquand. Infinite objects in type theory. In *Proceedings of the International Workshop on Types for Proofs and Programs*, TYPES '93, page 62–78, Berlin, Heidelberg, 1994. Springer-Verlag.

[20] Thierry Coquand, Simon Huber, and Anders Mörtberg. On Higher Inductive Types in Cubical Type Theory. In *Proceedings of the 33rd Annual*

ACM/IEEE Symposium on Logic in Computer Science, LICS '18, pages 255–264, New York, NY, USA, 2018. ACM.

[21] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, May 2001.

[22] Daniel Gratzer. Normalization for multimodal type theory. *CoRR*, abs/2106.01414, 2021.

[23] 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.

[24] Adrien Guatto. A generalized modality for recursion. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18. ACM, 2018.

[25] Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *Computer Science Logic*, 1994.

[26] Simon Huber. Canonicity for cubical type theory. *Journal Automated Reasoning*, 63(2):173–210, 2019.

[27] John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '96, page 410–423, New York, NY, USA, 1996. Association for Computing Machinery.

[28] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.

[29] RALF JUNG, ROBBERT KREBBERS, JACQUES-HENRI JOURDAN, ALEŠ BIZJAK, LARS BIRKEDAL, and DEREK DREYER. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming*, 28:e20, 2018.

[30] K Kapulkin and K Szumilo. Internal languages of finitely complete (?, 1)-categories. *Selecta Mathematica*, 25, June 2019. © Springer Nature Switzerland AG 2019. This is a post-peer-review, pre-copyedit version of an article published in Selecta Mathematica. The final authenticated version is available online at: https://doi.org/10.1007/s00029-019-0480-0.

[31] G. A. Kavvos. Modalities, cohesion, and information flow. *Proceedings of the ACM on Programming Languages*, 3:20:1–20:29, 2019.

[32] Neelakantan R. Krishnaswami. Higher-order functional reactive programming without spacetime leaks. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, page 221–232, New York, NY, USA, 2013. Association for Computing Machinery.

[33] Joachim Lambek and Philip J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, 1986.

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

[35] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal Universes in Models of Homotopy Type Theory. In H. Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 22:1–22:17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[36] Bassel Mannaa, Rasmus Ejlers Møgelberg, and Niccolò Veltri. Ticking clocks as dependent right adjoints: Denotational semantics for clocked type theory. *Logical Methods in Computer Science*, Volume 16, Issue 4, December 2020.

[37] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

[38] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., USA, 1989.

[39] Rasmus Ejlers Møgelberg. A type theory for productive coprogramming via guarded recursion. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, 2014.

[40] Rasmus Ejlers Møgelberg and Niccolò Veltri. Bisimulation as path type for guarded recursive types. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[41] Rasmus Ejlers Møgelberg and Marco Paviotti. Denotational semantics of recursive types in synthetic guarded domain theory. *Math. Struct. Comput. Sci.*, 29(3):465–510, 2019.

[42] H. Nakano. A modality for recursion. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*, pages 255–266. IEEE Computer Society, 2000.

[43] Ulf Norell. *Implementing Functional Generic Programming.* PhD thesis, 2004.

[44] Ian Orton and Andrew M. Pitts. Axioms for Modelling Cubical Type Theory in a Topos. *Logical Methods in Computer Science*, 14(4), 2018.

[45] FRANK PFENNING and ROWAN DAVIES. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

[46] Benjamin C. Pierce. *Types and Programming Languages.* The MIT Press, 1st edition, 2002.

[47] Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, 16(1), 2020.

[48] Dana S. Scott and Christopher S. Strachey. Toward a mathematical semantics for computer languages. 1971.

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

[50] Michael Shulman. All $(\infty, 1)$-toposes have strict univalent universes, 2019.

[51] Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '21, New York, NY, USA, 2021. ACM.

[52] Jonathan Sterling and Bas Spitters. Normalization by gluing for free λ-theories, 2018.

[53] The Coq Development Team. The coq proof assistant, October 2019.

[54] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* Institute for Advanced Study, 2013.

[55] Niccolò Veltri and Niels van der Weide. Guarded recursion in agda via sized types. volume 131, page 32:1–32:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, 2019.

[56] Vezzosi, Andrea. `agda-guarded`, 2021.

[57] Philip Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*, pages 347–359. ACM Press, 1989.

[58] Hongwei Xi and Dengping Zhu. To memory safety through proofs. *ArXiv*, abs/1810.12190, 2018.

2. GREATEST HITS: HIGHER INDUCTIVE TYPES IN COINDUCTIVE DEFINITIONS VIA INDUCTION UNDER CLOCKS

Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks

Magnus Baunsgaard Kristensen
Computer Science
IT University of Copenhagen
Copenhagen, Denmark
mbkr@itu.dk

Rasmus Ejlers Møgelberg
Computer Science
IT University of Copenhagen
Copenhagen, Denmark
mogel@itu.dk

Andrea Vezzosi
Computer Science
IT University of Copenhagen
Copenhagen, Denmark
vezzosi.ndr@gmail.com

Abstract

We present Clocked Cubical Type Theory, the first type theory combining multi-clocked guarded recursion with the features of Cubical Type Theory. Guarded recursion is an abstract form of step-indexing, which can be used for construction of advanced programming language models. In its multi-clocked version, it can also be used for coinductive programming and reasoning, encoding productivity in types. Combining this with Higher Inductive Types (HITs) this extends to coinductive types that are traditionally hard to represent in type theory, such as the type of finitely branching labelled transition systems.

Among our technical contributions is a new principle of *induction under clocks*, providing computational contents to one of the main axioms required for encoding coinductive types. This principle is verified using a denotational semantics in a presheaf model.

ACM Reference Format:
Magnus Baunsgaard Kristensen, Rasmus Ejlers Møgelberg, and Andrea Vezzosi. 2022. Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 31 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Homotopy type theory [49] is the extension of Martin-Löf type theory [36] with the univalence axiom and higher inductive types. It can be seen as a foundation for mathematics, allowing for synthetic approaches to mathematics, as exemplified by synthetic homotopy theory, as well as more direct formalisations of mathematics, closer to the traditional set-theory based developments. Higher Inductive Types (HITs) play a key role in both approaches: In synthetic homotopy theory these are used to described spaces such as the spheres and the torus, and in the more direct formalisations, HITs express free structures and quotients, both of which have traditionally been hard to represent and use in type theory.

The kind of mathematics that is particularly important to the LICS community often involves reasoning about recursion, and in particular programming with and reasoning about coinductive types. Doing this in type theory has historically been difficult for a number of reasons. One is that the definitions of the functors of interest in coalgebra often involve constructions such as finite or countable powerset (for modelling non-determinism), or quotients. All of these are known to be representable as HITs [15, 26, 49]. Another reason is that programming and reasoning about coinductive types involves productivity checking, which in most proof assistants is implemented as non-modular syntactic checks that can lead to considerable overhead for programmers [24]. A third reason is that bisimilarity, which is the natural notion of identity for coinductive types, rarely (at least provably) coincides with the built-in identity types of type theory, and so proofs cannot be transported along bisimilarity proofs.

Multi-clocked guarded recursion [6] allows for the productivity requirement for coinductive definitions to be encoded in types. The key ingredient in this is a modal operator $\triangleright$ (pronounced 'later') indexed by clocks κ, encoding a notion of time-steps in types. Guarded recursive types such as $\mathsf{Str}^\kappa \simeq \mathbb{N} \times \triangleright^\kappa \mathsf{Str}^\kappa$ are recursive types in which the recursion variable occurs only under a $\triangleright$. In the case of Str^κ the type equivalence above states that the head of a guarded stream is available now, but the tail only after a time step on clock κ. Guarded streams can be defined using a fixed point operator fix^κ of type $(\triangleright^\kappa A \to A) \to A$. In the case of Str^κ the delay in the domain of the input precisely captures the productivity requirement for programming with streams. Guarded recursive types can themselves be encoded as fixed points on a universe. Using quantification over clocks, the coinductive type of streams can be encoded as $\mathsf{Str} \stackrel{\mathsf{def}}{=} \forall\kappa.\mathsf{Str}^\kappa$.

Guarded recursion [42] can also be used as an abstract approach to step-indexing techniques [4, 5]. In particular, the type variables in guarded recursive types can also appear in negative positions, and so can be used to provide models of untyped lambda calculus or solve the advanced type equations needed for modelling higher-order store [10]

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and advanced notions separation logic [33]. A type theory combining guarded recursion with homotopy type theory will therefore provide a powerful framework, not just for coinductive reasoning but also for reasoning about recursion and advanced programming language features beyond the reach of traditional domain theory.

1.1 Clocked Cubical Type Theory

This paper presents Clocked Cubical Type Theory (CCTT), the first type theory to combine all the above mentioned features: Multi-clocked guarded recursion, HITs and univalence. Rather than basing this on the original formulation of homotopy type theory we build on Cubical Type Theory (CTT) [20], a variant of HoTT based on the cubical model of type theory, and implemented in the Cubical Agda proof assistant [53]. One reason for this choice is that the operational properties of CTT as well as the concrete implementation give hope for an implementation of our type theory. In fact we have already extended the implementation of Guarded Cubical Agda [52] to support most of the new constructions of CCTT. Another is that the path types of CTT make for a simple implementation of the extensionality principle for the $\triangleright^\kappa$ modality [9].

CCTT extends CTT with the constructions of Clocked Type Theory (CloTT) [7], a type theory for multi-clocked guarded recursion. CloTT uses a Fitch-style approach to programming with the $\triangleright$ modality. This means that elements of modal type are introduced by abstracting over tick variables $\alpha : \kappa$ and eliminated by application to ticks. One benefit of this is that let-expressions are avoided, and these do not interact well with dependent types in general. For example, one can define a dependently typed generalisation of applicative action [37] operator of type

$$\triangleright^\kappa(\textstyle\prod (x : A) .B) \to \prod (y : \triangleright^\kappa A) .\triangleright (\alpha : \kappa).B[y\,[\alpha]/x] \quad (1)$$

In this term the tick variable α should be thought of as evidence that time has passed, which is used in subterms like $y\,[\alpha]$ to access the element of type A delivered by $y : \triangleright^\kappa A$ in the next time step. CloTT has a single tick constant $\diamond$ that can be used for a controlled elimination of $\triangleright$. Bahr. et al. [7] define a strongly normalising operational semantics for CloTT, in which $\diamond$ unfolds fixed point operations. Since tick variables α can be substituted by $\diamond$, their identity is crucial for normalisation. For reasoning in the type theory, however, it is often necessary that $y\,[\alpha]$ does not depend on α up to path equality. This is referred to as the *tick irrelevance axiom*.

When adding such axioms to a univalent type theory one should be take care to avoid introducing extra structure that one will eventually need to reason about. Ideally, an axiom like tick irrelevance should state that the type of ticks is propositional, i.e., that any two elements can be identified, any two paths between elements can be identified, and so on for higher dimensions. In CCTT, there appears to be no such way of formulating the axiom, because ticks do not form a type. This paper shows how to obtain tick irrelevance by enriching the language of ticks with paths $\mathsf{tirr}\,u\,v$ between any pair of ticks u, v. This has the additional benefit that it allows for rules giving computational content to the axiom.

1.2 Induction under clocks

The encoding of coinductive streams from guarded streams mentioned above generalises to an encoding of coinductive solutions to type equations of the form $X \simeq F(X)$ for all functors F commuting with clock quantification in the sense that the canonical map

$$F(\forall\kappa.X) \to \forall\kappa.F(X) \quad (2)$$

is an equivalence of types. Note that this is a semantic condition, rather than the more standard grammars for coinductive types. For this to be useful, one must have a large supply of such functors, including functors formed using inductive and higher inductive types. This property has previously been obtained via axioms [6]. Here we formulate a principle of *induction under clocks*, which gives computational content to these axioms also in the case of higher inductive types.

Consider for example the finite powerset $\mathsf{P_f}$ which can be defined as a HIT given by constructors for singletons, union, equalities stating associativity, commutativity and idempotency of union as well as an axiom forcing $\mathsf{P_f}(X)$ to be a hset [26]. In this case induction under clocks states that to inhabit a family over an element of type $\forall\kappa.\mathsf{P_f}(X)$ it suffices to inductively describe the cases for elements of the form $\lambda\kappa.\mathsf{con}_i(p)$ for each constructor con_i for $\mathsf{P_f}$, i.e., for $\lambda\kappa.\{a\}, \lambda\kappa.x\,[\kappa] \cup y\,[\kappa]$ and for equality constructors such as $\lambda\kappa.\mathsf{idem}(x\,[\kappa], r)$, where idem implements idempotency of union. In these cases, x, y are assumed to be of type $\forall\kappa.\mathsf{P_f}(X)$. Using this, one can construct the map an inverse to (2) for $F = \mathsf{P_f}$, and also prove that both compositions are identities. As a consequence, the coinductive solution to $\mathsf{LTS} \simeq \mathsf{P_f}(A \times \mathsf{LTS})$ can be encoded as $\mathsf{LTS} \stackrel{\mathsf{def}}{=} \forall\kappa.\mathsf{LTS}^\kappa$ where $\mathsf{LTS}^\kappa \simeq \mathsf{P_f}(A \times \triangleright^\kappa \mathsf{LTS}^\kappa)$ is a guarded recursive type. LTS is the type of A-labelled finitely branching transition systems. We moreover prove that path equality for this type coincides with bisimilarity.

The results for LTS mentioned above hold for all types of labels A that are *clock-irrelevant*, meaning that the map $A \to \forall\kappa.A$ is an equivalence for κ fresh. In previous type theories for multi-clocked guarded recursion clock-irrelevance of all types has been taken as an axiom. Most types are clock-irrelevant, but ensuring this for universes require special care both syntactically and semantically [12]. In this paper we opt for a simpler solution, taking clock-irrelevance to be a side condition to the correctness of the encoding of coinductive types. Using the principle of induction under clocks we show that HITs constructed using clock-irrelevant types are themselves clock-irrelevant, an important step towards reintroducing clock irrelevance as an axiom.

1.3 Denotational semantics

Our type theory is justified by a denotational semantics in the form of a presheaf model. This model combines previous models of CTT [20, 43] and CloTT [35]. A key challenge for the construction of this model is the definition of a composition structure on the operation modelling $\triangleright$. We give a general theorem for defining composition structures for dependent right adjoints [19] which can be used to model other type theories combining Fitch-style modal operators [16] with Cubical Type Theory.

Overview of paper. The paper is organised as follows. We start by recalling Cubical Type Theory in section 2, and section 3 extends this to CCTT. The encoding of coinductive types using guarded recursion is recalled in section 4 which also details the example of labelled transition systems. The syntax and semantics for HITs in CCTT is presented in section 5, which also states the principle of induction under clocks. The denotational semantics of Clocked Cubical Type Theory is sketched in section 6. Finally, we discuss related work in section 7, and conclude and discuss future work in section 8. This submission is accompanied by an appendix of omitted proofs for reviewers.

2 Cubical Type Theory

Cubical Type Theory [20] is an extension of Martin-Löf type theory which gives a computational interpretation to extensionality principles like function extensionality and univalence. It does so by internalizing the notion from Homotopy Type Theory that equalities are paths, by explicitly representing them as maps from an interval object. The interval, $\mathbb{I}$, is not a type but contexts can contain $i : \mathbb{I}$ assumptions and there is a judgement, $\Gamma \vdash r : \mathbb{I}$ which specifies that r is built according to this grammar

$$r, s ::= 0 \mid 1 \mid i \mid 1 - r \mid r \wedge s \mid r \vee s$$

where i is taken from the assumptions in Γ. Equality of two interval elements, $\Gamma \vdash r \equiv s : \mathbb{I}$ corresponds to the laws of De Morgan algebras. A good intuition is to think of $\mathbb{I}$ as the real interval $[0, 1]$ with $r \wedge s$ and $r \vee s$ given by minimum and maximum operations.

Given $x, y : A$, we write $\mathsf{Path}_A(x, y)$ for the type of paths between x and y, which correspond to maps from $\mathbb{I}$ into A which are equal to x or y when applied to the endpoints $0, 1$ of the interval $\mathbb{I}$.

$$\frac{\Gamma, i : \mathbb{I} \vdash t : A}{\Gamma \vdash \lambda i.\, t : \mathsf{Path}_A(t[0/i], t[1/i])}$$

$$\frac{\Gamma \vdash p : \mathsf{Path}_A(a_0, a_1) \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash p\, r : A}$$

We sometimes write $=_A$ or simply $=$ as infix notation for path equality, and use $\equiv$ for judgemental equality. Path types support the β and η equalities familiar from function types, but we also have that a path applied to 0 or 1 reduces according to the endpoints specified in its type.

$$(\lambda i.\, t)\, r \equiv t[r/i] \qquad p \equiv (\lambda i.\, p\, i) \qquad p\, 0 \equiv a_0 \qquad p\, 1 \equiv a_1$$

The path witnessing reflexivity is defined as $\mathsf{refl}\, x \stackrel{\mathsf{def}}{=} \lambda i.\, x : \mathsf{Path}_A(x, x)$, given any $x : A$. It is also straightforward to provide a proof of function extensionality, as it is just a matter of reordering arguments:

$$\frac{\Gamma \vdash p : \Pi(x : A).\, \mathsf{Path}_B(f\, x, g\, x)}{\Gamma \vdash \lambda i.\, \lambda x.\, p\, x\, i : \mathsf{Path}_{\Pi(x:A).B}(f, g)}$$

Other constructions like transitivity, or more generally transport along path, require a further primitive operation called composition, which we now recall.

Define first the *face lattice* $\mathbb{F}$, as the free distributive lattice with generators $(i = 0)$ and $(i = 1)$, and satisfying the equality $(i = 0) \wedge (i = 1) = 0_{\mathbb{F}}$. Explicitly, elements $\Gamma \vdash \varphi : \mathbb{F}$ are given by the following grammar, where i ranges over the interval variables from Γ:

$$\varphi, \psi ::= 0_{\mathbb{F}} \mid 1_{\mathbb{F}} \mid (i = 0) \mid (i = 1) \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

Given $\Gamma \vdash \varphi : \mathbb{F}$ we can form the restricted context Γ, φ. Types and terms in context Γ, φ are called *partial*, matching the intuition that they are only defined for elements of Γ that satisfy the constraints specified by φ. For example while a type $i : \mathbb{I} \vdash A$ type corresponds to a whole line, a type $i : \mathbb{I}, (i = 0) \vee (i = 1) \vdash B$ type is merely two disconnected points. Given a partial element $\Gamma, \varphi \vdash u : A$ we write $\Gamma \vdash t : A[\varphi \mapsto u]$ to mean the conjunction of $\Gamma \vdash t : A$ and $\Gamma, \varphi \vdash t \equiv u : A$.

Composition is given by the following typing rule:

$$\frac{\Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I} \vdash A \text{ type} \qquad \Gamma, \varphi, i : \mathbb{I} \vdash u : A \qquad \Gamma \vdash u_0 : A[0/i][\varphi \mapsto u[0/i]]}{\Gamma \vdash \mathsf{comp}^i_A\, [\varphi \mapsto u]\, u_0 : A[1/i][\varphi \mapsto u[1/i]]}$$

the inputs u and u_0 represent the sides and the base of an open box, while composition gives us the lid, i.e. the side opposite to the base. The behaviour of composition is specified by u whenever φ is satified, and otherwise depends on the type A, consequently when we introduce new type formers we will also give a corresponding equation for comp. As an example we can prove transitivity for paths with the following term:

$$\frac{\Gamma \vdash p : \mathsf{Path}_A(x, y) \qquad \Gamma \vdash q : \mathsf{Path}_A(y, z)}{\Gamma \vdash \lambda i.\, \mathsf{comp}^j\, [i = 0 \mapsto x, i = 1 \mapsto q\, j]\, (p\, i) : \mathsf{Path}_A(x, z)}$$

The standalone syntax for partial elements is as a list of faces and terms $[\varphi_1\, u_1, \ldots, \varphi_n\, u_n]$, but we will write $[\vee_i \varphi_1 \mapsto [\varphi_1\, u_1, \ldots, \varphi_n\, u_n]]$ as $[\varphi_1 \mapsto u_1, \ldots, \varphi_n \mapsto u_n]$. We refer to Cohen et al. [20] for more details on partial elements, composition, and the *glueing construction*. The latter is what allows to derive the univalence principle as a theorem, by turning a partial equivalence into a total one. Writing $A \simeq B$ for the type stating that A and B are equivalent [49], univalence

states that the canonical map of type $A =_{\mathsf{U}} B \to A \simeq B$ is itself an equivalence for any types $A, B : \mathsf{U}$. We will use univalence in the rest of the paper, but we will not need to refer to a specific implementation, so we omit the details in this brief overview.

From Homotopy Type Theory [49] we also recall the notion of homotopy *proposition* which are types for which any two elements are path equal, and homotopy *set* which are types whose path type is a proposition. Given any type A its propositional truncation $\|A\|_0$ is the least proposition that admits a map from A. It can be defined as an Higher Inductive Type (HIT), which are inductive types defined by providing generators for both the type itself and its path equality. Another example of a HIT is the finite powerset [26] of a type A, which has generators for the empty set $\emptyset$, singletons $\{-\}$, union $\cup$, and equations stating that properties like associativity and commutativity of $\cup$, and finally constructors forcing the powerset to be a (homotopy) set. We will discuss HITs in more detail in section 5.

3 Clocked Cubical Type Theory

CCTT extends CTT with the constructions of CloTT [7, 35], a type theory with Nakano style guarded recursion, multiple clocks and ticks. This means that the delay type operator is associated with clocks: $\triangleright^\kappa A$ is the type of data of type A which is available in the next time step on clock κ. There are no operations on clocks, they can only be assumed by placing assumptions such as $\kappa : \mathsf{clock}$ in the context, and abstracted to form elements of type $\forall\kappa.A$. The status of clock is similar to $\mathbb{I}$ in the sense that it is not a type, and so, e.g., clock $\to$ clock is not a wellformed type. In some examples below it is convenient to assume a single clock constant κ_0, and this can be achieved by a precompilation adding this to the context. Similarly to function types, path equality in $\forall\kappa.A$ is equivalent to pointwise equality, and clock quantification preserves truncation levels.

Ticks are evidence that time has passed on a given clock. Tick assumptions in a context separate a judgement such as $\Gamma, \alpha : \kappa, \Gamma' \vdash t : A$ into a part (Γ) of assumptions arriving before the time tick α on the clock κ and the rest of the judgement that happens after. The introduction rule for $\triangleright\,(\alpha : \kappa).A$ in Figure 1 can therefore be read as stating that if t has type A one time step after the assumptions in Γ, then $\lambda(\alpha:\kappa).t$ is delayed data of type A. Because terms can appear in types, α can appear in A, and must therefore be mentioned in the type of $\lambda(\alpha:\kappa).t$. In many cases, however, it does not, and we will then write simply $\triangleright^\kappa A$ for $\triangleright\,(\alpha:\kappa).A$.

Elimination for $\triangleright\,(\alpha:\kappa).A$ is by application to ticks. There are two forms of ticks: simple ticks and forcing ticks. The judgement for simple ticks $\Gamma \vdash u : \kappa \rightsquigarrow \Gamma'$ states that u is a tick on the clock κ in context Γ with *residual context* Γ'. The residual context consists of the assumptions that were available before the tick u. In the case of a tick variable α, these are the assumptions to the left of α as well as the *timeless* assumptions to the right of α. Timeless assumptions are interval and clock assumptions ($i : \mathbb{I}$ and $\kappa : \mathsf{clock}$) as well as all faces, and $\mathsf{TL}(\Gamma)$ computes the timeless assumptions of Γ. The assumption $\Gamma'' \sqsubseteq \Gamma$ defined by the rule

$$\Gamma, \mathsf{TL}(\Gamma') \sqsubseteq \Gamma, \Gamma'$$

allows for further trimming of the residual context. The term $t\,_{\Gamma'}[u]$ applies t to the simple tick u. The assumption that t is well-typed in the residual context Γ' should be read as stating that it has the type $\triangleright\,(\alpha:\kappa).A$ before the tick u, and can therefore be opened after the tick u to produce something of type $A[u/\alpha]$. We will often omit the residual context Γ' from the term, writing simply $t\,[u]$ for $t\,_{\Gamma'}[u]$. Different choices of Γ' give the same term up to judgemental equality.

As basic examples of programming with simple ticks we mention the dependently typed generalisation of applicative action (1) mentioned in the introduction, and the unit

$$\lambda x.\lambda(\alpha:\kappa).x : A \to \triangleright^\kappa A \tag{3}$$

Note that this uses the rule for variables which allows these to be introduced from anywhere in the context, also across ticks. Some Fitch style modal type theories do not allow such introductions [16, 19]. The timelessness of the interval is needed to type the extensionality principle

$$\mathsf{Path}_{\triangleright^\kappa A}(x, y) \simeq \triangleright\,(\alpha:\kappa).\mathsf{Path}_A(x\,[\alpha], y\,[\alpha])$$

where the left to right direction $\lambda p.\lambda(\alpha:\kappa).\lambda i.p\;i\;[\alpha]$ requires $p\;i\;[\alpha]$ to be welltyped in a context where i occurs after α.

3.1 Forcing ticks

Unrestricted elimination of $\triangleright$ is unsound, since any type A with a map $\triangleright^\kappa A \to A$ can be inhabited using fixed points. However, it is safe to eliminate $\triangleright^\kappa$ in contexts of the form $\Gamma, \kappa : \mathsf{clock}$, i.e., where no variables mention κ in their type. To close such an elimination rule under substitutions, the clock variable κ must be abstracted in the term.

Application to forcing ticks does exactly that: If κ' is a clock in Γ then $\Gamma \vdash (\kappa', \diamond) \rightsquigarrow \Gamma$ is a forcing tick and so if $\Gamma, \kappa : \mathsf{clock} \vdash t : \triangleright(\alpha : \kappa).A$ then

$$\Gamma \vdash (\kappa.t)\,_\Gamma[(\kappa', \diamond)] : A[(\diamond : \kappa')/(\alpha:\kappa)]$$

This construction binds κ in t, and $A[(\diamond : \kappa')/(\alpha:\kappa)]$ uses a special simultaneous substitution of the pair $(\diamond, \kappa')$ for the pair (α, κ). In particular $[(\diamond : \kappa')/(\alpha:\kappa)]$ commutes as expected with type and term formers, replacing each of α and κ with $\diamond$ and κ' as intended, but it will also have to turn a simple tick application like $t\,[\alpha]$ into a forcing tick application $(\kappa.t)\,[(\kappa', \diamond)]$ to preserve typing. This substitution operation was originally described by Mannaa et al. [35], see Appendix A.2 for a description of substitution for our theory.

For example, forcing ticks can be used to define the force operator of Atkey and McBride [6]

$$\mathsf{force} \stackrel{\mathsf{def}}{=} \lambda x.\lambda\kappa.(\kappa.x\,[\kappa])\,[(\kappa, \diamond)] : \forall\kappa.\,\triangleright^\kappa A \to \forall\kappa.A \tag{4}$$

Context and type formation rules

$$\frac{\Gamma \vdash \qquad \kappa \notin \Gamma}{\Gamma, \kappa : \mathsf{clock} \vdash} \qquad \frac{\Gamma \vdash \kappa : \mathsf{clock} \qquad \alpha \notin \Gamma}{\Gamma, \alpha : \kappa \vdash} \qquad \frac{\Gamma, \alpha : \kappa \vdash A \ \mathsf{type}}{\Gamma \vdash \triangleright\,(\alpha{:}\kappa).A \ \mathsf{type}} \qquad \frac{\Gamma, \kappa : \mathsf{clock} \vdash A \ \mathsf{type}}{\Gamma \vdash \forall \kappa.A \ \mathsf{type}}$$

Typing rules

$$\frac{\Gamma, \kappa : \mathsf{clock} \vdash t : A}{\Gamma \vdash \lambda\kappa.t : \forall\kappa.A} \qquad \frac{\Gamma \vdash t : \forall\kappa.A \qquad \Gamma \vdash \kappa' : \mathsf{clock}}{\Gamma \vdash t[\kappa'] : A[\kappa'/\kappa]} \qquad \frac{\Gamma, \alpha : \kappa \vdash t : A}{\Gamma \vdash \lambda(\alpha{:}\kappa).t : \triangleright\,(\alpha{:}\kappa).A}$$

$$\frac{\Gamma \vdash u : \kappa \rightsquigarrow \Gamma' \qquad \Gamma' \vdash t : \triangleright\,(\alpha{:}\kappa).A}{\Gamma \vdash t\,{}_{\Gamma'}[u] : A[u/\alpha]} \qquad \frac{\Gamma \vdash (\kappa', u) \rightsquigarrow \Gamma' \qquad \Gamma', \kappa : \mathsf{clock} \vdash t : \triangleright(\alpha : \kappa).A}{\Gamma \vdash (\kappa.t)\,{}_{\Gamma'}[(\kappa', u)] : A[(u : \kappa')/(\alpha{:}\kappa)]} \qquad \frac{\Gamma \vdash t : \triangleright^{\kappa} A \to A}{\Gamma \vdash \mathsf{dfix}^{\kappa}\, t : \triangleright^{\kappa} A}$$

$$\frac{\Gamma \vdash t : \triangleright^{\kappa} A \to A}{\Gamma \vdash \mathsf{pfix}^{\kappa}\, t : \triangleright\,(\alpha{:}\kappa).\mathsf{Path}_A((\mathsf{dfix}^{\kappa} t)\,[\alpha], t(\mathsf{dfix}^{\kappa} t))} \qquad \frac{\Gamma, \alpha : \kappa \vdash A : \mathsf{U}}{\Gamma \vdash \overline{\triangleright}\,(\alpha{:}\kappa).A : \mathsf{U}} \qquad \frac{\Gamma, \kappa : \mathsf{clock} \vdash A : \mathsf{U}}{\Gamma \vdash \overline{\forall}\kappa.A : \mathsf{U}}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t : B} \qquad \frac{\kappa : \mathsf{clock} \in \Gamma}{\Gamma \vdash \kappa : \mathsf{clock}} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

Simple ticks

$$\frac{\Gamma'' \sqsubseteq \Gamma}{\Gamma, \alpha : \kappa, \Gamma' \vdash \alpha : \kappa \rightsquigarrow \Gamma'', \mathsf{TL}(\Gamma')} \qquad \frac{\Gamma \vdash u : \kappa \rightsquigarrow \Gamma' \qquad \Gamma \vdash v : \kappa \rightsquigarrow \Gamma' \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash \mathsf{tirr}(u, v, r) : \kappa \rightsquigarrow \Gamma'}$$

Forcing ticks

$$\frac{\Gamma' \sqsubseteq \Gamma \qquad \Gamma' \vdash \kappa' : \mathsf{clock}}{\Gamma \vdash (\kappa', \diamond) \rightsquigarrow \Gamma'} \qquad \frac{\Gamma'' \sqsubseteq \Gamma}{\Gamma, \alpha : \kappa', \Gamma' \vdash (\kappa', \alpha) \rightsquigarrow \Gamma'', \mathsf{TL}(\Gamma')}$$

$$\frac{\Gamma \vdash (\kappa', u) \rightsquigarrow \Gamma' \qquad \Gamma \vdash (\kappa', v) \rightsquigarrow \Gamma' \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash (\kappa', \mathsf{tirr}(u, v, r)) \rightsquigarrow \Gamma'}$$

Timeless assumptions

$$\mathsf{TL}(\Gamma, x : A) = \mathsf{TL}(\Gamma) \qquad \mathsf{TL}(\Gamma, \alpha : \kappa) = \mathsf{TL}(\Gamma) \qquad \mathsf{TL}(\Gamma, i : \mathbb{I}) = \mathsf{TL}(\Gamma), i : \mathbb{I}$$
$$\mathsf{TL}(\Gamma, \kappa : \mathsf{clock}) = \mathsf{TL}(\Gamma), \kappa : \mathsf{clock} \qquad \mathsf{TL}(\Gamma, \varphi) = \mathsf{TL}(\Gamma), \varphi \qquad \mathsf{TL}(\cdot) = \cdot$$

Figure 1. Selected typing rules for CCTT. The rules for ticks use the relation defined as $\Gamma, \mathsf{TL}(\Gamma') \sqsubseteq \Gamma, \Gamma'$

Judgemental equalities on terms

$$t\,{}_{\Gamma}[u] \equiv t\,{}_{\Gamma'}[u] \qquad (\kappa.t)\,{}_{\Gamma}[(\kappa, u)] \equiv (\kappa.t)\,{}_{\Gamma'}[(\kappa, u)] \qquad (\lambda\kappa.t)[\kappa'] \equiv t[\kappa'/\kappa]$$
$$\lambda\kappa.(t[\kappa]) \equiv t \qquad (\lambda(\alpha{:}\kappa).t)\,[u] \equiv t\,[u/\alpha] \qquad \lambda(\alpha{:}\kappa).(t\,[\alpha]) \equiv t$$
$$(\kappa.\lambda(\alpha{:}\kappa).t)\,[(\kappa', u)] \equiv t[(u : \kappa')/(\alpha{:}\kappa)] \qquad (\kappa.t)\,[(\kappa', u)] \equiv (t[\kappa'/\kappa])\,[u] \qquad (\kappa.\mathsf{dfix}^{\kappa}\, t)\,[(\kappa', \diamond)] \equiv t(\mathsf{dfix}^{\kappa}\, t)[\kappa'/\kappa]$$
$$\mathsf{El}\,(\overline{\forall}\kappa.A) \equiv \forall\kappa.\mathsf{El}\,(A) \qquad (\kappa.\mathsf{pfix}^{\kappa}\, t)\,[(\kappa', \diamond)]\, r \equiv t(\mathsf{dfix}^{\kappa}\, t)[\kappa'/\kappa] \qquad \mathsf{El}\,(\overline{\triangleright}\,(\alpha{:}\kappa).A) \equiv \triangleright\,(\alpha{:}\kappa).\mathsf{El}\,(A)$$

Judgemental equalities on ticks

$$\mathsf{tirr}(u, v, 0) \equiv u \qquad \mathsf{tirr}(\diamond, \diamond, r) \equiv \diamond \qquad \mathsf{tirr}(u, v, 1) \equiv v$$

Figure 2. Selected judgemental equality rules of CCTT. The three η rules are subject to the standard conditions of κ and α, respectively, not appearing in t. As a consequence of the first two rules, the residual context is omitted for tick application in the rules below. The 8th axiom listed assumes that $\diamond$ does not appear in u, so that u can be considered a simple tick.

which is used in the encoding of coinductive types. Replacing the delayed substitution of κ' for κ in $(\kappa.t)\,\Gamma[(\kappa', \diamond)]$ gives the elimination rule used by Bahr. et al. [7]. Here we opt for an explicit substitution because it is easier to type check and give semantics to, which is why Mannaa et al. [35] made a similar choice. Bahr. et al. [7] designed their application rule for $\diamond$ to obtain a normalisation result for their theory. We expect that using the present rule, one can prove the same property for CCTT.

3.2 Tick irrelevance

Since fixed points unfold when applied to $\diamond$, the identity of ticks is crucial for the operational properties of Clocked Type Theory, in particular for ensuring strong normalisation. However, on the extensional level the identity of ticks is irrelevant, as stated by the *tick irrelevance axiom*

$$\mathsf{tirr}^\kappa : \Pi(x : \triangleright^\kappa A).\triangleright\,(\alpha:\kappa).\triangleright\,(\beta:\kappa).\mathsf{Path}_A(x\,[\alpha], x\,[\beta]) \quad (5)$$

This can be inhabited in CCTT using the novel construction tirr building a path between any pair of ticks by defining

$$\mathsf{tirr}^\kappa(t) \stackrel{\text{def}}{=} \lambda(\alpha:\kappa).\lambda(\beta:\kappa).\lambda i.t\,[\mathsf{tirr}(\alpha, \beta, i)]$$

Similar axioms can be constructed for forcing tick applications. One important application of tirr is in showing that force (4) is a type equivalence with inverse $\mathsf{force}^{-1} \stackrel{\text{def}}{=} \lambda x.\lambda\kappa.\lambda(\alpha:\kappa).x\,[\kappa]$: If $x : \forall\kappa.\triangleright^\kappa A$, then

$$\begin{aligned}\mathsf{force}^{-1}(\mathsf{force}\,x) &\equiv \lambda\kappa.\lambda(\alpha:\kappa).(\kappa.x\,[\kappa])\,[(\kappa, \diamond)]\\ &= \lambda\kappa.\lambda(\alpha:\kappa).(\kappa.x\,[\kappa])\,[(\kappa, \alpha)]\\ &\equiv \lambda\kappa.\lambda(\alpha:\kappa).x\,[\kappa]\,[\alpha] \equiv x\end{aligned}$$

where the path equality is witnessed by

$$\lambda i.\lambda\kappa.\lambda(\alpha:\kappa).(\kappa.x\,[\kappa])\,[(\kappa, \mathsf{tirr}(\diamond, \alpha, i))].$$

The ability of tirr to form a path between any two ticks allows us to prove coherence laws between different uses of tick irrelevance. For example we can construct fillers for boxes whose boundary is given by tick applications as in the lemma below.

Lemma 3.1. *Let* $\Gamma' \vdash t : \triangleright\,(\alpha:\kappa).A$ *and* $\Gamma, i_0, \ldots, i_n : \mathbb{I}, \varphi \vdash u : \kappa \rightsquigarrow \Gamma'$ *where* $\varphi = \bigvee_{k,b}(i_k = b)$ *for* $k = 0, \ldots, n$ *and* $b = 0, 1$. *Then we have* $\Gamma, i_0, \ldots, i_n : \mathbb{I} \vdash B[\varphi \mapsto A[u/\alpha]]$ type *and a term* $\Gamma, i_0, \ldots, i_n : \mathbb{I} \vdash \mathsf{filler}(t, u) : B[\varphi \mapsto t\,[u]]$.

For example, the lemma constructs a filler of the diagram below, thus constructing a path from the composition of $\mathsf{tirr}^\kappa(t)\,[\alpha]\,[\beta]$ and $\mathsf{tirr}^\kappa(t)\,[\beta]\,[\gamma]$ to $\mathsf{tirr}^\kappa(t)\,[\alpha]\,[\gamma]$,

$$\begin{array}{ccc} A & \xrightarrow{\mathsf{tirr}^\kappa(t)\,[\beta]\,[\gamma]} & A \\ {\scriptstyle \mathsf{tirr}^\kappa(t)\,[\alpha]\,[\beta]}\big\downarrow & & \big\downarrow{\scriptstyle \mathsf{tirr}^\kappa(t)\,[\alpha]\,[\gamma]} \\ A & \xrightarrow{t\,[\alpha]} & A \end{array}$$

The rules for judgemental equality (Figure 2) include standard β and η-rules for functions, clock abstraction and tick abstraction. Note that there are two β equalities for tick application, one for each kind of ticks. There is also a rule stating that if a forcing tick does not contain $\diamond$, application to it reduces to application to a simple tick. One consequence of this is that an η-rule for forcing tick application can be derived as follows

$$\lambda(\alpha:\kappa').((\kappa.t)\,[(\alpha, \kappa')]) \equiv \lambda(\alpha:\kappa').(t[\kappa'/\kappa]\,[\alpha]) \equiv t[\kappa'/\kappa]$$

Although we do not prove canonicity for our type theory, we set up equalities that we believe are necessary to compute canonical forms in contexts of free clock and interval variables, but not free tick variables. For example, $\mathsf{tirr}(\diamond, \diamond, r) = \diamond$ ensures that in a context with no tick variables all ticks are equal to $\diamond$.

3.3 Fixed points

The operator $\mathsf{dfix}^\kappa : (\triangleright^\kappa A \to A) \to \triangleright^\kappa A$ computes fixed points of productive functions. Using this, one can construct Nakano's fixed point operator as

$$\mathsf{fix}^\kappa \stackrel{\text{def}}{=} \lambda f.f(\mathsf{dfix}^\kappa f) : (\triangleright^\kappa A \to A) \to A \quad (6)$$

To ensure termination, fixed points do not unfold judgementally, except when applied to $\diamond$. As in Guarded Cubical Type Theory [9] we add a path pfix^κ between a fixed point and its unfolding. Note that the right hand endpoint of $\mathsf{pfix}^\kappa t$ is $\mathsf{fix}^\kappa t$ and so,

$$\mathsf{fix}^\kappa t \equiv t(\lambda(\alpha:\kappa).\mathsf{dfix}^\kappa t\,[\alpha]) = t(\lambda(\alpha:\kappa).\mathsf{fix}^\kappa t) \quad (7)$$

With this we can prove that fixed points are unique, in fact the following stronger statement is true.

Lemma 3.2. *The type* $\Sigma(x : A).\mathsf{Path}_A(x, f\,(\lambda(\alpha:\kappa).x))$ *is contractible for every* $f : \triangleright^\kappa A \to A$.

By lemma 3.2 the pair $(\mathsf{dfix}^\kappa t, \mathsf{pfix}^\kappa t)$ is uniquely determined up to path, as it witnesses that $\mathsf{dfix}^\kappa t$ is the fixed point of $\lambda x.\lambda(\_:\kappa).t\,x$. Moreover both dfix^κ and pfix^κ unfold when applied to $\diamond$, which means they will not be stuck terms in a context with no tick variables.

The principles above can be used to encode guarded recursive types as fixed points on the universe. For example, assuming codes for products and natural numbers, we define

$$\overline{\mathsf{Str}^\kappa}(\mathbb{N}) \stackrel{\text{def}}{=} \mathsf{fix}^\kappa(\lambda X.\overline{\mathbb{N}}\overline{\times}\overline{\triangleright}\,(\alpha:\kappa).X\,[\alpha])$$

and then by (7) $\overline{\mathsf{Str}^\kappa}(\mathbb{N}) = \overline{\mathbb{N}}\overline{\times}\overline{\triangleright}\,(\alpha:\kappa).\overline{\mathsf{Str}^\kappa}(\mathbb{N})$ so, defining $\mathsf{Str}^\kappa(\mathbb{N}) \stackrel{\text{def}}{=} \mathsf{El}\,(\overline{\mathsf{Str}^\kappa}(\mathbb{N}))$ gives the type equivalence

$$\mathsf{Str}^\kappa(\mathbb{N}) \simeq \mathbb{N} \times \triangleright^\kappa \mathsf{Str}^\kappa(\mathbb{N}) \quad (8)$$

Note that we do not assume the *clock irrelevance axiom*

$$\frac{\Gamma \vdash t : \forall\kappa.A \qquad \kappa \notin \mathsf{fc}(A)}{\Gamma \vdash \mathsf{cirr}^\kappa t : \forall\kappa'.\forall\kappa''.\mathsf{Path}_A(t[\kappa'], t[\kappa''])} \quad (9)$$

often assumed in type theories with multi-clock guarded recursion [7, 11]. Instead we will use the following definition.

Definition 3.3. A type A is *clock-irrelevant* if the canonical map $A \to \forall\kappa.A$ (for κ fresh) is an equivalence.

If A is clock-irrelevant it satisfies axiom (9), and assuming a clock constant the two statements are equivalent. Clock irrelevance is needed for the correctness of encodings of coinductive types. For example, using (8) one can prove

$$\forall\kappa.\mathsf{Str}^\kappa(\mathbb{N}) \simeq (\forall\kappa.\mathbb{N}) \times \forall\kappa.(\triangleright^\kappa \mathsf{Str}^\kappa(\mathbb{N})) \simeq \mathbb{N} \times \forall\kappa.\mathsf{Str}^\kappa(\mathbb{N})$$

assuming $\mathbb{N}$ clock-irrelevant, and using that force is an equivalence. In this paper, rather than assuming all types clock irrelevant, we will prove clock-irrelevance for a large class of types and use this to construct final coalgebras for a correspondingly large class of functors.

4 Encoding coinductive types

In the previous section we saw that the type $\forall\kappa.\mathsf{Str}^\kappa(\mathbb{N})$ satisfies the type equivalence expected of a type of streams. In this section we prove that (assuming $\mathbb{N}$ clock-irrelevant) this type also satisfies the universal property characterising streams, as a special case of a more general property. In Section 5 we will show that $\mathbb{N}$ is indeed clock-irrelevant (as a special case of Theorem 5.4) as are a large collection of inductive and higher inductive types.

Here we essentially adapt the final coalgebra construction from Møgelberg [38]. Given a type I consider the category[1] $I \to \mathsf{U}$ of I-indexed types whose type of objects is $I \to \mathsf{U}$ and whose morphisms from X to Y is the type

$$X \to Y \stackrel{\text{def}}{=} \Pi(i : I).\, \mathsf{El}\,(X\,i) \to \mathsf{El}\,(Y\,i)$$

An *I-indexed endofunctor* is an endofunctor on $I \to \mathsf{U}$.

Let F be an I-indexed endofunctor, an *F-coalgebra* is a pair (X, f) such that $X : I \to \mathsf{U}$ and $f : X \to FX$. We say an F-coalgebra (Y, g) is *final* if for all coalgebras (X, f) the following type is contractible

$$\Sigma(h : X \to Y).\, g \circ h = F(h) \circ f$$

Using contractibility we address the fact that there can be more than one proof of path equality, as is also done by Ahrens et al. [3].

If $X : \forall\kappa.\ (I \to \mathsf{U})$ write $\forall\kappa.X$ for $\lambda(i : I).\overline{\forall}\kappa.(X\ [\kappa]\ i)$.

Definition 4.1. Say that an I-indexed endofunctor F *commutes with clock quantification* if for all families X the canonical map $\mathsf{can}_F : F\,(\forall\kappa.X\ [\kappa]) \to \forall\kappa.\,F\,(X\ [\kappa])$ is a pointwise equivalence.

For example, if $A : \mathsf{U}$ the constant functor to A commutes with clock quantification if and only if A is clock-irrelevant.

Lemma 4.2. *The collection of endofunctors commuting with clock quantification is closed under composition, pointwise product, pointwise Π, pointwise Σ over clock irrelevant types, and pointwise universal quantification over clocks. If F commutes with clock quantification then the guarded recursive type X satisfying $X \simeq F(\triangleright^\kappa X)$ is clock irrelevant. Any path type of a clock irrelevant type is clock irrelevant.*

The following theorem states that all such endofunctors have final coalgebras. The idea of this originates with Atkey and McBride [6], and our proof is an adaptation of the proof by Møgelberg [38] of a similar statement in extensional type theory. It refers to the endofunctor $\triangleright^\kappa$, defined as the pointwise extension of the functor $\mathsf{U} \to \mathsf{U}$ mapping A to $\overline{\triangleright}\,(\alpha : \kappa).A$ for a fresh α.

Theorem 4.3. *Let F be an I-indexed endofunctor which commutes with clock quantification, and let $\nu^\kappa(F)$ be the guarded recursive type satisfying $\nu^\kappa(F) \simeq F(\triangleright^\kappa(\nu^\kappa(F))$. Then $\nu(F) \stackrel{def}{=} \forall\kappa.\nu^\kappa(F)$ has a final F-coalgebra structure.*

Example 4.4. In Section 5 we will prove that $\mathbb{N}$ is clock-irrelevant, and hence the functor $F(X) = \mathbb{N} \times X$ commutes with clock quantification. Theorem 4.3 then states correctness of the encoding of streams as $\forall\kappa.\mathsf{Str}^\kappa(\mathbb{N})$.

Example 4.5. Consider the functor $F(X) = \mathsf{P_f}(A \times X)$, where $\mathsf{P_f}$ is the finite powerset functor defined as a HIT [26], and A is assumed to be a clock-irrelevant set. In Section 5 we will prove that $\mathsf{P_f}$ commutes with clock quantification, which implies that F does the same. Theorem 4.3 then gives an encoding of the final coalgebra for F. This type plays an important role in automata theory as it describes the finitely branching A-labelled transition systems. Let LTS^κ denote fixed point for $F \circ \triangleright^\kappa$, let $\mathsf{LTS} \stackrel{\text{def}}{=} \forall\kappa.\mathsf{LTS}^\kappa$ be the coinductive type and let $\mathsf{ufld} : \mathsf{LTS} \to \mathsf{P_f}(A \times \mathsf{LTS})$ be the final coalgebra structure.

Suppose now $x, y : \mathsf{LTS}$ and $R : \mathsf{LTS} \times \mathsf{LTS} \to \mathsf{U}$. Define

$$\mathsf{Bisim}_\forall(R)(x, y) = \mathsf{Sim}_\forall(R)(x, y) \times \mathsf{Sim}_\forall(R)(y, x)$$

where $\mathsf{Sim}_\forall(R)(x, y)$ is

$$\Pi x', a.(a, x') \in \mathsf{ufld}(x) \to \exists y'.((a, y') \in \mathsf{ufld}(y)) \times R(x', y')$$

By Lemma 4.2, LTS is clock irrelevant, and an easy induction on $X : \mathsf{P_f}(A \times \mathsf{LTS})$ shows that the predicate $(a, x') \in X$ is clock irrelevant. In Section 5 we will prove that propositional truncation commutes with clock quantification, and since $\exists$ in homotopy type theory is defined as the composition of truncation and Σ [49], putting all this together shows that $\mathsf{Bisim}_\forall(-)$ defines a $\mathsf{LTS} \times \mathsf{LTS}$-indexed endofunctor commuting with clock quantification, and so Theorem 4.3 gives an encoding of bisimilarity as a coinductive type.

Møgelberg and Veltri [40] prove that path equality coincides with bisimilarity for guarded recursive types. Using their results we can prove the following.

Theorem 4.6. *Two elements of the type* LTS *from Example 4.5 are bisimilar if and only if they are path equal. Since both these are propositions, they are equivalent as types.*

[1] Here we mean the naive internal notion where the type of arrows is not necessarily truncated, but we also do not require higher coherences. Same for functor below.

$$\frac{}{\vdash \cdot \square} \qquad \frac{\vdash \Psi \ \square}{\vdash \Psi, i : \mathbb{I} \ \square} \qquad \frac{\Gamma_0 \vdash}{\Gamma_0 \vdash \cdot \ \mathsf{tel}}$$

$$\frac{\Gamma_0 \vdash \Gamma \ \mathsf{tel} \qquad \Gamma_0, \Gamma \vdash A \ \mathsf{type}}{\Gamma_0 \vdash \Gamma, x : A \ \mathsf{tel}}$$

Figure 3. Telescope judgements.

We leave it to future work to generalise these results to a general statement about bisimilarity for coinductive types.

5 Higher Inductive Types

We now extend CCTT with higher inductive types, adapting a schema for these from Cavallo and Harper [13] to our version of CTT. For simplicity we exclude indexed families, but the schema is still general enough to cover examples like spheres, pushouts, W-suspensions [46], (higher) truncations, as well as finite and countable powersets. We first present the schema and the introduction rules for HITs, then (subsection 5.2) we present our principle of induction under clocks, which generalises the elimination rule for HITs.

5.1 Introduction and formation

The judgements of Figure 4 capture declarations of the form

$$\begin{array}{l}\mathsf{data}\ \mathsf{H}\ (\delta : \Delta)\ \mathsf{where}\\ \quad\vdots\\ \quad \ell_i : (\gamma : \Gamma) \to (\overline{x : \overline{\Xi \to \mathsf{H}\,\delta}}) \to (\overline{i} : \Psi) \to \mathsf{H}\,\delta[\varphi \mapsto e]\\ \quad\vdots\end{array}$$

which list constructors and their types for a new datatype H, taking parameters in the telescope Δ. On top of the declared constructors every HIT has an introduction form for *homogeneous composition*, written $\mathsf{hcomp}^i_{\mathsf{H}\,\delta}\,[\varphi \mapsto u]\,u_0$, where δ is not allowed to depend on i, as well as a transport operation. Following Coquand et al. [21], the composition structure for H is given by combining these two operations. We refer to appendix A.5 for further details, but just recall that from the homogeneous composition operator one can derive a *homogeneous filling* operator $\mathsf{hfill}^i_A\,[\varphi \mapsto u]\,u_0 : \mathsf{Path}_A(u_0, \mathsf{hcomp}^i_A\,[\varphi \mapsto u]\,u_0)$ which provides a filler for the box specified by u and u_0 and closed by homogeneous composition [30].

A constructor for a HIT H is specified by a tuple $\Delta; \mathscr{K} \vdash (\Gamma, \overline{\Xi}, \Psi, \varphi, e)\ \mathsf{constr}$ where $\mathscr{K}$ lists the previously declared constructors. The telescope Γ lists the non-recursive arguments, and each Ξ in $\overline{\Xi}$ is the arity for a recursive argument. Path constructors further take a non-empty telescope of interval variables Ψ and have a boundary e specified as a partial element over the formula φ. The judgements for telescopes are given in Figure 3. Note that these imply that all of Δ, Γ, and Ξ only contain variables of proper types, i.e., no interval, face, clock or tick assumptions. Only the boundary is allowed to refer to H and the previous constructors from $\mathscr{K}$. We signify this by adding those as subscripts to the typing judgement for e. Cavallo and Harper require the components M of the boundary to conform to a dedicated typing judgement, which restricts their shape and makes it possible to correctly specify the inputs to the dependent eliminator for H. For conciseness we replicate those restrictions with a grammar, $\mathsf{Bndr}(\mathscr{K}, \Theta)$, which specifies that a boundary term must be built either from an applied recursive argument $x\,\overline{u}$, a previous constructor, or a use of hcomp. We also assume

Constructor declarations (presuppose $\cdot \vdash \Delta\ \mathsf{tel}$)

$$\frac{\mathscr{K} = (\ell_1, C_1) \ldots (\ell_n, C_n) \qquad (\forall i)\ \Delta; (\ell_1, C_1) \ldots (\ell_{i-1}, C_{i-1}) \vdash C_i\ \mathsf{constr}}{\Delta \vdash \mathscr{K}\ \mathsf{constrs}}$$

$$\frac{\begin{array}{c}\Delta \vdash \Gamma\ \mathsf{tel} \quad (\forall k)\ \Delta, \Gamma \vdash \Xi_k\ \mathsf{tel} \quad \vdash \Psi\ \square \quad \Psi \vdash \varphi : \mathbb{F}\\ \delta : \Delta, \Gamma, \overline{\Xi \to \mathsf{H}\,\delta}, \Psi, \varphi \vdash_{\mathscr{K}, \mathsf{H}\,\delta} e : \mathsf{H}\,\delta\\ e = [\varphi_1\ M_1 \ldots \varphi_m\ M_m] \quad (\forall k)\ M_k \in \mathsf{Bndr}(\mathscr{K}; \overline{\Xi \to \mathsf{H}\,\delta})\end{array}}{\Delta; \mathscr{K} \vdash (\Gamma; \overline{\Xi}; \Psi; \varphi; e)\ \mathsf{constr}}$$

Grammar for boundary terms

$$\begin{array}{rcl} N, M \in \mathsf{Bndr}(\mathscr{K}; \Theta) & ::= & x\ \overline{u}\\ & \mid & \mathsf{con}_\ell(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r})\\ & \mid & \mathsf{hcomp}^j_{\mathsf{H}\,\delta}\,[\varphi \mapsto M]\,M_0\end{array}$$

Formation

$$\frac{\Gamma \vdash \delta : \Delta}{\Gamma \vdash \mathsf{H}\,\delta\ \mathsf{type}}$$

Introductions

$$\frac{(\ell, (\Gamma; \overline{\Xi}; \Psi; \varphi; e)) \in \mathscr{K} \qquad \Gamma_0 \vdash \delta : \Delta \qquad \Gamma_0 \vdash \overline{t} : \Gamma[\delta] \qquad \Gamma_0 \vdash \overline{a} : \overline{\Xi[\delta, \overline{t}] \to \mathsf{H}\,\delta} \qquad \Gamma_0 \vdash \overline{r} : \Psi}{\Gamma_0 \vdash \mathsf{con}_\ell(\overline{t}, \overline{a}, \overline{r}) : \mathsf{H}\,\delta[\varphi[\overline{r}] \mapsto e[\overline{t}, \overline{a}, \overline{r}]]}$$

$$\frac{\Gamma \vdash \delta : \Delta \qquad \Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma \vdash u_0 : \mathsf{H}\,\delta[\varphi \mapsto u[0/i]]}{\Gamma \vdash \mathsf{hcomp}^i_{\mathsf{H}\,\delta}\,[\varphi \mapsto u]\,u_0 : \mathsf{H}\,\delta[\varphi \mapsto u[1/i]]}$$

$$\frac{\Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I} \vdash \delta : \Delta \qquad \Gamma, i : \mathbb{I}, \varphi \vdash \delta \equiv \delta[0] : \Delta \qquad \Gamma \vdash u_0 : \mathsf{H}\,\delta[0]}{\Gamma \vdash \mathsf{trans}^i_{\mathsf{H}\,\delta}\,\varphi\,u_0 : \mathsf{H}\,\delta[1][\varphi \mapsto u_0]}$$

Figure 4. Schema for Higher Inductive Types, $\delta : \Delta \vdash \mathsf{H}\,\delta$ type. In the typing of the boundary e the subscript $\mathscr{K}, \mathsf{H}\,\delta$ indicates that this judgement can refer to the labels from $\mathscr{K}$ and $\mathsf{H}\,\delta$ itself. The grammar for boundary terms assumes $x \in \Theta$, that $\overline{u}, \overline{t}$ not mention variables in Θ and that $\ell \in \mathscr{K}$. For further judgemental equalities for trans see Appendix A.5.

a code $\overline{\mathsf{H}}\,\delta : \mathsf{U}_i$ whenever the types in all of the Γ and Ξ telescopes in $\mathscr{K}$ have a code in U_i as well.

Remark 1. *Formally, boundary terms are a separate syntactic entity which we silently include in ordinary terms. In particular, they have a separate equality judgement which is used when typing systems and compositions of boundary terms. This equality is the congruence relation induced by reducing compositions and constructors to their boundary terms. It is likely that this equality can be proved to coincide with the one induced by equality on terms. For details see appendix A.4. This approach is similar to the one of Cavallo and Harper [13].*

We now give some concrete examples. For readability we write $(\ell, (\Gamma; \overline{(x : \Xi \to \mathsf{H}\,\delta)}; \Psi; [\varphi_1 \mapsto M_1, \ldots, \varphi_n \mapsto M_n]))$ in place of $(\ell, (\Gamma, \overline{\Xi}, \Psi, \vee_i \varphi_i, [\varphi_1 \mapsto M_1, \ldots, \varphi_n \mapsto M_n]))$. We also write ℓ rather than con_ℓ.

Example 5.1. Pushout. The context Δ contains the data of a pushout, i.e., $\Delta \stackrel{\text{def}}{=} (A\,B\,C : \mathsf{U})(f : \mathsf{El}\,(C \to A))(g : \mathsf{El}\,(C \to B))$. The pushout has two point constructors, and a path, none with recursive arguments.

- $(\mathsf{inl}, ((a : \mathsf{El}\,A); \cdot; \cdot; []))$
- $(\mathsf{inr}, ((b : \mathsf{El}\,B), \cdot; \cdot; []))$
- $\left(\mathsf{push}, \left((c : \mathsf{El}\,C); \cdot; (i : \mathbb{I}); \left[\begin{array}{l} i = 0 \mapsto \mathsf{inl}(f\,c, \cdot, \cdot) \\ i = 1 \mapsto \mathsf{inr}(g\,c, \cdot, \cdot) \end{array}\right]\right)\right)$

Sphere. We can encode the circle and higher spheres, $\mathbb{S}^n$ for $n \geq 1$, as a point and an n-dimensional surface

- $(\mathsf{base}, (\cdot; \cdot; \cdot; []))$
- $\left(\mathsf{surface}, \left(\cdot; \cdot; \overline{i} : \mathbb{I}^n; \left[\begin{array}{l} i_1 = 0 \vee i_1 = 1 \mapsto \mathsf{base}(\cdot; \cdot; \cdot) \\ \ldots \\ i_n = 0 \vee i_n = 1 \mapsto \mathsf{base}(\cdot; \cdot; \cdot) \end{array}\right]\right)\right)$

Propositional and Higher Truncation. Let $\Delta \stackrel{\text{def}}{=} (A : \mathsf{U})$, and write propositional truncation as $\|A\|_0$.

- $(\mathsf{in}, (\mathsf{El}\,A; \cdot; \cdot; []))$
- $\left(\mathsf{squash}, \left(\cdot; (a_0, a_1 : \|A\|_0); (i : \mathbb{I}); \left[\begin{array}{l} i = 0 \mapsto a_0 \\ i = 1 \mapsto a_1 \end{array}\right]\right)\right)$

For higher truncations $\|A\|_n$, where $n \geq 0$, we use the hub and spoke construction [49, Sect. 7.3], as the schema does not allow quantifying over paths of the HIT itself.[2] Instead of squash then we have the following two constructors:

- $(\mathsf{hub}, (\cdot; (f : \mathbb{S}^{n+1} \to \|A\|_n); \cdot; []))$
- $(\mathsf{spoke}, ((x : \mathbb{S}^{n+1}); (f : \mathbb{S}^{n+1} \to \|A\|_n); (i : \mathbb{I}); e))$

where $e \stackrel{\text{def}}{=} [i = 0 \mapsto f\,x, i = 1 \mapsto \mathsf{hub}(\cdot, f, \cdot)]$.

Finite Powerset. The finite powerset $\mathsf{P_f}(A)$ can also be constructed within this schema, using constructions similar to the ones above, including the hub and spoke constructors to ensure that $\mathsf{P_f}(A)$ is set truncated. We spell out the declaration of the path constructor for commutativity of $\cup$ as we will reference it later

[2] We believe the semantics would support doing so, but it would complicate the schema.

- $\left(\mathsf{comm}, \left(\cdot, (X, Y : \mathsf{P_f}(A)), (i : \mathbb{I}), \left[\begin{array}{l} i = 0 \mapsto X \cup Y \\ i = 1 \mapsto Y \cup X \end{array}\right]\right)\right)$

5.2 Induction under clocks

We now present the principle of induction under clocks. This is an elimination principle defining elements of dependent families of the form $\Gamma, h : \forall\overline{\kappa}.\mathsf{H}\,(\delta[\overline{\kappa}]) \vdash D$ type for a vector of clock variables $\overline{\kappa}$ by defining its action on elements of the form $h = \lambda\overline{\kappa}.\mathsf{con}_\ell(\overline{t}, \overline{a}, \overline{r})$ in such a way that boundary conditions are respected. In the case of an empty list of clock variables, this principle specialises to the elimination principle for HITs of Cavallo and Harper [13], which we refer to as the *usual elimination principle for* H.

Figure 5 presents the rule along with judgemental equalities. The figure first defines the judgement $\Gamma \vdash \mathscr{E} : \mathscr{K} \multimap^\delta D$ for an *elimination list* $\mathscr{E}$, which contains the premises necessary to handle the constructors in $\mathscr{K}$. The case for $\mathscr{K}, (\ell, (\Gamma, \overline{\Xi}, \Psi, \varphi, e))$ requires an elimination list of the form $\mathscr{E}, u$ where $\mathscr{E}$ is an elimination list for $\mathscr{K}$ and u is an element of the type family D at an index built with con_ℓ. In particular u is typed in a context extended with non-recursive arguments γ, recursive arguments $\overline{x}$, interval variables $\overline{i}$ for the constructor $\mathsf{con}_\ell(\gamma, \overline{x}, \overline{i})$, and also induction hypotheses $\overline{y}$ about the variables $\overline{x}$. The element u also needs to suitably match the boundary e whenever φ is satisfied. To express this we transform $e : \mathsf{H}\,\delta$ into $(\!|e|\!)^\delta_{\mathscr{E},\overline{y}}$, the corresponding element of the family D, built using the elimination list $\mathscr{E}$ to handle the previous constructors, and the induction hypotheses $\overline{y}$ to handle the recursive arguments $\overline{x}$. This transformation satisfies the following typing principle, as can be verified by induction on e.

Lemma 5.2. *Let $e = [\varphi_0\,M_0, \ldots, \varphi_{n_i}\,M_{n_i}]$ be the boundary condition for con_i. Assume $\Gamma \vdash \mathscr{E} : \mathscr{K}_{<i} \multimap^\delta D$; then the following typing holds:*

$$\Gamma, \gamma : \forall\overline{\kappa}.\Gamma_i[\delta[\overline{\kappa}]], \overline{x} : \overline{\forall\overline{\kappa}.\Xi_i[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]] \to H(\delta[\overline{\kappa}])},$$
$$\overline{y} : \overline{\Pi(\xi : \forall\overline{\kappa}.\Xi[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]]).D[\lambda\kappa.x[\overline{\kappa}](\xi[\overline{\kappa}])]}, \overline{i} : \Psi_i, \varphi_i \vdash$$
$$(\!|e|\!)^\delta_{\mathscr{E},\overline{x}\mapsto\overline{y}} : D[\lambda\overline{\kappa}.\mathsf{con}_i(\gamma[\overline{\kappa}], \overline{x\,[\overline{\kappa}]}, \overline{i})]$$

In the following examples we will see how instantiating induction under clocks with 1 or 0 clocks respectively induces equivalences which prove that many HITs commute with clock abstraction. In particular, these examples verify the results used in section 4.

Example 5.3. Spheres. In this case induction under clocks unfolds to the principle

$$\frac{\begin{array}{c}\Gamma, x : \forall\kappa.\mathbb{S}^n \vdash D \text{ type} \quad \Gamma \vdash u_b : D[\lambda\kappa.\mathsf{base}] \quad \Gamma \vdash t : \forall\kappa.\mathbb{S}^n \\ \Gamma, \overline{i} : \mathbb{I}^n \vdash u_s : D[\lambda\kappa.\mathsf{surface}(i)][\bigvee_{0 \leq k < n} (i_k = 0 \vee i_k = 1) \mapsto u_b]\end{array}}{\Gamma \vdash (\mathbb{S}^n) - \mathsf{elim}_D(u_b, u_s, t) : D[t]}$$

Elimination lists

$$\frac{\Gamma_0 \vdash \delta : \forall\overline{\kappa}.\Delta \qquad \Gamma_0, h : \forall\overline{\kappa}.\mathsf{H}\,(\delta[\overline{\kappa}]) \vdash D\ \mathsf{type}}{\Gamma_0 \vdash \cdot : \cdot \rightharpoonup^{\delta} D}$$

$$\frac{\Gamma_0 \vdash \mathscr{E} : \mathscr{K} \rightharpoonup^{\delta} D \qquad \Gamma_0, \gamma : \forall\overline{\kappa}.\Gamma_i[\delta[\overline{\kappa}]], \overline{x} : \overline{(\forall\overline{\kappa}.\Xi_i[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]] \to \mathsf{H}\,(\delta[\overline{\kappa}]))}, \overline{y} : \overline{\Pi(\xi : \forall\overline{\kappa}.\Xi_i[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]]).D[\lambda\overline{\kappa}.x[\overline{\kappa}]\xi[\overline{\kappa}]]},\ \overline{i} : \Psi \vdash u(\gamma, x, y, \overline{i}) : D[\lambda\overline{\kappa}.\mathsf{con}_\ell(\gamma[\overline{\kappa}], \overline{x[\overline{\kappa}], i})][\varphi \mapsto (\!|e|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y}}]}{\Gamma_0 \vdash \mathscr{E}, u : \mathscr{K}, (\ell, (\Gamma, \overline{\Xi}, \Psi, \varphi, e)) \rightharpoonup^{\delta} D}$$

Boundary interpretation

$$\begin{aligned}
(\!|e|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y}} &= (\!|e|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\cdot}\\
(\!|[\varphi_0\ M_0, \ldots, \varphi_n\ M_n]|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} &= [\varphi_0\ (\!|M_0|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}, \ldots, \varphi_n\ (\!|M_n|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}]\\
(\!|x_j\ \overline{u}|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} &= y_j\ \lambda\overline{\kappa}.\overline{u}[\delta[\overline{\kappa}], \gamma[\overline{\kappa}], \overline{i}, \hat{\gamma}[\overline{\kappa}]]\\
(\!|\mathsf{con}_\ell(\overline{t_\ell}, \overline{\lambda\xi.\,M}, \overline{r_\ell})|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} &= \mathscr{E}_\ell[\lambda\overline{\kappa}.\overline{t_\ell}[\delta[\overline{\kappa}], \gamma[\overline{\kappa}], \overline{i}, \hat{\gamma}[\overline{\kappa}]], \overline{S}, \overline{R}, \overline{r_\ell}]\\
&\quad (S_i = \lambda\overline{\kappa}.\lambda\overline{\xi_i}.\,M[\delta[\overline{\kappa}], \gamma[\overline{\kappa}], \overline{x}[\overline{\kappa}], \overline{i}, \hat{\gamma}[\overline{\kappa}], \overline{\xi_i}], \quad R_i = \lambda\xi_i.(\!|M_i|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},(\hat{\gamma},\xi_i)})\\
(\!|\mathsf{hcomp}^j\ [\psi \mapsto M]\ M_0|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} &= \mathsf{comp}^j_{D[v\,j]}\ [\psi \mapsto (\!|M|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},(\hat{\gamma},j)}]\ (\!|M_0|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}\\
&\quad (v = \mathsf{hfill}^j_{\forall\overline{\kappa}.\,\mathsf{H}\,(\delta[\overline{\kappa}])}[\psi \mapsto \lambda\overline{\kappa}.\,M[\delta[\overline{\kappa}], \gamma[\overline{\kappa}], \overline{x}[\overline{\kappa}], \overline{i}, \hat{\gamma}[\overline{\kappa}], j]]\ (\lambda\overline{\kappa}.\,M_0[\delta[\overline{\kappa}], \gamma[\overline{\kappa}], \overline{x}[\overline{\kappa}], \overline{i}, \hat{\gamma}[\overline{\kappa}]]))
\end{aligned}$$

Induction under clocks

$$\frac{\Gamma \vdash \delta : \forall\overline{\kappa}.\Delta \quad \Gamma, x : \forall\kappa.\mathsf{H}\,(\delta[\overline{\kappa}]) \vdash D\ \mathsf{type} \qquad \Gamma \vdash \mathscr{E} : \mathscr{K} \rightharpoonup^{\delta} D \qquad \Gamma \vdash u : \forall\overline{\kappa}.\mathsf{H}\,(\delta[\overline{\kappa}])}{\Gamma \vdash (H) - \mathsf{elim}_D(\mathscr{E}, u) : D[u]}$$

Judgemental equalities

$$\begin{aligned}
(H) - \mathsf{elim}_D(\mathscr{E}, \lambda\overline{\kappa}.\mathsf{con}_\ell(\overline{t}, \overline{a}, \overline{r})) &\equiv \mathscr{E}_\ell[\overline{\lambda\overline{\kappa}.t}, \overline{\lambda\overline{\kappa}.a}, \overline{\lambda\xi.\,(H) - \mathsf{elim}_D(\mathscr{E}, \lambda\overline{\kappa}.a(\xi[\overline{\kappa}]))}, \overline{r}]\\
(H) - \mathsf{elim}_D(\mathscr{E}, \lambda\overline{\kappa}.\mathsf{hcomp}[\varphi \mapsto u]\ u_0) &\equiv \mathsf{comp}^i_{D[v\,i]}[\varphi \mapsto (H) - \mathsf{elim}_D(\mathscr{E}, \lambda\overline{\kappa}.u)]\ (H) - \mathsf{elim}_D(\mathscr{E}, \lambda\overline{\kappa}.u_0)\\
&\quad (v = \mathsf{hfill}^i_{\forall\overline{\kappa}.\,\mathsf{H}\,(\delta[\overline{\kappa}])}[\varphi \mapsto \lambda\overline{\kappa}.\,u]\ (\lambda\overline{\kappa}.\,u_0))
\end{aligned}$$

Figure 5. The principle of induction under clocks. In the definition of the boundary interpretation the notation $\mathscr{E}_\ell$ refers to the component of $\mathscr{E}$ for the label ℓ. Also the environment $\hat{\gamma}$ contains the extra assumptions introduced in the con and hcomp cases: in the case of interval variables $i[\overline{\kappa}]$ stands for just i.

Using this, we define $\alpha : \forall\kappa.\mathbb{S}^n \to \mathbb{S}^n$ as follows

$$\alpha(t) \stackrel{\mathsf{def}}{=} (\mathbb{S}^n) - \mathsf{elim}_{\mathbb{S}^n}(\mathsf{base}, \lambda\overline{i}.\,\mathsf{surface}(\overline{i}), t)$$

We must verify the boundary condition which states that if a component of $\overline{i}$ is an endpoint then $u_s(\overline{i}) \equiv u_b$. This follows from the boundary condition for surface.

Since $s \equiv (\mathsf{const}\,s)\ [\kappa_0]$ for all $s : \mathbb{S}^n$, if α as defined above is a right inverse to const we will have produced an equivalence $\mathbb{S}^n \to \forall\kappa.\,\mathbb{S}^n$ and shown the sphere to be clock irrelevant. We can achieve this with another application of induction under a clock, inhabiting the type $s = \mathsf{const}(\alpha(s))$:

$$\Gamma \vdash \mathsf{refl}\ : \lambda\kappa.\,\mathsf{base} = \mathsf{const}(\mathsf{base})$$

$$\Gamma, \overline{i} : \mathbb{I}^n \vdash \mathsf{refl}\ : \lambda\kappa.\,\mathsf{surface}(\overline{i}) = \mathsf{const}(\mathsf{surface}(\overline{i}))$$

The boundary condition in this case states that the second case reduces to the first when $\overline{i}$ contains an endpoint. This follows from congruence of refl .

Propositional and Higher Truncation. To show that propositional truncation commutes with clock quantification we first define a map $\alpha : \forall\kappa.\,\|A\| \to \|\forall\kappa.\,A\|$ by induction under clocks as follows:

$$\Gamma, a : \forall\kappa.\,A \vdash \mathsf{in}(a) : \|\forall\kappa.\,A\|$$

$$\Gamma, x_0, x_1 : \forall\kappa.\,\|A\|, y_0, y_1 : \|\forall\kappa.\,A\|, i : \mathbb{I} \vdash$$
$$\mathsf{squash}(y_0, y_1, i) : \|\forall\kappa.\,A\|\,[(i = 0) \mapsto y_0, (i = 1) \mapsto y_1]$$

The above data defines a map α satisfying

$$\alpha(\lambda\kappa.\mathsf{in}(a\ [\kappa])) = \mathsf{in}(a)$$
$$\alpha(\lambda\kappa.\,\mathsf{squash}(x_0\ [\kappa], x_1\ [\kappa], r) = \mathsf{squash}(\alpha(x_0), \alpha(x_1), r)$$

Let $\beta : \|\forall\kappa.\,A\| \to \forall\kappa.\,\|A\|$ be the canonical map. To show that $\beta \circ \alpha = \mathsf{id}$, it suffices by induction under clocks to show

$$\beta(\alpha(\lambda\kappa.\mathsf{in}(a\ [\kappa]))) = \lambda\kappa.\mathsf{in}(a\ [\kappa])$$
$$\beta(\alpha(\lambda\kappa.\,\mathsf{squash}(x_0\ [\kappa], x_1\ [\kappa], r)) = \lambda\kappa.\mathsf{squash}(x_0\ [\kappa], x_1\ [\kappa], r)$$

In the latter case the left hand side reduces to

$$\lambda\kappa.\mathsf{squash}(\beta(\alpha(x_0))\ [\kappa], \beta(\alpha(x_1))\ [\kappa], r)$$

and so the case follows by induction. Note that in both these uses of induction under clocks, the boundary condition is satisfied. For example, in the latter case, when $r = 0$ the term reduces to the proof of $\beta(\alpha(x_0)) = x_0$ given by the induction hypothesis. Showing that $\alpha \circ \beta = \mathsf{id}$ follows from an application of the usual elimination principle.

For the higher truncations we apply induction under clocks again, starting by observing that we have terms as follows:

$$a : \forall\kappa.A \vdash \mathsf{in}(a) : \|\forall\kappa.\, A\|_n$$
$$x, y \vdash \mathsf{hub}(y \circ \mathsf{const}) : \|\forall\kappa.A\|_n$$
$$s : \forall\kappa.\, \mathbb{S}^{n+1}, x, y, i : \mathbb{I} \vdash \mathsf{spoke}(s\ [\kappa_0], y \circ \mathsf{const}, i) : \|\forall\kappa.A\|_n$$

where $x : \forall\kappa.\ (\mathbb{S}^{n+1} \to \|A\|_n)$ and $y : (\forall\kappa.\mathbb{S}^{n+1}) \to \|\forall\kappa.A\|_n$. For us to apply the principle, the third term would need to reduce the second when $i = 1$, which it does, and $y(s)$ when $i = 0$. When $i = 0$ we have instead $\mathsf{spoke}(s\ [\kappa_0], y \circ \mathsf{const}, i) = y(\lambda\kappa.\, s\ [\kappa_0])$. From the sphere example we know that $\mathbb{S}^{n+1}$ is clock irrelevant, and hence we have a path $p : \lambda\kappa.\, s\ [\kappa_0] = s$. This means that we can obtain the term needed for the induction as

$$\mathsf{hcomp}^j \begin{bmatrix} (i = 0) \mapsto y\ (p\ j) \\ (i = 1) \mapsto \mathsf{hub}(y \circ \mathsf{const}) \end{bmatrix} \mathsf{spoke}(s\ [\kappa_0], y \circ \mathsf{const}, i).$$

We defer the details of showing that this map is inverse to the canonical one to the appendix.

Pushout. Using constructions similar to the examples above, one can prove that pushouts commute with $\forall\kappa$ using induction under clocks. More precisely, let $A, B, C : \forall\kappa.\mathsf{U}$, $f : \forall\kappa.C\ [\kappa] \to A\ [\kappa]$, and $g : \forall\kappa.C\ [\kappa] \to B\ [\kappa]$. Then the canonical map

$$(\forall\kappa.\, A) \sqcup_{(\forall\kappa.\, C)} (\forall\kappa.\, B) \to \forall\kappa.((A\ [\kappa]) \sqcup_{(C\ [\kappa])} (B\ [\kappa]))$$

is an equivalence

Finite Powerset. A full account of the constructors for the finite powerset is not enlightening, so we instead exemplify the induction principle in the cases for singleton, union, and idempotence constructors:

$$\Gamma, x : \forall\kappa.A\ [\kappa] \vdash u_{\{-\}} : D[\lambda\kappa.\{x\ [\kappa]\}]$$
$$\Gamma, x, x' : \forall\kappa.\mathsf{P_f}(A\ [\kappa]), y : D(x), y' : D(x') \vdash$$
$$u_\cup(x, x', y, y') : D(\lambda\kappa.x\ [\kappa] \cup x'\ [\kappa])$$
$$\Gamma, x : \forall\kappa.\mathsf{P_f}(A\ [\kappa]), y : D(x), i : \mathbb{I} \vdash$$
$$u_{\mathsf{idem}}(x, y, i) : D(\lambda\kappa.\mathsf{idem}(x\ [\kappa], i)) \begin{bmatrix} (i = 0) \mapsto u_\cup(x, x, y, y) \\ (i = 1) \mapsto y \end{bmatrix}$$

In addition it will contain the hub and spoke constructors derived from $\mathbb{S}^1$, towards obtaining a set type. Finally, this principle shows that there is an equivalence $\mathsf{P_f}(\forall\kappa.A) \simeq \forall\kappa.\mathsf{P_f}(A)$ as required for the encoding of labelled transition systems. The concrete calculations are analogous to those for the pushout or trunctation depending on the constructor.

As a step towards reintroducing clock-irrelevance as an axiom, one would need to show that the collection of clock irrelevant types is closed under HIT formation:

Theorem 5.4. *Let* $\delta : \Delta \vdash H\delta$ type *be a higher inductive type with constructors*

$$\mathsf{con}_i : (\gamma : \Gamma_i[\delta])(x : \Xi_i[\delta, \gamma] \to H\delta)(r : \Psi_i) \to H\delta[\varphi_i \mapsto e_i]$$

Then $H\delta$ *is clock irrelevant if* $\Gamma_i[\delta]$ *is clock irrelevant for all* i.

As a special case $\mathbb{N}$ is clock-irrelevant as needed for the encoding of streams of $\mathbb{N}$.

6 Denotational semantics

Previous work has defined a model of CTT in presheaves over a cube category [20, 43] and a model of CloTT in covariant presheaves over a category of time objects $\mathscr{T}$ [35]. We combine these by considering the category $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$ of covariant presheaves over $\mathscr{C} \times \mathscr{T}$, where $\mathscr{C}$ is the opposite category of the usual choice of cube category. Let U be a Hofmann-Streicher universe [29] in $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$.

Following the approach of Licata et al. [34], we construct the model using the internal type theory of $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$. We must provide an interval object and a cofibration object, and show that they satisfy certain axioms. As noted by Coquand et al. [23] presheaves over the product of the cube category with any small category admits a model of CTT. The interval $\mathbb{I}$ and cofibration object $\mathbb{F}$ are taken to be the inclusion of those from $\mathsf{PSh}(\mathscr{C})$.

Recall [43] that a *CCHM fibration* (A, α) over a context $\Gamma : \mathsf{U}$ is a family $A : \Gamma \to \mathsf{U}$ equipped with a fibration structure α. The denotational semantics is given by the category with families (CwF) [25] Fib constructed as follows: Contexts are global elements of U, families are global CCHM fibrations, and elements of (A, α) are elements of A.

To model HITs we follow Coquand et al. [21]. The principle of induction under clocks is justified semantically using that $\forall\kappa.(-)$ can be modelled as an ω^{op}-limit [12] and that the structure maps of this limit cannot change the outermost constructor in HITs, allowing for an inductive proof.

To model a Fitch-style modality like $\triangleright$ we must define a dependent adjunction [19] on Fib. A dependent adjunction consists of an endofunctor L and an action on types R such that elements of $L(\Gamma) \vdash A$ type correspond bijectively to elements of $\Gamma \vdash R(A)$ type. Most of this data is as defined by Mannaa et al. [35], but to lift this to Fib we must define a CCHM fibration on the right adjoint types. This can be done using the following more general construction.

Definition 6.1. Let L be an endofunctor on $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$ which preserves finite limits. We say that L *preserves the interval* if there is an isomorphism $L(\mathbb{I}) \cong \mathbb{I}$ which preserves endpoints. We say that L *preserves cofibrations* if it maps cofibrations to cofibrations and preserves pullbacks where the vertical maps are cofibrations.

Theorem 6.2. *Let $L \dashv R$ be a dependent adjunction in the internal type theory of* $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$. *If L preserves cofibrations and the interval one can define a dependent adjunction $L' \dashv R'$ on* Fib *such that the underlying type of $R'(A, \alpha)$ is $R(A)$.*

7 Related work

Guarded Cubical Type Theory [9] combines Cubical Type Theory with single-clocked guarded recursion. While this case is useful for many purposes, it cannot be used to encode coinductive types. Møgelberg and Veltri [40] extend Guarded Cubical Type Theory with ticks as in CloTT. They give a model of this calculus including HITs, and show that bisimilarity coincides with path equality for a large class of guarded recursive types. This paper can be seen as an extension of that with multiple clocks, allowing for these results to be lifted from guarded recursive types to coinductive types. Note that modelling the multiclocked case is much more complex than the single clock case. In particular, equipping $\triangleright$ with a composition structure is much more challenging. The extended language of ticks giving computation rules for clock irrelevance presented here is also new.

The encoding of coinductive types using guarded recursion was first described by Atkey and McBride [6] in the simply typed setting. Since then a number of dependent type theories have been developed for programming and reasoning with these [17, 38], of which CloTT is the most advanced. Bahr. et al. [7] prove syntactic properties of CloTT including strong normalisation and canonicity, but only for a pure calculus without identity or path types. The model of CloTT constructed by Mannaa et al. [35] considers extensional identity types, but no cubical features. Coinductive types can also be encoded using a combination of guarded recursion and a $\Box$-modality [18]. This approach has not been studied in combination with CTT yet, and also appears to be less flexible, e.g., it does not seem possible to define nested coinductive types.

Sized types [32] is another approach to encoding productivity in types, by annotating (co)inductive types with sizes indicating a bound on the size of the allowed elements. Specifically, sized types reduce both termination and productivity to (well-founded) induction on sizes. They have been extensively studied from the syntactic perspective [1, 2, 45] but are not well understood from the perspective of denotational semantics. Our view is that sized types are closer to working in the models of type theory, like the one provided here, and guarded recursion is a more abstract, principled perspective. This is supported by the model of guarded recursion in sized types constructed by Veltri and van der Weide [51]. To our knowledge sized types have never been used to solve equations with negative occurrences, which is an important application of guarded recursion.

The coincidence of bisimilarity and path equality for streams as coinductive records has been proved in Cubical Agda [53], and it is likely that the proof can be extended to general M-types. Veltri [50] proves that bisimilarity implies path equality for the final coalgebra for the P_f using sized types in Cubical Agda. This coincidence should therefore be seen as a feature of Cubical Type Theory, rather than guarded recursion. On the other hand, when proving such results guarded recursion is a powerful framework for ensuring productivity of definitions, as illustrated by the examples in this paper.

The final coalgebra for the finite powerset functor can be constructed in set theory as a limit of an $\omega + \omega$-indexed sequence [54]. This construction has been formalised in Cubical Agda by Veltri [50] using the lesser limited principle of omniscience, a weak choice principle. From this perspective it is interesting that our model uses ω-indexed step-indexing only, and therefore LTS as constructed in section 4 is realised in the model as an ω-limit. This construction works because of the formulation of P_f as a HIT and because the ω-chain is constructed *externally*, using judgemental equality. In particular, the counter example constructed in Proposition 5 of [50] uses an ω-chain of elements whose projections are only equal up to path equality.

Multimodal dependent type theory [27] is a general framework for dependent modal type theories parametrised over *mode theories*. By instantiating this appropriately, one can recover e.g., the basic modal framework needed for internalising parametricity in type theory [8, 14], or for the combination of $\triangleright$ and $\Box$ used for guarded recursion by Clouston et al. [18]. Generalising this to multiple clocks seems to require a notion of dependent mode theory, as for example, the modal operators $\triangleright$ depend on Clk.

8 Conclusion and future work

We have presented the type theory CCTT, and shown that the principle of induction under clocks can be used to construct a rich supply of functors for which coinductive types can be encoded using guarded recursion. This allows for simple programming with a wide range of coinductive types, including ones constructed using higher inductive types. We have seen by example how to prove coincidence of path equality with bisimilarity for these types. We believe this type theory is useful not just for coinductive reasoning, but also for reasoning about advanced programming language features using a form of synthetic guarded domain theory [39, 44]. In fact, an earlier version of CCTT has already been used for a semantic proof of applicative simulation being a congruence for a lambda calculus with finite non-determinism [41].

We are currently developing a prototype implementation of CCTT as an extension of Cubical Agda. The main thing missing is the principle of induction under clocks. The proof of Theorem 4.3 has been verified in this.

We would also like to prove canonicity for CCTT, building on similar results for Cubical Type Theory [22, 31, 47] and

Clocked Type Theory [7]. For Cubical Type Theory canonicity is proved for terms in a context of only interval variables. For CCTT, the context should be allowed to also contain free clock variables in order for terms of type $\forall\overline{\kappa}.H(\delta\,[\overline{\kappa}])$ to reduce to a form in which the β-rule for induction under clocks can be applied. On the other hand, we believe that it should not be necessary to include free tick variables, and so the only tick that needs to be considered in reductions is $\diamond$. Our equational rules have been designed with this in mind.

Unlike this paper, other type theories for multi-clocked guarded recursion [11, 35] take clock irrelevance (9) as an axiom. This requires that universes be indexed by clock contexts, and the $\triangleright$ modality be restricted to $\triangleright^\kappa : \mathsf{U}_\Delta \to \mathsf{U}_\Delta$ for $\kappa \in \Delta$, because an unrestricted $\triangleright^\kappa$ breaks clock irrelevance [12]. Bizjak and Møgelberg [12] show how to construct such universes of types that are clock-irrelevant in the sense of the map $A \to \forall\kappa.A$ being an isomorphism, rather than an equivalence. Future work includes constructing larger universes of types clock-irrelevant in the more liberal sense used in this paper.

Acknowledgment

This work was supported by a research grant (13156) from VILLUM FONDEN.

References

[1] Andreas Abel and Brigitte Pientka. 2013. Wellfounded Recursion with Copatterns: A Unified Approach to Termination and Productivity. In *Proceedings ICFP 2013*. ACM, 185–196.

[2] Andreas Abel, Andrea Vezzosi, and Theo Winterhalter. 2017. Normalization by evaluation for sized dependent types. *Proceedings of the ACM on Programming Languages* 1, ICFP (2017), 1–30.

[3] Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. 2015. Non-Wellfounded Trees in Homotopy Type Theory. In *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 38)*, Thorsten Altenkirch (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 17–30. https://doi.org/10.4230/LIPIcs.TLCA.2015.17

[4] Andrew W. Appel and David McAllester. 2001. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst* 23, 5 (2001), 657–683.

[5] Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. 2007. A very modal model of a modern, major, general type system. In *POPL*. 109–122.

[6] Robert Atkey and Conor McBride. 2013. Productive coprogramming with guarded recursion. *ACM SIGPLAN Notices* 48, 9 (2013), 197–208.

[7] Patrick Bahr., Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. 2017. The clocks are ticking: No more delays!. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–12.

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

[9] Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. 2019. Guarded Cubical Type Theory: Path Equality for Guarded Recursion. *Journal of Automated Reasoning* 63, 2 (2019), 211–253.

[10] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. 2012. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science* 8, 4 (2012).

[11] Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus E Møgelberg, and Lars Birkedal. 2016. Guarded dependent type theory with coinductive types. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 20–35.

[12] Aleš Bizjak and Rasmus Ejlers Møgelberg. 2020. Denotational semantics for guarded dependent type theory. *Mathematical Structures in Computer Science* 30, 4 (2020), 342–378.

[13] Evan Cavallo and Robert Harper. 2019. Higher inductive types in cubical computational type theory. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–27.

[14] Evan Cavallo and Robert Harper. 2020. Internal Parametricity for Cubical Type Theory. In *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[15] James Chapman, Tarmo Uustalu, and Niccolò Veltri. 2019. Quotienting the delay monad by weak bisimilarity. *Mathematical Structures in Computer Science* 29, 1 (2019), 67–92.

[16] Ranald Clouston. 2018. Fitch-style modal lambda calculi. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 258–275.

[17] Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. 2015. Programming and Reasoning with Guarded Recursion for Coinductive Types. In *FoSSaCS*.

[18] Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. 2015. Programming and Reasoning with Guarded Recursion for Coinductive Types. In *Proceedings of FoSSaCS*. 407–421.

[19] Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. 2020. Modal Dependent Type Theory and Dependent Right Adjoints. *Mathematical Structures in Computer Science* 30, 2 (2020), 118–138.

[20] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2018. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *21st International Conference on Types for Proofs and Programs (TYPES 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[21] Thierry Coquand, Simon Huber, and Anders Mörtberg. 2018. On Higher Inductive Types in Cubical Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (Oxford, United Kingdom) *(LICS '18)*. Association for Computing Machinery, New York, NY, USA, 255–264. https://doi.org/10.1145/3209108.3209197

[22] Thierry Coquand, Simon Huber, and Christian Sattler. 2019. Homotopy Canonicity for Cubical Type Theory. In *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 11:1–11:23. https://doi.org/10.4230/LIPIcs.FSCD.2019.11

[23] Thierry Coquand, Fabian Ruch, and Christian Sattler. 2020. Constructive sheaf models of type theory. arXiv:1912.10407 [math.LO]

[24] Nils Anders Danielsson. 2010. Beating the Productivity Checker Using Embedded Languages. In *PAR*, Vol. 43. 29–48.

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

[26] Dan Frumin, Herman Geuvers, Léon Gondelman, and Niels van der Weide. 2018. Finite sets in homotopy type theory. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, June Andronick and Amy P. Felty (Eds.). ACM, 201–214. https://doi.org/10.1145/3167085

[27] Daniel Gratzer, GA Kavvos, Andreas Nuyts, and Lars Birkedal. 2020. Multimodal dependent type theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. 492–506.

[28] Martin Hofmann. 1997. Syntax and semantics of dependent types. In *Extensional Constructs in Intensional Type Theory*. Springer, 13–54.

[29] Martin Hofmann and Thomas Streicher. 1999. Lifting Grothendieck universes. (1999). www.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf Unpublished.

[30] Simon Huber. 2017. A Cubical Type Theory for Higher Inductive Types. (2017). http://www.cse.chalmers.se/~simonhu/misc/hcomp.pdf

[31] Simon Huber. 2019. Canonicity for cubical type theory. *Journal of Automated Reasoning* 63, 2 (2019), 173–210.

[32] J. Hughes, L. Pareto, and A. Sabry. 1996. Proving the Correctness of Reactive Systems Using Sized Types. In *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*. 410–423.

[33] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming* 28 (2018).

[34] R. D. Licata, I. Orton, A.M. Pitts, and B. Spitters. 2018. Internal Universes in Models of Homotopy Type Theory. In *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 108)*, Hélène Kirchner (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 22:1–22:17.

[35] Bassel Mannaa, Rasmus Ejlers Møgelberg, and Niccolò Veltri. 2020. Ticking clocks as dependent right adjoints: Denotational semantics for clocked type theory. *Logical Methods in Computer Science* 16 (2020).

[36] Per Martin-Löf. 1984. *Intuitionistic Type Theory*. Bibliopolis, Napoli.

[37] Conor McBride and Ross Paterson. 2008. Applicative programming with effects. *Journal of functional programming* 18, 1 (2008), 1–13.

[38] Rasmus Ejlers Møgelberg. 2014. A type theory for productive coprogramming via guarded recursion. In *CSL-LICS*. 71:1–71:10.

[39] Rasmus Ejlers Møgelberg and Marco Paviotti. 2016. Denotational semantics of recursive types in synthetic guarded domain theory. In *LICS*.

[40] Rasmus Ejlers Møgelberg and Niccolò Veltri. 2019. Bisimulation as path type for guarded recursive types. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29. https://doi.org/10.1145/3290317

[41] Rasmus Ejlers Møgelberg and Andrea Vezzosi. 2021. Two Guarded Recursive Powerdomains for Applicative Simulation. *arXiv preprint arXiv:2112.14056* (2021).

[42] Hiroshi Nakano. 2000. A modality for recursion. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science*. IEEE, 255–266.

[43] I. Orton and A.M. Pitts. 2018. Axioms for Modelling Cubical Type Theory in a Topos. *Logical Methods in Computer Science* Volume 14, Issue 4 (Dec 2018).

[44] Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. 2015. A model of PCF in guarded type theory. *Electronic Notes in Theoretical Computer Science* 319 (2015), 333–349.

[45] Jorge Luis Sacchini. 2013. Type-Based Productivity of Stream Definitions in the Calculus of Constructions. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. 233–242.

[46] Kristina Sojakova. 2015. Higher Inductive Types as Homotopy-Initial Algebras. *SIGPLAN Not.* 50, 1 (Jan. 2015), 31–42. https://doi.org/10.1145/2775051.2676983

[47] Jonathan Sterling and Carlo Angiuli. 2021. Normalization for cubical type theory. In *LICS*.

[48] Jonathan Sterling and Robert Harper. 2018. Guarded Computational Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 879–888.

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

[50] Niccolò Veltri. 2021. Type-theoretic constructions of the final coalgebra of the finite powerset functor. In *FSCD*. To appear.

[51] Niccolò Veltri and Niels van der Weide. 2019. Guarded Recursion in Agda via Sized Types. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*. 32:1–32:19.

[52] Niccolò Veltri and Andrea Vezzosi. 2020. Formalizing π-Calculus in Guarded Cubical Agda. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New Orleans, LA, USA) *(CPP 2020)*. Association for Computing Machinery, New York, NY, USA, 270–283. https://doi.org/10.1145/3372885.3373814

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

[54] James Worrell. 2005. On the final sequence of a finitary set functor. *Theor. Comput. Sci.* 338, 1-3 (2005), 184–199. https://doi.org/10.1016/j.tcs.2004.12.009

A Appendix

A.1 Omitted proofs Section 3

Proof of Lemma 3.2. Given $f : \triangleright^\kappa A \to A$, let p be the corresponding proof of (7), then as mentioned the center of contraction for

$$\Sigma(x : A).\mathrm{Path}_A(x, f\,(\lambda(\alpha:\kappa).x))$$

will be the pair $(\mathrm{fix}^\kappa f, p)$. Then for any other such pair (h, p_h) we have to show $(\mathrm{fix}^\kappa f, p) = (h, p_h)$, which is equivalent to

$$\Sigma(q : \mathrm{fix}^\kappa f = h).\ q_* p = p_h$$

We define q by guarded recursion

$$\mathrm{fix}^\kappa\, \lambda(r : \triangleright^\kappa(\mathrm{fix}^\kappa f = h)).(p^{-1}, p_h^{-1})_*(\mathrm{ap}_f(\lambda i.\lambda(\alpha:\kappa).r\,[\alpha]\,i))$$

We then proceed to prove $q_* p = p_h$ by first observing that it is equivalent to

$$(p, p_h)_* q = \mathrm{ap}_f(\lambda i.\lambda(\alpha:\kappa).q\,[\alpha]\,i)$$

so that by expanding q by (7) on the left hand side and canceling the transports we obtain the right hand side and the proof is concluded. □

Proof of Lemma 3.1. We have

$$\Gamma, i_0, \ldots, i_n : \mathbb{I} \vdash u_0 \stackrel{\mathrm{def}}{=} u[0/i_0] : \kappa \rightsquigarrow \Gamma'$$

and

$$\Gamma, i_0, \ldots, i_n : \mathbb{I}, \varphi, j : \mathbb{I} \vdash p(j) \stackrel{\mathrm{def}}{=} \mathrm{tirr}(u_0, u, j) : \kappa \rightsquigarrow \Gamma'$$

Define $\Gamma, i_0, \ldots, i_n : \mathbb{I}, j : \mathbb{I} \vdash C$ type by

$$\mathrm{hfill}^j\,[\varphi \mapsto A[p(j)/\alpha]]\,A[u_0/\alpha]$$

and then take $B \stackrel{\mathrm{def}}{=} C\,1$. Then we define $\mathrm{filler}(t, u)$ as

$$\mathrm{hcomp}^j_C\,[\varphi \mapsto t\,[p(j)]]\,(t\,[u_0])$$

□

A.2 Substitution for Tick Application

In figure 6 we present the formation rules for substitutions, based on the ones from Mannaa et al. [35], and extended to account for the new tick judgemnts and the contexts from cubical type theory. In what follows we explain how to apply a substitution to a tick application term.

Operation 1. *Given* $\Delta \vdash \sigma : \Gamma$ *and* $\Gamma \vdash u : \kappa \rightsquigarrow \Gamma'$ *we define an operation* $\mathrm{residual}(\sigma, u)$ *returning tuples of one of two forms*

- (Δ', σ') *such that* $\Delta \vdash u\sigma : \kappa\sigma \rightsquigarrow \Delta'$ *and* $\Delta' \vdash \sigma' : \Gamma'$, *and* $\Delta \vdash \sigma|_{\Gamma'} \equiv \sigma' : \Gamma'$.
- $(\Delta', \kappa'', \sigma')$ *such that* $\Delta \vdash (\kappa\sigma, u\sigma) \rightsquigarrow \Delta'$ *and*

$$\Delta', \kappa'' : \mathsf{clock} \vdash \sigma' : \Gamma',$$

such that $\kappa\sigma' = \kappa''$ *and* $\Delta \vdash \sigma|_{\Gamma'} \equiv \sigma'[\kappa\sigma/\kappa''] : \Gamma'$.

Construction. The tick u contains tick variables $\alpha_0 : \kappa \ldots \alpha_n : \kappa$, here given in the order they appear in Γ, so in particular we have $\Gamma = \Gamma_1, \alpha_0 : \kappa, \Gamma_2$ and $\Gamma' \sqsubseteq \Gamma_1, \mathsf{TL}(\Gamma_2)$. Then let us look at the restriction of σ to $\Gamma_1, \alpha_0 : \kappa$. We have two cases:

Substitutions

$$\frac{\Gamma \vdash}{\Gamma \vdash [] : \cdot} \qquad \frac{\Gamma \vdash \sigma : \Gamma' \qquad \Gamma \vdash t : A\sigma}{\Gamma \vdash (\sigma, t) : \Gamma', x : A}$$

$$\frac{\Gamma \vdash \sigma : \Gamma' \qquad \Gamma \vdash \kappa' : \mathsf{clock}}{\Gamma \vdash (\sigma, \kappa') : \Gamma', \kappa : \mathsf{clock}} \qquad \frac{\Gamma \vdash \sigma : \Gamma' \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash (\sigma, r) : \Gamma', i : \mathbb{I}}$$

$$\frac{\Gamma \vdash \sigma : \Gamma' \qquad \Gamma' \vdash \varphi : \mathbb{F} \qquad \Gamma \vdash \varphi\sigma = 1_{\mathbb{F}} : \mathbb{F}}{\Gamma \vdash \sigma : \Gamma', \varphi}$$

$$\frac{\Gamma_0 \vdash \sigma : \Gamma' \qquad \Gamma \vdash u : \kappa\sigma \rightsquigarrow \Gamma_0}{\Gamma \vdash (\sigma, u) : \Gamma', \alpha : \kappa}$$

$$\frac{\Gamma_0 \vdash \sigma : \Gamma' \qquad \Gamma \vdash (\kappa', v) \rightsquigarrow \Gamma_0}{\Gamma \vdash \sigma, (\kappa', v) : \Gamma', \kappa : \mathsf{clock}, \alpha : \kappa}$$

Figure 6. Formation rules for substitutions.

(i) (σ_1, v) with $\Delta' \vdash \sigma_1 : \Gamma_1$ and $\Delta_0 \vdash v : \kappa\sigma_1 \rightsquigarrow \Delta'$ with $\Delta_0 \sqsubseteq \Delta$.
(ii) $(\sigma_0, (\kappa', v))$ with $\Gamma_1 = \Gamma_0, \kappa : \mathsf{clock}$, and $\Delta' \vdash \sigma_0 : \Gamma_0$, and $\Delta_0 \vdash (\kappa', v) \rightsquigarrow \Delta'$, with $\Delta_0 \sqsubseteq \Delta$.

In either case the tick v lives in the context $\Delta_0 \sqsubseteq \Delta$ because other components of the substitution σ, e.g. for $\alpha_1 \ldots \alpha_n$, might have shrinked the context so.

In case (i) we extend σ_1 to $\Delta' \vdash \sigma_1' : \Gamma_1, \mathsf{TL}(\Gamma_2)$ because whenever we have one of $\Delta \vdash \kappa_i : \mathsf{clock}$, or $\Delta \vdash r : \mathbb{I}$, or $\Delta \vdash \varphi \equiv 1_{\mathbb{F}} : \mathbb{F}$ we also have the same in $\Delta' \sqsubseteq \Delta$. Finally we take σ' to be $\sigma_1'|_{\Gamma'}$, which agrees with $\sigma|_{\Gamma'}$ by construction. The constructed tuple will be (Δ', σ').

In case (ii) we extend σ_0 first to $\Delta', \kappa'' \vdash (\sigma_0, \kappa'') : \Gamma_0, \kappa : \mathsf{clock}$ then to a subsitution $\Delta', \kappa'' \vdash \sigma_0' : \Gamma_1, \mathsf{TL}(\Gamma_2)$ as above, furthermore noting that $\mathsf{TL}(\Gamma_2)$ does not depend on κ. Finally we take σ' to be $\sigma_0'|_{\Gamma'}$. The substitution $\sigma'[\kappa'/\kappa'']$ then agrees with $\sigma|_{\Gamma'}$ by construction, and we also have $\kappa\sigma' \equiv \kappa''$. The constructed tuple will be $(\Delta', \kappa'', \sigma')$.

To show that we have the correct typing for $u\sigma$ we observe that Δ' is smaller as a subcontext of Δ than the ones the ticks $\alpha_1\sigma \ldots \alpha_n\sigma$ target, so they can all be weakened to fit the typing $\Delta \vdash \alpha_i\sigma : \kappa\sigma_1 \rightsquigarrow \Delta'$. In case (i) then we are done by extending this observation to v. In case (ii) we can further derive $\Delta \vdash (\kappa\sigma_1, \alpha_i\sigma) \rightsquigarrow \Delta'$ which gives us what we want. □

For $\Delta \vdash \sigma : \Gamma$, and $\Gamma' \vdash t : \triangleright\,(\alpha:\kappa).A$, and $\Gamma \vdash u : \kappa \rightsquigarrow \Gamma'$, we have that substitution commutes with tick application in the following way:

$$(t\,{}_{\Gamma'}[u])\sigma = \begin{cases} t\sigma'\,{}_{\Delta'}[u\sigma] & \text{if } \mathrm{residual}(\sigma, u) = (\Delta', \sigma') \\ (\kappa''.\, t\sigma')\,{}_{\Delta'}[(\kappa\sigma, u\sigma)] & \text{if } \mathrm{residual}(\sigma, u) = (\Delta', \kappa'', \sigma') \end{cases}$$

We want to show that typing is preserved. For the first case, we can assume $\Delta' \vdash t\sigma' : \triangleright\,(\alpha:\kappa\sigma').A(\sigma', \alpha)$, so by the tick

application rule we have $\Delta \vdash t\sigma'\ {}_{\Delta'}[u\sigma] : A(\sigma', \alpha)[u\sigma/\alpha]$, where the latter type is equal to $A[u/\alpha]\sigma$ as expected. For the second case, we can assume $\Delta', \kappa'' \vdash t\sigma' : \triangleright(\alpha : \kappa\sigma').A(\sigma', \alpha)$, so by the forcing tick application rule we have

$$\Delta \vdash (\kappa''.\, t\sigma')\ {}_{\Delta'}[(\kappa\sigma, u\sigma)] : A(\sigma', \alpha)[u\sigma : \kappa\sigma/\alpha : \kappa''],$$

where the latter type is equal to $A[u/\alpha]\sigma$ as expected.

Operation 2. *Given* $\Delta \vdash \sigma : \Gamma$ *and* $\Gamma \vdash (\kappa, u) \rightsquigarrow \Gamma'$ *we define an operation* $\mathsf{bresidual}(\sigma, (\kappa, u))$ *returning tuples of the form*

- (Δ', σ') *such that* $\Delta \vdash (\kappa\sigma, u\sigma) \rightsquigarrow \Delta'$ *and* $\Delta' \vdash \sigma' : \Gamma'$, *and* $\Delta \vdash \sigma|_{\Gamma'} \equiv \sigma' : \Gamma'$

Construction. Here the tick u might not contain any tick variables, in which case we take Δ' to be Δ and σ' to be $\sigma|_{\Gamma'}$. If u does contain tick variables then we have cases (i) and (ii) as in the construction of $\mathsf{residual}(-,-)$. We chose to use κ in the assumption $\Gamma \vdash (\kappa, u) \rightsquigarrow \Gamma'$ so that the names in the cases would line up with the previous construction.

In case (i) we construct the tuple (Δ', σ') as we did before, and the same reasoning extends to the well-typing of $\Delta \vdash (\kappa\sigma, u\sigma) \rightsquigarrow \Delta'$.

In case (ii) let us recall that here Γ is of the form $\Gamma_0, \kappa : \mathsf{clock}, \alpha_0 : \kappa, \Gamma_2$, with $\Gamma' \sqsubseteq \Gamma_0, \kappa : \mathsf{clock}, \mathsf{TL}(\Gamma_2)$. We also have $\Delta' \vdash (\sigma_0, \kappa') : \Gamma_0, \kappa : \mathsf{clock}$, which agrees with σ restricted to the same context. As before we can extend (σ_0, κ') to a substitution for $\Gamma_0, \kappa : \mathsf{clock}, \mathsf{TL}(\Gamma_2)$ by using the relevant components of σ, and finally obtain the desired σ' by restriction to Γ'. □

For $\Delta \vdash \sigma : \Gamma$, and $\Gamma', \kappa : \mathsf{clock} \vdash t : \triangleright(\alpha : \kappa).A$, and $\Gamma \vdash (\kappa', u) \rightsquigarrow \Gamma'$, we have that substitution commutes with forcing tick application in the following way:

$$((\kappa.t)\ {}_{\Gamma'}[(\kappa', u)])\sigma = (\kappa.\, t(\sigma', \kappa))\ {}_{\Delta'}[(\kappa'\sigma, u\sigma)]$$

where $\mathsf{bresidual}(\sigma, (\kappa', u)) = (\Delta', \sigma')$. We want to show that typing is preserved. We can assume

$$\Delta', \kappa : \mathsf{clock} \vdash t(\sigma', \kappa) : \triangleright(\alpha : \kappa).\, A(\sigma', \kappa, \alpha),$$

then by the forcing tick application rule we have

$$\Delta \vdash (\kappa.\, t(\sigma', \kappa))\ {}_{\Delta'}[(\kappa'\sigma, u\sigma)] : A(\sigma', \kappa, \alpha)[u\sigma : \kappa'\sigma/\alpha : \kappa],$$

where the latter type is equal to $A[u : \kappa'/\alpha : \kappa]\sigma$ as expected.

A.3 Omitted proofs Section 4

Proof of Lemma 4.2. The case of composition is clear, and products follow from the fact that $\forall\kappa.(A \times B) \simeq (\forall\kappa.A) \times (\forall\kappa.B)$. Likewise, the case of Π-types follows from the fact that $\forall\kappa.\Pi(a : A).B \simeq \Pi(a : A).\forall\kappa.B$ which can be proved by commuting two arguments. In the case of Σ, since $\forall\kappa.(-)$ behaves as a function space from a type of clocks, one can prove

$$\begin{aligned}\forall\kappa.\Sigma(a : A).B(a) &\simeq \Sigma(a : \forall\kappa.A).\forall\kappa.B(a[\kappa]) \\ &\simeq \Sigma(a : A).\forall\kappa.B(a)\end{aligned}$$

using the assumption that A is clock invariant in the last equivalence. The case for universal quantification over clocks uses $\forall\kappa.\forall\kappa'.A \simeq \forall\kappa'.\forall\kappa.A$.

In the case of guarded recursive types, first note that if F commutes with clock quantification, so does $\triangleright^\kappa F$. This can be proved using $\forall\kappa'.\, \triangleright^\kappa A \simeq \triangleright^\kappa\forall\kappa'.A$, the left to right map of which maps a to

$$\lambda(\alpha : \kappa).\lambda\kappa'.a[\kappa']\ [\alpha]$$

for α fresh. This map type checks because $\mathsf{TL}(\kappa' : \mathsf{clock}) = \kappa' : \mathsf{clock}$. Using this, we can prove by guarded recursion that X is clock irrelevant as follows

$$\begin{aligned}\forall\kappa'.X &\simeq \forall\kappa'.F(\triangleright^\kappa X) \\ &\simeq F(\forall\kappa'.\, \triangleright^\kappa X) \\ &\simeq F(\triangleright^\kappa\forall\kappa'.X) \\ &\simeq F(\triangleright^\kappa X)\end{aligned}$$

using the guarded recursion assumption in the last line.

In the case of path types, if $x, y : A$ then

$$\begin{aligned}\forall\kappa.(\mathsf{Path}_A(x, y)) &\simeq \mathsf{Path}_{\forall\kappa.A}(\lambda\kappa.x, \lambda\kappa.y) \\ &\simeq \mathsf{Path}_A(x, y)\end{aligned}$$

The first of these equivalences simply swaps the clock and interval argument, the second uses the assumption that A is clock invariant, which means precisely that $\lambda a.\lambda\kappa.a$ is an equivalence, and so preserves path types. □

Example 4.5 uses the following lemma.

Lemma A.1. *Let* A *be a clock irrelevant set and let* $X : \mathsf{P_f}(A)$, $a : A$. *Then* $a \in X$ *is clock irrelevant.*

Proof. The proof is by induction on X, which is valid since statements of the form $\mathsf{IsEquiv}(f)$ are propositions. If $X = \{b\}$ then $a \in X$ is by definition $\mathsf{Path}_A(a, b)$, which is clock irrelevant by Lemma 4.2. If $X = Y \cup Z$ then $a \in X$ is $(a \in Y) \times (a \in Z)$ which is clock-irrelevant by induction and Lemma 4.2. □

We now give a proof of Theorem 4.6. We will write

$$\mathsf{ufld}^\kappa : \mathsf{LTS}^\kappa \to \mathsf{P_f}(A \times \triangleright^\kappa \mathsf{LTS}^\kappa)$$

for the equivalence associated with the guarded recursive type LTS^κ. First note the following.

Lemma A.2. *The following diagram commutes up to path equality.*

$$\begin{array}{ccc} \mathsf{LTS} & \xrightarrow{\mathsf{ufld}} & \mathsf{P_f}(A \times \mathsf{LTS}) \\ {\scriptstyle \mathsf{ev}_\kappa}\downarrow & & \downarrow{\scriptstyle \mathsf{P_f}(A \times f)} \\ \mathsf{LTS}^\kappa & \xrightarrow{\mathsf{ufld}^\kappa} & \mathsf{P_f}(A \times \triangleright^\kappa \mathsf{LTS}^\kappa) \end{array}$$

where $\mathsf{ev}_\kappa \stackrel{def}{=} \lambda x.x[\kappa]$ *and* $f = \lambda x.\lambda(\alpha : \kappa).x[\kappa]$.

Proof. The map ufld is defined to be the composition of the following maps

$$\forall\kappa.\mathsf{ufld}^\kappa : \forall\kappa.\mathsf{LTS}^\kappa \to \forall\kappa.\mathsf{P_f}(A \times \triangleright^\kappa \mathsf{LTS}^\kappa)$$
$$\varphi : \forall\kappa.\mathsf{P_f}(A \times \triangleright^\kappa \mathsf{LTS}^\kappa) \to \mathsf{P_f}(A \times \forall\kappa.\triangleright^\kappa \mathsf{LTS}^\kappa)$$
$$\mathsf{P_f}(A \times \mathsf{force}) : \mathsf{P_f}(A \times \forall\kappa.\triangleright^\kappa \mathsf{LTS}^\kappa) \to \mathsf{P_f}(A \times \forall\kappa.\mathsf{LTS}^\kappa)$$

where $\forall\kappa.\mathsf{ufld}^\kappa(x) = \lambda\kappa.\mathsf{ufld}^\kappa(x[\kappa])$, φ is the witness that $\mathsf{P_f}(A \times (-))$ commutes with clock quantification, and force : $\forall\kappa.\triangleright^\kappa \mathsf{LTS}^\kappa \to \forall\kappa.\mathsf{LTS}^\kappa$ is the inverse to $\lambda x.\lambda(\alpha:\kappa).x$ up to path equality. By the latter, we get the following sequence of path equalities

$$\begin{aligned}\mathsf{P_f}(A \times f) \circ \mathsf{ufld} &= \mathsf{P_f}(A \times \mathsf{ev}_\kappa) \circ \varphi \circ \forall\kappa.\mathsf{ufld}^\kappa \\ &= \mathsf{ev}_\kappa \circ \forall\kappa.\mathsf{ufld}^\kappa \\ &= \mathsf{ufld}^\kappa \circ \mathsf{ev}_\kappa\end{aligned}$$

as desired. □

Proof of Theorem 4.6. Møgelberg and Veltri [40] prove that path equality coincides with bisimilarity for guarded recursive types. Using their results we can prove that, given $x, y : \mathsf{LTS}$

$$\begin{aligned}\mathsf{Path}_{\mathsf{LTS}}(x,y) &\simeq \forall\kappa.\mathsf{Path}_{\mathsf{LTS}^\kappa}(x[\kappa], y[\kappa]) \\ &\simeq \forall\kappa.\mathsf{Bisim}^\kappa(x[\kappa], y[\kappa]) \qquad (10)\end{aligned}$$

where the first equivalence uses functional extensionality for universal quantification over clocks and the second is [40, Corollary 5.4]. Here $\mathsf{Bisim}^\kappa(x,y) = \mathsf{Sim}^\kappa(x,y) \times \mathsf{Sim}^\kappa(y,x)$ where

$$\begin{aligned}\mathsf{Sim}^\kappa(x,y) \simeq \Pi(x' : \triangleright^\kappa\mathsf{LTS}^\kappa, a : A).(a,x') \in \mathsf{ufld}^\kappa(x) \to \\ \exists y' : \triangleright^\kappa\mathsf{LTS}^\kappa.(a,y') \in \mathsf{ufld}^\kappa(y) \times \\ \triangleright(\alpha:\kappa).\mathsf{Sim}^\kappa(x'\,[\alpha], y'\,[\alpha])\end{aligned}$$

We must compare this to bisimilarity of x and y which is defined as

$$\forall\kappa.(\mathsf{Sim}^\kappa_\forall(x,y) \times \mathsf{Sim}^\kappa_\forall(y,x))$$

where

$$\begin{aligned}\mathsf{Sim}^\kappa_\forall(x,y) \simeq \Pi(x' : \mathsf{LTS}, a : A).(a,x') \in \mathsf{ufld}(x) \to \\ \exists y' : \mathsf{LTS}.(a,y') \in \mathsf{ufld}(y) \times \\ \triangleright(\alpha:\kappa).\mathsf{Sim}^\kappa_\forall(x',y')\end{aligned}$$

By an easy guarded recursive argument one can show that $\mathsf{Sim}^\kappa_\forall$ is a reflexive relation, and from this it follows that path equality implies bisimilarity. To prove the other implication it suffices to show that

$$\Pi(x,y : \mathsf{LTS}).\mathsf{Sim}^\kappa_\forall(x,y) \to \mathsf{Sim}^\kappa(x[\kappa], y[\kappa])$$

and this statement is proved by guarded recursion. So suppose $x, y : \mathsf{LTS}$ and $\mathsf{Sim}^\kappa_\forall(x,y)$. Suppose further that $x' : \triangleright^\kappa\mathsf{LTS}^\kappa, a : A$ and $(a,x') \in \mathsf{ufld}^\kappa(x[\kappa])$. By Lemma A.2 this means that $(a,x') \in \mathsf{P_f}(A \times f)(\mathsf{ufld}(x))$ where $f = \lambda x.\lambda(\alpha:\kappa).x[\kappa]$. By [40, Lemma 4.1] there then (merely, i.e. in the sense of $\exists$) exists an $x'' : \mathsf{LTS}$ such that $x' = f(x'')$, i.e.,

$$x' = \lambda(\alpha:\kappa).(x''[\kappa]) \qquad (11)$$

and $(a,x'') \in \mathsf{ufld}(x)$. By the assumption that $\mathsf{Sim}^\kappa_\forall(x,y)$ there then merely exists a $y'' : \mathsf{LTS}$ such that $(a,y'') \in \mathsf{ufld}(y)$ and

$$\triangleright(\alpha:\kappa).\mathsf{Sim}^\kappa_\forall(x'',y''). \qquad (12)$$

Setting $y' = f(y'')$ then, again by [40, Lemma 4.1] $(a,y') \in \mathsf{P_f}(A \times f)(\mathsf{ufld}(y))$ and so by Lemma A.2, $(a,y') \in \mathsf{ufld}^\kappa(y[\kappa])$. It remains to show that

$$\triangleright(\alpha:\kappa).\mathsf{Sim}^\kappa(x'\,[\alpha], y'\,[\alpha])$$

which reduces to

$$\triangleright(\alpha:\kappa).\mathsf{Sim}^\kappa(x''[\kappa], y''[\kappa])$$

using (11) and definition of y'. This follows by guarded recursion from (12). □

We now give a proof of Theorem 4.3. It uses the following lemma establishing the existence of a final $F \circ \triangleright^\kappa$-coalgebra for any endofunctor F.

Lemma A.3. *Let F be an I-indexed endofunctor, then for all κ, the type $\nu^\kappa(F) \stackrel{def}{=} \mathsf{fix}^\kappa(\lambda X.F(\triangleright^\kappa X)) : I \to \mathsf{U}$ has a final $F \circ \triangleright^\kappa$-coalgebra structure, i.e., there is a map $\mathsf{out}^\kappa : \nu^\kappa(F) \to F(\triangleright^\kappa \nu^\kappa(F))$, such that for all maps $f : X \to F(\triangleright^\kappa X)$ the following type is contractible*

$$\Sigma(h : X \to \nu^\kappa(F)).\,\mathsf{out}^\kappa \circ h = F(\triangleright^\kappa(h)) \circ f$$

Proof. The map out^κ is given by the equivalence between a fixpoint in the universe and its unfolding, so we also have an inverse $(\mathsf{out}^\kappa)^{-1}$. The functor $\triangleright^\kappa$ is locally contractible [10] in the sense that the action on morphisms factors as a composition of two maps

$$(X \to Y) \to \triangleright^\kappa(X \to Y) \to (\triangleright^\kappa X \to \triangleright^\kappa Y)$$

and so also the mapping $\lambda h.(\mathsf{out}^\kappa)^{-1} \circ F(\triangleright^\kappa(h)) \circ f$ factors as a composition

$$(X \to \nu^\kappa(F)) \to \triangleright^\kappa(X \to \nu^\kappa(F)) \to (X \to \nu^\kappa(F))$$

Then by uniqueness of fixpoints we get the contractibility of

$$\Sigma(h : X \to \nu^\kappa(F)).\,h = (\mathsf{out}^\kappa)^{-1} \circ F(\triangleright^\kappa(h)) \circ f$$

which in turn is equivalent to our goal by postcomposition with out^κ. □

Proof of Theorem 4.3. We define the coalgebra $\mathsf{out} : \nu(F) \to F\,\nu(F)$ as $F(\mathsf{force}) \circ \mathsf{can}_F^{-1} \circ \forall\kappa(\mathsf{out}^\kappa)$. Given any coalgebra $f : X \to F\,X$, we can extend it to $\tilde{f}^\kappa : X \to F\,(\triangleright^\kappa X)$ for any κ. Then by lemma A.3 we have that for any κ the type

$$\Sigma(h : X \to \nu^\kappa(F)).\,\mathsf{out}^\kappa \circ h = F(\triangleright^\kappa(h)) \circ \tilde{f}^\kappa$$

is contractible. By clock quantification preserving contractibility and commuting with Σ types we have that

$$\Sigma(h : \forall\kappa.X \to \nu^\kappa(F)).\,\forall\kappa.\,(\mathsf{out}^\kappa) \circ h\,[\kappa] = F(\triangleright^\kappa(h\,[\kappa])) \circ \tilde{f}^\kappa$$

is also contractible. Then by $\forall\kappa.X \to \nu^\kappa(F) \simeq X \to \nu(F)$ and that $(\lambda\kappa.F(\triangleright^\kappa(h\,[\kappa])) \circ \tilde{f}^\kappa) = \mathsf{can}_F \circ F(\mathsf{force}^{-1}) \circ F(h)$ we obtain the desired result. More details on the calculations can be found in [38]. □

A.4 Omitted proofs section 5

The equational theory of boundary terms is given by $\equiv_b$ defined in Figure 7. This theory is used in the typing of boundary terms, when typing systems and homogenous compositions. More precisely, the requirement is that for a system of boundary conditions $[\varphi_1\ M_1 \dots \varphi_m\ M_m]$ to be wellformed, it must be the case that $\varphi_i \wedge \varphi_j$ implies $M_i \equiv_b M_j$, for any i, j. Similarly, for $\mathsf{hcomp}^j_{\mathsf{H}\,\delta}[\psi \mapsto M']\,M'_0$ to be well typed, we must have $M'[0/j] \equiv_b M'_0$.

Proof of Lemma 5.2. Throughout we assume that $\Gamma \vdash \delta : \forall\kappa.\Delta$ unless otherwise specified. Through an induction on the structure of the boundary terms, we prove the following more general typing:

$$\Gamma', \hat{\gamma} : \forall\overline{\kappa}.\,\hat{\Gamma}[\delta[\overline{\kappa}], \gamma\,[\kappa]] \vdash (\!|M|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} : D[\lambda\overline{\kappa}.M[\gamma[\overline{\kappa}], \overline{x[\overline{\kappa}]}, \overline{i}, \hat{\gamma}[\overline{\kappa}]]]$$

where Γ' is defined as follows

$$\Gamma' \stackrel{\text{def}}{=} \Gamma, \gamma : \forall\overline{\kappa}.\Gamma_i[\delta[\overline{\kappa}]], \overline{x} : \overline{\forall\overline{\kappa}.\Xi_i[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]] \to \mathsf{H}\,(\delta[\overline{\kappa}])}, \overline{y} : \overline{(\xi : \forall\overline{\kappa}.\Xi[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]]) \to D[\lambda\overline{\kappa}.x[\overline{\kappa}](\xi[\overline{\kappa}]))}, \overline{i} : \Psi_i, \varphi_i$$

and M is assumed to be a boundary term of type $\mathsf{H}\,\delta$ in context

$$\delta : \Delta, \gamma : \Gamma_i[\delta], \overline{x} : \Xi_i[\delta, \gamma] \to \mathsf{H}\,\delta, \overline{i} : \Psi_i, \varphi_i, \hat{\gamma} : \hat{\Gamma}[\delta, \gamma]$$

The desired typing is then the case where $\hat{\Gamma}$ above is empty, since we have that φ_k implies that $\lambda\overline{\kappa}.\mathsf{con}_i(\dots)$ reduces to $\lambda\overline{\kappa}.\,M_k$. The $\hat{\Gamma}$ crops up because the interpretation of constructors adds a $\xi : \forall\overline{\kappa}.\Xi[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]]$ to the context while the hcomp case adds an interval variable and a face restriction to the context for the system of the composition. In the very first step, unfolding the list of partial elements adds a face restriction as well. Concretely we proceed as follows in each case. Also, we write σ for the substitution $[\delta[\overline{\kappa}]/\delta, \gamma[\overline{\kappa}]/\gamma, \overline{x[\overline{\kappa}]}/\overline{x}, \hat{\gamma}[\overline{\kappa}]/\hat{\gamma}]$ and τ for the same but without the $\overline{x}$ component.

- Assume $M = x_j\,\overline{u}$. From the typing assumptions on M we have that

  $$\delta : \Delta, \gamma : \Gamma_i, \overline{x} : \overline{\Xi_i[\delta, \gamma] \to \mathsf{H}\,\delta}, \overline{i} : \Psi, \varphi_i, \hat{\Gamma} \vdash u : \Xi_{i,j}$$

  This means that $\xi = \lambda\overline{\kappa}.\overline{u}\tau$ has type $\forall\overline{\kappa}.\Xi_{i,j}[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]]$, which is exactly the input to y_j, so that $y_j\ \xi$ has type $D[\lambda\overline{\kappa}.\,x[\overline{\kappa}](\overline{u}\tau)]$, as desired.
- Assume $M = \mathsf{con}_j(\overline{t}, \overline{\lambda\xi.M'}, \overline{r})$. By inductive hypothesis we have that

  $$\Gamma', \hat{\gamma} : \forall\overline{\kappa}.\,\hat{\Gamma}[\delta[\overline{\kappa}], \gamma[\overline{\kappa}]], \xi : \forall\overline{\kappa}.\Xi_{j,k}[\delta[\overline{\kappa}], \overline{t}\tau] \vdash (\!|M'_k|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},(\hat{\gamma},\xi)} : D[\lambda\overline{\kappa}.M'_k\sigma]$$

  This gives us the input necessary to apply $\mathscr{E}_j$. The $\overline{t}$ family has a similar typing structure to $\overline{u}$ in the previous example so we get $\overline{t'} = \lambda\overline{\kappa}.\overline{t}\tau$ of type $\forall\overline{\kappa}.\Gamma_j[\delta[\overline{\kappa}]]$. We obtain maps

  $$S_k : \forall\overline{\kappa}.\,\Xi_{j,k}[\delta[\overline{\kappa}], \overline{t'}[\overline{\kappa}]] \to \mathsf{H}\,(\delta[\overline{\kappa}])$$
  $$S_k = \lambda\overline{\kappa}.\lambda\xi.M'_k\sigma$$

  directly from the typing assumptions and by inductive hypothesis we obtain maps

  $$R_k : (\xi : \forall\overline{\kappa}.\Xi_{j,k}[\delta[\overline{\kappa}], \overline{t'}[\overline{\kappa}]]) \to D[\lambda\overline{\kappa}.S_k[\overline{\kappa}](\xi[\overline{\kappa}])]$$
  $$R_k = \lambda\xi.(\!|M'_k|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},(\hat{\gamma},\xi)}$$

  This means we have the required data to apply $\mathscr{E}_j$, and by the definition of clock abstracted elimination lists we have that $\mathscr{E}_j[\overline{t'}, \overline{S}, \overline{R}, \overline{r}]$ inhabits D over $\lambda\overline{\kappa}.M\sigma$.
- Assume $M = \mathsf{hcomp}^j_{\mathsf{H}\,\delta}[\psi \mapsto M']\,M'_0$. By inductive hypothesis and the typing assumptions for the hcomp to be well formed we have that $(\!|M'_0|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}$ inhabits D over $\lambda\overline{\kappa}.\,M'_0\sigma$, and $(\!|M'|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma},j}$ inhabits D over $\lambda\overline{\kappa}.\,M'\sigma$ in the context extended by $j : \mathbb{I}$ and restricted by ψ. The v term provides a path between $\lambda\overline{\kappa}.\,M'_0\sigma$ and its composition with $\lambda\overline{\kappa}.\,M'\sigma$, which means that the composition in $D[vj]$ provides a term over $\lambda\overline{\kappa}.\,M\sigma$ as desired. Finally, for the composition to be well typed, we must verify that

  $$(\!|M'|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma},j}[0/j] \equiv (\!|M'_0|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}$$

  An easy induction shows that the left hand side equals $(\!|M'[0/j]|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}$ and since $M'_0 \equiv_b M'[0/j]$, this follows from Lemma A.4 below.
- Assume $M = [\varphi_0\ M_0, \dots, \varphi_{n_i}\ M_{n_i}]$. In this case we have the following typing by inductive hypothesis:

  $$\Gamma', \varphi_k \vdash (\!|M_k|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} : D[\lambda\overline{\kappa}.M_k\sigma]$$

  Finally, in order to conclude that

  $$[\varphi_1\ (\!|M_1|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}, \dots, \varphi_{n_i}\ (\!|M_{n_i}|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}]$$

  defines a system, and so a partial element of $D[\lambda\overline{\kappa}.M\sigma]$, we must show that on faces of the form $\varphi_j \wedge \varphi_k$ the judgemental equality

  $$(\!|M_j|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} \equiv (\!|M_k|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}$$

  holds. Since on this face $M_j \equiv_b M_k$, this follows from Lemma A.4 below. □

Lemma A.4. *If* $\Gamma \vdash M \equiv_b N$ *then* $\Gamma \vdash (\!|M|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} \equiv (\!|N|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}$

Proof. The proof is by induction on the proof of $M \equiv_b N$. The interesting case is the reduction of a constructor to its boundary, which requires showing that $\varphi_i \equiv \top$ implies

$$(\!|\mathsf{con}_\ell(\overline{t}, \overline{\lambda\xi.\,M}, \overline{r})|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} \equiv (\!|N_i(\overline{t}, \overline{\lambda\xi.\,M}, \overline{r})|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}} \quad (13)$$

The left hand side of this equation is $\mathscr{E}_\ell[\overline{t'}, \overline{S}, \overline{R}, \overline{r}]$, which under the assumption that $\varphi_i \equiv \top$ equals

$$(\!|N_i|\!)^{\delta}_{\mathscr{E},\overline{x}\mapsto\overline{y},\hat{\gamma}}[\overline{t'}, \overline{S}, \overline{R}, \overline{r}/\overline{\gamma}, \overline{x}, \overline{y}, \overline{i}]$$

Rules for equality of boundary terms

$$\frac{\Gamma \vdash \overline{u} \equiv \overline{v}}{\Gamma \vdash x\ \overline{u} \equiv_b x\ \overline{v}} \qquad \frac{\Gamma \vdash \overline{t} \equiv \overline{t'} \qquad \Gamma \vdash \overline{r} \equiv \overline{r'} \qquad \forall i.(\Gamma, \overline{\xi_i} \vdash M_i \equiv_b M_i')}{\Gamma \vdash \mathsf{con}_\ell(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r}) \equiv_b \mathsf{con}_\ell(\overline{t'}, \overline{\lambda\xi.\, M'}, \overline{r'})}$$

$$\frac{\Gamma, j, \varphi \vdash M \equiv_b M' \qquad \Gamma \vdash M_0 \equiv_b M_0'}{\Gamma \vdash \mathsf{hcomp}^j_{\mathsf{H}\,\delta}\ [\varphi \mapsto M]\ M_0 \equiv_b \mathsf{hcomp}^j_{\mathsf{H}\,\delta}\ [\varphi \mapsto M']\ M_0'} \qquad \frac{e_\ell = [\varphi_1\ N_1 \ldots \varphi_m\ N_m] \qquad \Gamma \vdash \varphi_i \equiv \top}{\Gamma \vdash \mathsf{con}_\ell(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r}) \equiv_b N_i(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r})}$$

$$\frac{\Gamma \vdash \varphi \equiv \top}{\Gamma \vdash \mathsf{hcomp}^j_{\mathsf{H}\,\delta}\ [\varphi \mapsto M]\ M_0 \equiv_b M[1/j]}$$

Substitution operation

$$(x_j\ \overline{u})(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r}) = M_j[\overline{u}[\overline{t}, \overline{r}]/\overline{\xi_j}]$$

$$\mathsf{con}_\ell(\overline{t'}, \overline{\lambda\xi'.\, M'}, \overline{r'})(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r}) = \mathsf{con}_\ell(\overline{t'}[\overline{t}, \overline{r}], \overline{\lambda\xi'.\, M'(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r}, \overline{\xi'})}, \overline{r'}[\overline{r}])$$

$$(\mathsf{hcomp}^j_{\mathsf{H}\,\delta}\ [\varphi \mapsto N]\ N_0)(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r}) = \mathsf{hcomp}^j_{\mathsf{H}\,\delta}\ [\varphi \mapsto (N(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r}, j))]\ (N_0(\overline{t}, \overline{\lambda\xi.\, M}, \overline{r}))$$

Figure 7. The equational theory of boundary terms is the least equivalence relation generated by the rules above. The rule for reducing a constructor to its boundary uses the substitution also defined above.

It thus suffices to prove equality of the above with the right hand side of (13). We omit the straight forward induction proof of this substitution property. □

Details of higher truncation. Recall that we defined a map $\alpha : \forall\kappa.\, \|A\|_n \to \|\forall\kappa.\, A\|_n$. For $f : \forall\kappa.\, \mathbb{S}^{n+1} \to \|A\|_n$ we let $f' = \lambda s.\, \alpha(\lambda\kappa.\, (f\ [\kappa])(s))$ and $p_s : \lambda\kappa.\, s\ [\kappa_0] = s$ is the path extracted from clock irrelevance of $\mathbb{S}^{n+1}$. Note that from the definition of α, we get the following reductions:

$$\begin{aligned}
\alpha(\lambda\kappa.\mathsf{in}(a\ [\kappa])) &\equiv \mathsf{in}(a)\\
\alpha(\lambda\kappa.\mathsf{hub}(f\ [\kappa])) &\equiv \mathsf{hub}(f')\\
\alpha(\lambda\kappa.\,\mathsf{spoke}(s\ [\kappa], f\ [\kappa], i)) &\equiv\\
\mathsf{hcomp}^j[(i=0) &\mapsto \alpha(\lambda\kappa.\,(f\ [\kappa])((p_s\ j)\ [\kappa])),\\
(i=1) &\mapsto \mathsf{hub}(f')]\\
&\mathsf{spoke}(s\ [\kappa_0], f', i)
\end{aligned}$$

For ease of reasoning, we write out the reductions for the canonical map β:

$$\begin{aligned}
\beta(\mathsf{in}(a)) &\equiv \lambda\kappa.\mathsf{in}(a\ [\kappa])\\
\beta(\mathsf{hub}(f)) &\equiv \lambda\kappa.\mathsf{hub}(\lambda s.\,(\beta(f(s)))\ [\kappa])\\
\beta(\mathsf{spoke}(s, f, i)) &\equiv \lambda\kappa.\,\mathsf{spoke}(s, \lambda s.\,(\beta(f(s)))\ [\kappa], i)
\end{aligned}$$

We show that the two maps are inverse to one another by induction, leaving out the trivial first case. For hub we reason as follows:

$$\begin{aligned}
\alpha(\beta(\mathsf{hub}(f))) &\equiv \alpha(\lambda\kappa.\mathsf{hub}(\lambda s.\,(\beta(f(s)))\ [\kappa]))\\
&\equiv \mathsf{hub}(\lambda s.\,\alpha(\lambda\kappa.\,(\beta(f(s)))\ [\kappa]))\\
&\equiv \mathsf{hub}(\lambda s.\,\alpha(\beta(f(s))))\\
&= \mathsf{hub}(\lambda s.\,f(s))\\
&\equiv \mathsf{hub}(f)
\end{aligned}$$

$$\begin{aligned}
\beta(\alpha(\lambda\kappa.\mathsf{hub}(f\ [\kappa]))) &\equiv \beta(\mathsf{hub}(\lambda s.\,\alpha(\lambda\kappa.\,(f\ [\kappa])(s))))\\
&\equiv \lambda\kappa'.\,\mathsf{hub}(\lambda s.\,\beta(\alpha(\lambda\kappa.\,(f\ [\kappa])(s)))\ [\kappa'])\\
&= \lambda\kappa'.\,\mathsf{hub}(\lambda s.\,(\lambda\kappa.\,(f\ [\kappa])(s))\ [\kappa'])\\
&\equiv \lambda\kappa.\,\mathsf{hub}(\lambda s.\,(f\ [\kappa])(s))\\
&\equiv \lambda\kappa.\,\mathsf{hub}(f\ [\kappa])
\end{aligned}$$

Note that the only non-judgemental equality is the inductively justified cancellation of the compositions i.e., the path $q : \lambda s.\,\alpha(\beta(f(s))) = f$ and $r : \lambda s.\,\beta(\alpha(\lambda\kappa.\,(f\ [\kappa])(s))) = \lambda s.\,\lambda\kappa.\,(f\ [\kappa])(s)$ under the hub constructor.

We now supply the first calculation for the spoke constructor:

$$\begin{aligned}
&\alpha(\beta(\mathsf{spoke}(s, f, i))) \equiv \alpha(\lambda\kappa.\,\mathsf{spoke}(s, \lambda s.\,(\beta(f(s)))\ [\kappa], i))\\
&\equiv \mathsf{hcomp}^j[(i=0) \mapsto \alpha(\lambda\kappa.\,(\beta(f((p_{\mathsf{const}\,s}\ j\ [\kappa]))))\ [\kappa]),\\
&\qquad (i=1) \mapsto \mathsf{hub}(\lambda s.\,\alpha(\lambda\kappa.\,(\beta(f(s)))\ [\kappa]))]\\
&\qquad \mathsf{spoke}(s, \lambda s.\,\alpha(\lambda\kappa.\,(\beta(f(s)))\ [\kappa]), i)\\
&\qquad = \mathsf{spoke}(s, \lambda s.\,\alpha(\lambda\kappa.\,(\beta(f(s)))\ [\kappa]), i)\\
&\qquad \equiv \mathsf{spoke}(s, \lambda s.\,\alpha(\beta(f(s))), i)\\
&\qquad = \mathsf{spoke}(s, f, i)
\end{aligned}$$

For us to apply induction with the above path it would need to reduce strictly to the recursive call at s, $\lambda j.\, q(j)(s)$,

on $i = 0$ and the hub case, $\lambda j.\, \mathsf{hub}(\lambda s.\, q(j)(s))$, on $i = 1$. This is too tall an ask, but luckily it is sufficient for our purposes that we can show such reductions up to a path. The path used for the induction is then defined by an appropriate composition as in the definition of the map α.

The calculation above uses two non-trivial paths. The first is the reduction of a composition to its base, which we note restricts to $\lambda j.\, \alpha(\lambda \kappa.\, (\beta(f((p_{\mathsf{const}\, s}\ j)\ [\kappa]))))\ [\kappa])$ and reflexivity on $i = 0$ and $i = 1$ respectively. The second is exactly an application of q from above to the function input, or more concretely the path $\lambda j.\mathsf{spoke}(s, \lambda s.\, q(j)(s)), i)$. The reductions of spoke then mean that this reduces to exactly the paths we are looking for. Since p_{const} is given by clock irrelevance at a constant function it is path equal to reflexivity by lemma A.7. Cancelling this path at $i = 0$ and reflexivity at $i = 1$ with a composition then yields the desired path.

$$
\begin{aligned}
&\beta(\alpha(\lambda\kappa.\, \mathsf{spoke}(s\ [\kappa], f\ [\kappa], i)))\\
&\equiv \beta(\mathsf{hcomp}^j\,[(i=0) \mapsto \alpha(\lambda\kappa.\, (f\ [\kappa])((p_s\ j)\ [\kappa])),\\
&\qquad\qquad (i=1) \mapsto \mathsf{hub}(\lambda s.\, \alpha(\lambda\kappa.\, (f\ [\kappa])(s))))]\\
&\qquad\qquad \mathsf{spoke}(s\ [\kappa_0], \lambda s.\, \alpha(\lambda\kappa.\, (f\ [\kappa])(s))), i))\\
&\equiv \mathsf{hcomp}^j\,[(i=0) \mapsto \beta(\alpha(\lambda\kappa.\, (f\ [\kappa])((p_s\ j)\ [\kappa]))),\\
&\qquad\qquad (i=1) \mapsto \beta(\mathsf{hub}(\lambda s.\, \alpha(\lambda\kappa.\, (f\ [\kappa])(s)))))]\\
&\qquad\qquad \beta(\mathsf{spoke}(s\ [\kappa_0], \lambda s.\, \alpha(\lambda\kappa.\, (f\ [\kappa])(s))), i)))\\
&\equiv \mathsf{hcomp}^j\,[(i=0) \mapsto \beta(\alpha(\lambda\kappa.\, (f\ [\kappa])((p_s\ j)\ [\kappa]))),\\
&\qquad\qquad (i=1) \mapsto \lambda\kappa'.\, \mathsf{hub}(\lambda s.\, \beta(\alpha(\lambda\kappa.\, (f\ [\kappa])(s)))\ [\kappa'])]\\
&\qquad\qquad \lambda\kappa'.\mathsf{spoke}(s\ [\kappa_0], \lambda s.\, \beta(\alpha(\lambda\kappa.\, (f\ [\kappa])(s))))\ [\kappa'], i)\\
&\qquad = \lambda\kappa'.\mathsf{spoke}(s\ [\kappa_0], \lambda s.\, \beta(\alpha(\lambda\kappa.\, (f\ [\kappa])(s))))\ [\kappa'], i)\\
&\qquad = \lambda\kappa'.\mathsf{spoke}(s\ [\kappa'], \lambda s.\, \beta(\alpha(\lambda\kappa.\, (f\ [\kappa])(s))))\ [\kappa'], i)\\
&\qquad = \lambda\kappa.\mathsf{spoke}(s\ [\kappa], f\ [\kappa], i)
\end{aligned}
$$

This time the boundary obligation is that on $i = 0$ the above must reduce to $\lambda j.\lambda\kappa.\, r(j)(s\ [\kappa])\ [\kappa]$ and on $i = 1$ it must reduce to $\lambda j.\, \lambda\kappa.\, \mathsf{hub}(\lambda s.\, r(j)(s\ [\kappa])\ [\kappa])$. The calculation is a composition of three non-trivial paths. We first reduce the composition to its base, which reduces to reflexivity on $i = 1$ and $\lambda j.\, \beta(\alpha(\lambda\kappa.\, (f\ [\kappa])((p_s\ j)\ [\kappa])))$ on $i = 0$. The next path is again the path extracted from clock irrelevance, this time the one that shows that $s\ [\kappa_0] = s\ [\kappa']$, which is exactly the inverse application of clock irrelevance. This reduces to reflexivity on $i = 1$ and the path $\lambda j.\, \beta(\alpha(\lambda\kappa.\, (f\ [\kappa])((p_s^{-1}\ j)\ [\kappa])))$ on $i = 0$. At this point the composite path has the shape $\mathsf{refl} \circ \lambda j.\, \beta(\alpha(p'(j)) \circ \lambda j.\, \beta(\alpha(p'^{-1}(j))$ on $i = 0$ and refl^3 on $(i = 1)$, meaning that it reduces to refl up to a path in either case. The last path is then the inductively obtained path

$$\lambda j.\, \lambda\kappa.\mathsf{spoke}(s\ [\kappa], \lambda s.\, r(j)(s)\ [\kappa], i).$$

The desired reduction now again follows from an application of the equalities governing the behavior of spoke.

Proving Theorem 5.4. We proceed by constructing a map

$$(h : \forall\kappa.\, \mathsf{H}\,\delta) \to \forall\kappa.\, h\,[\kappa] = h\,[\kappa_0].$$

To do this we first show a slightly modified induction under clocks principle with constant δ and $\Gamma_{(-)}$ parameters. Using this modified principle we construct the map in two stages: first we provide a candidate case for each constructor and secondly we show that the candidate terms satisfy the appropriate boundary condition up to path equality. The latter will allow us to rectify the terms given by the former constructions via a composition to obtain the input for the modified principle, thus allowing us to define a map of the desired type. The new induction principle needs a modified version of the boundary condition, given by transporting the usual one along the equivalence between the two, which means that we need to, at each stage, cohere with the transport in the earlier stages.

Before beginning the proof we need to introduce some notation and prove a small lemma about the structure of HIT constructors.

Lemma A.5. *Let $H\,\delta$ be a HIT with constructors taking input $\gamma^0 : \Gamma_i^0[\delta], \gamma^1 : \Gamma_i^1[\delta], \ldots, \quad x : \Xi_i^0[\delta, \overline{\gamma}] \to H\,\delta, \ldots$ and $\overline{i} : \Psi_i$ and boundary conditions $\varphi_i \vdash e_i$. Then there exists an equivalent HIT $H'\delta$ with constructors taking input of the form*

$$\gamma : \Gamma_i[\delta], x : \Xi_i[\delta, \overline{\gamma}] \to H'\delta \text{ and } \overline{i} : \Psi_i$$

with boundary conditions $\varphi_i \vdash e_i$ where φ_i is of the form $\bigvee(i = 0) \vee (i = 1)$ with i ranging over all variables in Ψ. We say that HITs satisfying these three criteria are of reduced form.

Proof. Modifying the Γ and $\Xi \to \mathsf{H}\,\delta$ input is trivial, simply take $\Gamma_i[\delta]$ to be an iterated Σ type consisting of the Γ_i^j and $\Xi_i[\delta, \gamma] \stackrel{\text{def}}{=} \Xi_i^j[\delta, \gamma] + \Xi_i^j[\delta, \gamma] + \ldots$. This procedure clearly yields an equivalent HIT, so we make this assumption freely.

We achieve boundaries of the desired shape by adding constructors to specify the full boundary, noting that $\bigvee(i = 0) \vee (i = 1)$ as above is the second largest element of the face lattice specifying the entire boundary of a cube but not the interior. We call it the total face relative to Ψ. Let con_i be a constructor of $\mathsf{H}\,\delta$ with boundary extent φ_i and interval input Ψ_i. We add a number of constructors to $\mathsf{H}'\,\delta$ recursively in the following way: Write φ_i in disjunctive normal form. For each $j \in \Psi_i$ if $(j = 0)$ and $(j = 1)$ appear as disjuncts of φ_i we add con_i as is. Say that $(j = 0)$ is missing; we then add a constructor $\mathsf{con}_i^{(j=0)}$ with the same Γ and Ξ as con_i. This new constructor then has interval input Ψ with the j variable deleted. Now consider the face $\varphi_i \wedge (j = 0)$. If this is the total face relative to Ψ without j, we define the boundary term of $\mathsf{con}_i^{(j=0)}$ to be the boundary of con_i restricted to $(j = 0)$ and proceed with the next interval variable in Ψ. If it is not the total face and is missing for instance $k \in \Psi$, we repeat the procedure, defining a new constructor $\mathsf{con}_i^{(j=0)\wedge(k=0))}$ in the

same way. Having recursively defined this new constructor, we use it for the boundary at $k = 0$.

It is immediate that we can define mutually inverse maps between H δ and H' δ using their respective elimination principles. □

The point of the modified boundary shape is that constructors with such boundaries exactly correspond to constructors for heterogeneous, iterated path types. This means that for HITs of the above form we can treat the end result $(i : \Psi) \to A[\varphi \mapsto e]$ as a proper type, allowing it to appear in e.g., Σ-types. We treat it as we would path types, introducing terms of them by abstraction and eliminating from them by application. As for path types we can compose iterated paths, and we record this fact in the following lemma:

Lemma A.6.

$$\frac{\Gamma \vdash p : \Pi(i : \Psi).\, A[\varphi \mapsto u] \qquad \Gamma, (i : \Psi), \varphi \vdash q : u = v}{\Gamma \vdash (i.\, q)^*\, p : \Pi(i : \Psi).\, A[\varphi \mapsto v]}$$

Proof. $(i.\, q)^*\, p \stackrel{\text{def}}{=} \lambda i.\, \mathsf{hcomp}^j\, [\varphi \mapsto q\, j]\, (p\, i)$ □

We define the following types in context where $\delta : \Delta$, and D a family over $\forall\kappa.\mathsf{H}\,\delta$, where $\overline{u_{<\ell}} : \mathscr{E}_{<\ell}(\delta, D)$ as specified below, and $\delta : \forall\kappa.\Gamma_\ell(\delta)$:

$$\begin{aligned}
&\mathscr{P}_\ell(\delta, D, \overline{u_{<\ell}}, \gamma) \stackrel{\text{def}}{=} \\
&\quad \Pi(x : (\forall\kappa.\, \Xi_\ell[\delta, \gamma\,[\kappa]] \to \mathsf{H}\,\delta)) \\
&\qquad (y : \Pi(\xi : \forall\kappa.\, \Xi_\ell[\delta, \gamma\,[\kappa]]).D[\lambda\kappa.x\,[\kappa](\xi\,[\kappa])]) \\
&\qquad (i : \Psi_\ell). \\
&\qquad D[\lambda\kappa.\, \mathsf{con}_\ell(\gamma\,[\kappa], x\,[\kappa], i)][\varphi_\ell \mapsto (\!|e_\ell|\!)^{\delta}_{\overline{u_{<\ell}}, x \mapsto y}] \\
&\mathscr{E}_\ell(\delta, D, \overline{u_{<\ell}}) \stackrel{\text{def}}{=} \Pi(\gamma : \forall\kappa.\Gamma_\ell[\delta]).\, \mathscr{P}_\ell(\delta, D, \overline{u_{<\ell}}, \gamma) \\
&\mathscr{E}'_\ell(\delta, D, \overline{u_{<\ell}}) \stackrel{\text{def}}{=} \Pi(\gamma : \Gamma_\ell[\delta]).\, \mathscr{P}_\ell(\delta, D, \overline{u_{<\ell}}, \lambda_.\, \gamma)
\end{aligned}$$

where

$$\begin{aligned}
\mathscr{E}(\delta, D) = \Sigma(u_{\ell_0} : \mathscr{E}_{\ell_0}(\delta)).\, \Sigma(u_{\ell_1} : \mathscr{E}_{\ell_1}(\delta, u_{\ell_0})).\, \\
\ldots\, \mathscr{E}_{\ell_n}(\delta, (u_{\ell_0}, u_{\ell_1}, \ldots, u_{\ell_{n-1}}))
\end{aligned}$$

and $\mathscr{E}_{<\ell}(\delta, D)$ is the prefix of the above iterated Σ types containing fields only for the labels below ℓ. The above definition is well-founded because $\mathscr{E}_\ell(\delta, -)$ only refers to $\mathscr{E}_{\ell'}(\delta, -)$ for labels $\ell' < \ell$.

Lemma A.7. *For each ℓ we have*

$$\begin{aligned}
\mathsf{cirr}_{\Gamma_\ell} &: \Pi(\gamma : \forall\kappa.\Gamma_\ell[\delta]).\, \lambda_.\, \gamma\,[\kappa_0] = \gamma \\
\mathsf{cirr{-}coh}_{\Gamma_\ell} &: \Pi(\gamma : \Gamma_\ell[\delta]).\, \mathsf{refl} = \mathsf{cirr}_{\Gamma_\ell}\, (\lambda_.\, \gamma)
\end{aligned}$$

Proof. From clock irrelevance of Γ_ℓ we get a proof that the constant map from $\Gamma_\ell[\delta]$ to $\forall\kappa.\Gamma_\ell[\delta]$ has a right inverse. Moreover it has application to κ_0 as a left inverse with reflexivity as the proof. From this we obtain a proof that the constant map is an half adjoint equivalence, from which we can project $\mathsf{cirr}_{\Gamma_\ell}$ and $\mathsf{cirr{-}coh}_{\Gamma_\ell}$. □

Lemma A.8. *For each ℓ, δ and $u_{<\ell}$ we have an equivalence of types $f_\ell : \mathscr{E}'_\ell(\delta, D, u_{<\ell}) \simeq \mathscr{E}_\ell(\delta, D, u_{<\ell})$.*

Proof. We will use $\mathsf{cirr}_{\Gamma_\ell}$ to transport $\mathscr{P}_\ell(\delta, D, u_{<\ell}, \lambda\_.\, \gamma\,[\kappa_0])$ to $\mathscr{P}_\ell(\delta, D, u_{<\ell}, \gamma)$. Concretely we define

$$f_\ell(u') \stackrel{\text{def}}{=} \lambda\gamma.\, \mathsf{cirr}_{\Gamma_\ell}(\gamma)^*(u'\ (\gamma\,[\kappa_0]))$$

□

We then define $\mathscr{E}'(\delta, D)$ as the iterated sigma type

$$\begin{aligned}
\Sigma(u_{\ell_0} : \mathscr{E}'_{\ell_0}(\delta, D)).\Sigma(u_{\ell_1} : \mathscr{E}'_{\ell_1}(\delta, D, f_{\ell_0}(u_{\ell_0})).\, \\
\ldots\, \mathscr{E}'_{\ell_n}(\delta, D, f(\overline{u_{<\ell_n}})
\end{aligned}$$

where we wrote $f(\overline{u_{<\ell_n}})$ in place of

$$f_{\ell_0}(u_{\ell_0}), f_{\ell_1}(u_{\ell_1}), \ldots, f_{\ell_{n-1}}(u_{\ell_{n-1}})$$

as we will do going forward. In fact the family f_ℓ can be collected into an equivalence of type $\mathscr{E}'(\delta, D) \simeq \mathscr{E}(\delta, D)$ which also restricts to the $< \ell$ case.

Lemma A.9. *Let $\delta : \Delta$, and $h : \forall\kappa.\, H(\delta) \vdash D$ type and let $t : \forall\kappa.\, H\,\delta$.*

(a) *Elimination under a single clock allows us to produce a term of the type* $\mathsf{Elim}(\delta, D, t) \stackrel{\text{def}}{=} \mathscr{E}(\delta, D) \to D[t]$.

(b) *Clock elimination with constant Γ_ℓ parameter allows us to produce a term of the type* $\mathsf{Elim}^{\mathsf{const}}(\delta, D, t) \stackrel{\text{def}}{=} \mathscr{E}'(\delta, D) \to D[t]$

Proof. Case (a) is direct from typing of elimination under a single clock, case (b) follows by composing (a) with the equivalence f. □

In the following we will fix $D[t]$ to be $\forall\kappa.t\,[\kappa_0] = t\,[\kappa]$.

Lemma A.10. *For each ℓ we can type b_ℓ as shown:*

$$\begin{aligned}
\mathscr{T}_\ell(\delta) \stackrel{\text{def}}{=} &\Pi(\gamma : \Gamma_\ell[\delta]) \\
&(x : \forall\kappa.\, \Xi_\ell[\delta, \gamma] \to H\,\delta) \\
&(y : \Pi(\xi : \forall\kappa.\, \Xi_\ell[\delta, \gamma]).D[\lambda\kappa.x\,[\kappa](\xi\,[\kappa])]). \\
&(i : \Psi_\ell). \\
&D[\lambda\kappa.\, \mathsf{con}_\ell(\gamma, x\,[\kappa], i)] \\
&\quad [\varphi_\ell \mapsto \lambda\kappa.\, \lambda j.\, e_\ell[\gamma, \lambda\xi.\, y\,(\lambda_.\, \xi)\,[\kappa]\, j, i]]
\end{aligned}$$

$$\delta : \Delta \vdash b_\ell \stackrel{def}{=} \lambda\gamma\, x\, y\, i\, \kappa\, j.\, \mathsf{con}_\ell(\gamma, \lambda\xi.\, y\,(\lambda\_.\, \xi)\,[\kappa]\, j, i) : \mathscr{T}_\ell(\delta)$$

Proof. Follows directly by the typing rule for con_ℓ. □

Note that $\mathscr{T}_\ell(\delta)$ differs from $\mathscr{E}'_\ell(\delta, D, \overline{u_{<l}})$ only in the boundary of the final result. We bridge this gap with lemma A.11.

Lemma A.11. *The terms defined in lemma A.10 satisfy the boundary conditions of lemma A.9 (b) up to path equality, i.e., for each ℓ we have a term* lemD_ℓ *of type*

$$\lambda\kappa.\, \lambda j.\, e_\ell[\gamma, \lambda\xi.\, y\,(\lambda\_.\, \xi)\,[\kappa]\, j, i] = (\!|e_\ell|\!)^{\lambda\_.\,\delta}_{\overline{v_{<\ell}}, x \mapsto y}[\lambda\_.\, \gamma]$$

in the appropriate context, and where each v_ℓ is defined as

$$\begin{aligned} v_\ell &: \mathscr{E}(\delta, D, \overline{v_{<\ell}}) \\ v_\ell &\stackrel{def}{=} f_\ell(g_\ell(b_\ell)) \\ g_\ell &: \mathscr{T}_\ell(\delta) \to \mathscr{E}'(\delta, D, \overline{v_{<l}}) \\ g_\ell(t) &\stackrel{def}{=} \lambda\gamma.\lambda x.\lambda y.(i.\mathsf{lemD}_\ell)^*(t\ \gamma\ x\ y) \end{aligned}$$

Proof. To arrive at the desired conclusion we need a generalized version of the lemma which keeps track of how the proof in each case coheres with the earlier stages. We prove the following:

$$\begin{aligned} \Omega_\ell \stackrel{\text{def}}{=} \delta &: \Delta, \gamma : \Gamma_\ell, x : \forall\kappa.\,\Xi_\ell \to \mathsf{H}\,\delta, \\ & y : \Pi(\xi : \forall\kappa.\,\Xi_\ell).D[\lambda\kappa.\,x\,[\kappa]\,(\xi\,[\kappa])], \\ & y_K : \Pi(\xi : \Xi_\ell).D[\lambda\kappa.\,x\,[\kappa]\,\xi], \\ & \tilde{y} : \Pi(\xi : \Xi_\ell).y_K\,\xi = y(\lambda\_.\,\xi), \\ & i : \Psi_\ell, \varphi_\ell \end{aligned}$$

$$\begin{aligned} \Omega_\ell, \hat{\Gamma} \vdash\ & \mathsf{lemDg}_\ell(M) \\ & : (\lambda\kappa.\,\lambda j.\,M[\lambda\xi.\,y_K\,\xi\,[\kappa]\,j/x]) \\ & = (\!|M|\!)^{\lambda\_.\,\delta}_{\overline{v_{<\ell}}, x\mapsto y, \hat{\gamma}}[\lambda\_.\,\gamma/\gamma, \lambda\_.\,\hat{\gamma}/\hat{\gamma}] \end{aligned}$$

lemD_ℓ is then defined as

$$\mathsf{lemDg}_\ell(e_\ell)[\lambda\xi.\,y\,(\lambda\_.\xi)/y_K, \lambda\xi.\,\mathsf{refl}/\tilde{y}]$$

we will write σ for $[\lambda\xi.\,y_K\,\xi\,[\kappa]\,j/x]$ and τ for $[\lambda\_.\,\gamma/\gamma, \lambda\_.\,\hat{\gamma}/\hat{\gamma}]$. The extra y_K and $\tilde{y}$ parameters provide what to do for the case $M = x_j\,\overline{u}$ where we will apply $\tilde{y}$ to $\overline{u}$ to obtain the necessary equality between the applications of y_K and y. This allows us to derive that $\mathsf{lemDg}_\ell(-)$ commutes with substitution in the following way,

$$\mathsf{lemDg}_\ell(M[\bar{t}, \lambda\xi.\,M', \bar{r}]) \equiv \mathsf{lemDg}_\ell(M)[\bar{t}, S, R, R_K, \tilde{R}, \bar{r}]$$

where

$$\begin{aligned} S &\stackrel{\text{def}}{=} \lambda\kappa.\lambda\xi.\,M'[x\,[\kappa]/x] \\ R &\stackrel{\text{def}}{=} \lambda\xi.\,(\!|M'|\!)^{\lambda\_.\,\delta}_{\overline{v_{<\ell}}, x\mapsto y, (\hat{\gamma}, \xi)}\tau \\ R_K &\stackrel{\text{def}}{=} \lambda\xi.\lambda\kappa.\lambda j.\,M'\sigma \\ \tilde{R} &\stackrel{\text{def}}{=} \lambda\xi.\,\mathsf{lemDg}_\ell(M') \end{aligned}$$

and moreover we have $\mathsf{lemDg}_{\ell'}(M) = \mathsf{lemDg}_\ell(M)$ for $\ell' < \ell$, whenever M only contains constructor nodes with labels smaller than ℓ'. We will use these properties for the constructor case of $\mathsf{lemDg}_\ell(-)$.

Finally, we need to show that $\mathsf{lemDg}_\ell(-)$ preserves judgemental equality in the sense that $M \equiv_b N$ implies

$$\mathsf{lemDg}_\ell(M) \equiv \mathsf{lemDg}_\ell(N)$$

This is needed to show that $\mathsf{lemDg}_\ell(-)$ is well-defined in the case of systems as in the proof of Lemma 5.2, and to define $\mathsf{lemDg}_\ell(-)$ in the case of homogeneous compositions.

We induct on the structure of M. In case $M = x\,\overline{u}$, we need to build a path between $\lambda\kappa.\,\lambda j.\,(y_K\,(\overline{u}))\,[\kappa]\,j$ and

$$y(\lambda\kappa.\,\overline{u}[(\lambda\_.\,\gamma)[\overline{\kappa}]/\gamma, (\lambda\_.\,\hat{\gamma})[\overline{\kappa}]/\hat{\gamma}]).$$

The clock applications on the right hand side simplify, making the body of the lambda abstraction constant in κ. The left hand side η-contracts to $y_K\,\overline{u}$, so we can conclude by setting $\mathsf{lemDg}_\ell(x_j\,\overline{u})$ equal to $\tilde{y}\,\overline{u}$.

In the case $M = \mathsf{hcomp}^{j'}\,[\psi \mapsto M']\,M'_0$, we must build a path between $\lambda\kappa.\,\lambda j.\,\mathsf{hcomp}^{j'}\,[\psi \mapsto M'\sigma]\,M'_0\sigma$ and

$$\mathsf{comp}^{j'}_{D[v\tau\,j']}\,[\psi \mapsto (\!|M'|\!)^{\lambda\_.\,\delta}_{\overline{v_{<\ell}}, x\mapsto y, (\hat{\gamma}, j')}\tau]\,((\!|M'_0|\!)^{\delta}_{\overline{v_{<\ell}}, x\mapsto y, \hat{\gamma}}\tau).$$

Let p be the path connecting

$$\begin{aligned} \lambda\kappa.\lambda j.\mathsf{hcomp}^{j'}\,[\psi \mapsto\ & (\!|M'|\!)^{\lambda\_.\,\delta}_{\overline{v_{<\ell}}, x\mapsto y, (\hat{\gamma}, j')}\tau\,[\kappa]\,j] \\ & ((\!|M'_0|\!)^{\delta}_{\overline{v_{<\ell}}, x\mapsto y, \hat{\gamma}}\tau\,[\kappa]\,j) \end{aligned}$$

to the right hand side, obtained from the fact that both terms fill the same open box. We let q be the path connecting the left hand side to $p\,0$, obtained by combining $\mathsf{lemDg}_\ell(M')$ and $\mathsf{lemDg}_\ell(M'_0)$ with $\mathsf{hcomp}^{j'}$. Note that this is well-defined because $\mathsf{lemDg}_\ell(-)$ preserves judgemental equality. By transitivity we get $q \cdot p$ connecting the desired endpoints.

To prove that $\mathsf{lemDg}_\ell(-)$ preserves judgemental equality, we need that $\mathsf{lemDg}_\ell(M)$ when restricted by ψ is strictly equal to $\mathsf{lemDg}_\ell(M')$. Fortunately that is already true for q, while p is a constant path under those conditions, so by the right unit law we have path from $q \cdot p$ to $\mathsf{lemDg}_\ell(M')$. Using an hcomp with this latter path we define $\mathsf{lemDg}_\ell(M)$ so that it satifies the strict equality.

In case $M = \mathsf{con}_{\ell'}(\bar{t}, \lambda\xi.\,M', \bar{r})$, we need to build a path between $\lambda\kappa.\,\lambda j.\,\mathsf{con}_{\ell'}(\bar{t}, \lambda\xi.\,M'\sigma, \bar{r})$ and $v_{\ell'}(\lambda\_.\,\bar{t}, S, R, \bar{r})$ where $S = \lambda\kappa.\lambda\xi.\,M'[x\,[\kappa]/x]$ and $R = \lambda\xi.\,(\!|M'|\!)^{\lambda\_.\,\delta}_{\overline{v_{<\ell}}, x\mapsto y, (\hat{\gamma}, \xi)}\tau$. We will work right to left by expanding the right hand side definition. By definition we have $v_{\ell'} = f_{\ell'}(g_{\ell'}(b_{\ell'}))$, so $v_{\ell'}(\lambda\_.\,\bar{t})$ is equal to $\mathsf{cirr}_{\Gamma_{\ell'}}(\lambda\_.\,\bar{t})^*(g_{\ell'}(b_{\ell'})(\bar{t}))$, so that by cirr–$\mathsf{coh}_{\Gamma_\ell}(\bar{t})$ the right hand side is equal to $g_{\ell'}(b_{\ell'})(\bar{t}, S, R, \bar{r})$, let us call this path p_1. Note that p_1 will be a constant path when $\varphi_{\ell'}[\bar{r}] = 1_{\mathbb{F}}$ as it will collapse to an equality between the boundaries of its endpoints, specified by their type. By definition we have $g_{\ell'}(b_{\ell'})(\bar{t}, S, R, \bar{r})$ equal to $((i_{\ell'}.\mathsf{lemD}_{\ell'}[\bar{t}, S, R])^*(b_{\ell'}(\bar{t}, S, R)))(\bar{r})$, which is defined as an hcomp and so by filling it is equal to the base of the composition $b_{\ell'}(\bar{t}, S, R, \bar{r})$, let us call this path p_2. Note that when $\varphi_{\ell'}[\bar{r}] = 1_{\mathbb{F}}$ we will have $p_2 = \lambda i_{\ell'}.\,\mathsf{lemD}_{\ell'}[\bar{t}, S, R]$. Finally $b_{\ell'}(\bar{t}, S, R, \bar{r})$ is equal to

$$\lambda\kappa.\lambda j.\,\mathsf{con}_{\ell'}(\bar{t}, \lambda\xi.\,R\,(\lambda\_.\,\xi)\,[\kappa]\,j, \bar{r}),$$

which is path equal to the left hand side by $\mathsf{lemDg}_\ell(M')$ and $\mathsf{con}_{\ell'}$ itself, we call the resulting path p_3. Note that when $\varphi_{\ell'}[\bar{r}] = 1_{\mathbb{F}}$ we will have p_3 built from $e_{\ell'}$ and $\mathsf{lemDg}_\ell(M')$ instead. By transitivity, $p_3 \cdot p_2 \cdot p_1$ forms a path between the left and right hand sides.

To preserve judgemental equality, when $\varphi_{\ell'}[\bar{r}] = 1_{\mathbb{F}}$, the path $\mathsf{lemDg}_\ell(M)$ must be equal to $\mathsf{lemDg}_\ell(e_{\ell'}[\bar{t}, \lambda\xi.\,M', \bar{r}])$

which in turn is equal to $\mathsf{lemDg}_{\ell'}(e_{\ell'})[\overline{t}, S, R, R_K, \tilde{R}, \overline{r}]$ by the commuting with substitution property, and where $R_K = \lambda\xi.\lambda\kappa.\lambda j.\, M'\sigma$ and $\tilde{R}$ is given by $\mathsf{lemDg}_\ell(M')$. To build the necessary path we first contract the singleton pair $(R_K, \tilde{R})$, so that the only non-trivial path in the composition is p_2, which as we noted matches $\mathsf{lemD}_{\ell'}$, i.e.

$$\mathsf{lemDg}_{\ell'}(e_{\ell'})[\overline{t}, S, R, R, \xi.\,\mathsf{refl}, \overline{r}].$$

Using this, we define $\mathsf{lemDg}_\ell(M)$ as an hcomp of the path just defined and $p_3 \cdot p_2 \cdot p_1$. □

Proof of Theorem 5.4. The v_ℓ terms from lemma A.11 collectively form a proof of $\mathscr{C}(\delta, D)$, so by lemma A.9 we conclude $\Pi(t : \forall\kappa.\mathsf{H}\,\delta).D[t] \equiv \Pi(t : \forall\kappa.\mathsf{H}\,\delta).\forall\kappa.t\,[\kappa_0] = t\,[\kappa]$, as required. □

A.5 Composition Structure for Higher Inductive Types (Sect. 5.1)

Following [21], for any HIT $\delta : \Delta \vdash \mathsf{H}\,\delta$ type we define its composition operation, comp, in terms of the trans and hcomp operations, resulting in the following judgemental equality rule

$$\mathsf{comp}^i_{\mathsf{H}\,\delta}\,[\varphi \mapsto u]\,u_0 \equiv \mathsf{hcomp}^i_{\mathsf{H}\,\delta[1/i]}\,[\varphi \mapsto v(i)]\,(\mathsf{trans}^i_{\mathsf{H}\,\delta}\,\varphi\,u_0)$$

where $v(i)$ is $\mathsf{trans}^j_{\mathsf{H}\,\delta[i\vee j/i]}\,(\varphi \vee i = 1)\,(u\,i)$.

Furthermore we include judgemental equalities for trans when applied to elements of the HIT built by homogeneous composition or by constructors. In Section 3.4 of [21], the authors describe how trans computes when applied to constructors specified by a signature of the form

$$\mathsf{c} : (\overline{x} : \overline{A}(\delta))(\overline{i} : \mathbb{I}) \to \mathsf{H}\,\delta[\varphi \mapsto e]$$

where $\overline{A}(\delta)$ is a telescope including both non-recursive and recursive arguments. They are able to treat both kinds of arguments uniformly by working in a variation of cubical type theory where trans and hcomp are the primitive operations for all types while comp is derived, so that they can use $\mathsf{trans}^i_{\overline{A}(\delta)}$ to transport all the arguments at once. We have kept comp as the primitive in our type theory, but we can reuse their description as long as we show how to transport the arguments for the constructors in our schema, i.e., provide a replacement for $\mathsf{trans}^i_{\overline{A}(\delta)}$.

Given a constructor declaration $(\Gamma, \overline{\Xi}, \Psi, \varphi, e)$ and $\delta : \mathbb{I} \to \Delta$ we have to define $\mathsf{trans}^i_{\Gamma[\delta\,i], \Theta_{\overline{\Xi}[\delta\,i], \mathsf{H}\,(\delta\,i)}}\,\psi\,(\overline{t}, \overline{a})$. The $\Gamma[\delta\,i]$ part of the telescope can be dealt with using comp, as shown in [21] by the definition of ctrans: a transport operation derived from composition. We are left with having to define $\mathsf{trans}^i_{\Theta_{\overline{\Xi}[\delta\,i, \tilde{t}[i]], \mathsf{H}\,\delta\,i}}\,\psi\,(\overline{t}, \overline{a})$ where $\tilde{t}$ connects $\overline{t}$ to the result of transporting it. It is then sufficient to show how to transport elements of $C(i) := \Xi_k[\delta\,i, \tilde{t}[i]] \to \mathsf{H}\,\delta\,i$ for each Ξ_k in $\overline{\Xi}$. Here we follow the recipe for transport in function types [30], like so

$$\mathsf{trans}^i_{C(i)}\,\psi\,a_k = \lambda\xi.\mathsf{trans}^i_{\mathsf{H}\,\delta\,i}\,\psi\,(a_k\,(\mathsf{ctrans}^i_{\Xi_k[\delta\,(1-i), \tilde{t}[1-i]]}\,\psi\,\xi))$$

where we make use of the fact that $a_k\,(-)$ is a subtree of $\mathsf{con}(\overline{t}, \overline{a}, \overline{i})$ so it is well-founded to recursively transport. On top of the above, the transport operation commutes with homogeneous composition, as described in Sect. 3.2 of [21].

A.6 Detailed version of Section 6

The standard models of both Cubical Type Theory and Clocked Type Theory are based on presheaf categories. In this section we recall these models and show how to model the combined CCTT in a presheaf category over the product of the categories used in the interpretations of Cubical and Clocked Type Theory. One of the challenges in constructing the model is to equip the types of Clocked Type Theory (such as $\triangleright\,(\alpha : \kappa).A$) with composition structures as required to model types in Cubical Type Theory.

In this paper, following the convention of Mannaa et al. [35] (but breaking with the convention of Cohen et al. [20]) we will work with *covariant* presheaves. Recall that a covariant presheaf over a category $\mathbb{C}$ is a family of sets $X(c)$ indexed by objects of $\mathbb{C}$ together with a map mapping $f : c \to c'$ in $\mathbb{C}$ and $x \in X(c)$ to $f \cdot x \in X(c')$, respecting identities and composition:

$$\mathsf{id}_c \cdot x = x \qquad g \cdot (f \cdot x) = (gf) \cdot x \tag{14}$$

We write $\mathsf{PSh}(\mathbb{C})$ for the category of covariant presheaves on $\mathbb{C}$. We recall the notion of Category with Family (CwF) [25] a standard notion of model of dependent type theory.

Definition A.12. A category with family comprises:

1. A category $\mathbb{C}$. We use Γ, Δ to range over objects of $\mathbb{C}$
2. A functor $\mathsf{Fam} : \mathbb{C}^{\mathrm{op}} \to \mathsf{Set}$. Elements $A \in \mathsf{Fam}(\Gamma)$ are referred to as *families* over Γ. If $\sigma : \Delta \to \Gamma$ and $A \in \mathsf{Fam}(\Gamma)$ we write $A[\sigma]$ for $\mathsf{Fam}(\sigma)(A)$
3. A functor El associating to each Γ and each $A \in \mathsf{Fam}(\Gamma)$ a set $\mathsf{El}(A)$ of *elements* of A, and to each $\sigma : \Delta \to \Gamma$ a mapping $(-)[\sigma] : \mathsf{El}(A) \to \mathsf{El}(A[\sigma])$.
4. A *comprehension* operation mapping a family A over Γ to an object $\Gamma.A$ such that maps $\Delta \to \Gamma.A$ correspond bijectively to pair of maps $\sigma : \Delta \to \Gamma$ and elements $t \in A[\sigma]$, naturally in Δ.

We often refer to a CwF simply by the name of the underlying category $\mathbb{C}$. A CwF gives rise to a model of dependent type theory [28] in which contexts are modelled as objects of $\mathbb{C}$, types as families and terms as elements. Recall that any presheaf category is the underlying category of a CwF where families over an object Γ are indexed families of sets $X(c, \gamma)$ for $\gamma \in \Gamma(c)$ with maps $f \cdot (-) : X(c, \gamma) \to X(c', f \cdot \gamma)$ satisfying the equations (14), and elements are assignments mapping each $\gamma \in \Gamma(c)$ to $t(\gamma) \in X(c, \gamma)$ such that $t(f \cdot \gamma) = f \cdot t(\gamma)$.

We recall also the category $\int \Gamma$ of elements for a covariant presheaf Γ over a category $\mathbb{C}$. This has as objects pairs (c, γ) where $\gamma \in \Gamma(c)$ and morphisms from (c, γ) to (c', γ') morphisms $f : c \to c'$ such that $f \cdot \gamma = \gamma'$. Note that a family

over Γ is precisely the same as a presheaf over $\int \Gamma$. We will also use the equivalence

$$\mathsf{PSh}(\mathbb{C})/\Gamma \simeq \mathsf{PSh}(\textstyle\int \Gamma) \tag{15}$$

which states that a slice of a presheaf category is itself a presheaf category.

A.6.1 Modelling Cubical Type Theory. The standard model of Cubical Type Theory [20] uses presheaves over the *category of cubes*. Here, since we use covariant presheaves, we will write $\mathscr{C}$ for the opposite of the category used by Cohen et al. [20]. So $\mathscr{C}$ has as objects finite sets I, J, K and as morphisms from I to J maps $f : I \to \mathsf{dM}(J)$ where $\mathsf{dM}(J)$ is the free de Morgan algebra on the set J. Composition is the standard Kleisli composition, using the fact that $\mathsf{dM}(-)$ is a monad. From the model construction we recall in particular the interval object $\mathbb{I}(I) \stackrel{\text{def}}{=} \mathsf{dM}(I)$, representing the singleton set, and the face lattice $\mathbb{F}$, which at stage I is the free distributive lattice generated by elements $(i = 0)$ and $(i = 1)$ for each $i \in I$, and relations $(i = 0) \wedge (i = 1) = \bot_{\mathbb{F}}$.

This model construction can be extended work for any category of the form $\mathsf{PSh}(\mathscr{C} \times \mathbb{D})$. To do so we rely on the Orton and Pitts' axiomatisation of models of CTT [34, 43]. Rather than recalling these axioms, we recall the sufficient conditions summarised by Coquand et al. [23] for a presheaf topos to satisfy these axioms. These conditions are as follows.

- The interval object $\mathbb{I}$ is connected, and a bounded distributive algebra structure with distinct 0 and 1 elements. Exponentiation by $\mathbb{I}$ has a right adjoint.
- The canonical map from $\mathbb{F}$ to the subobject classifier is a monomorphism, and the universal cofibration $\top : 1 \to \mathbb{F}$ is a levelwise decidable inclusion. Monomorphisms that can be described as pullbacks of the latter are referred to as *cofibrations*. The interval endpoint inclusions $0, 1 : 1 \to \mathbb{I}$ must be cofibrations, and cofibrations must be closed under finite union (finite disjunction), composition (dependent conjunction), and universal quantification over $\mathbb{I}$.

A *cubical model* is a presheaf category satisfying the axioms above. In $\mathsf{PSh}(\mathscr{C} \times \mathbb{D})$, defining both objects as constant on the $\mathbb{D}$ component, using the corresponding objects in $\mathsf{PSh}(\mathscr{C})$

$$\mathbb{I}(I, d) \stackrel{\text{def}}{=} \mathbb{I}(I) \qquad\qquad \mathbb{F}(I, d) \stackrel{\text{def}}{=} \mathbb{F}(I)$$

make $\mathsf{PSh}(\mathscr{C} \times \mathbb{D})$ a cubical model, as also observed by Coquand et al. [23].

Given such a model, Orton and Pitts [43] express the structure sufficient to model CTT using the internal language of the presheaf topos as an extensional type theory. We recall here some definitions that will be relevant later. We assume a $\omega + 1$ long hierarchy of Grothendieck universes, leading to a corresponding hierarchy of universes a la Russell U_i in the internal language, each classifying presheaves of the appropriate size.

Definition A.13. A *CCHM fibration* (A, α) over a type $\Gamma : \mathsf{U}_\omega$ is a family $A : \Gamma \to \mathsf{U}_\omega$ with a fibration structure $\alpha : \mathsf{isFib}\,\Gamma\,A$ where

$$\mathsf{isFib}\,\Gamma\,A \stackrel{\text{def}}{=} (e : \{0,1\})(p : \mathbb{I} \to \Gamma) \to \mathsf{Comp}\,e\,(A \circ p)$$

$$\begin{aligned}\mathsf{Comp}\,e\,A \stackrel{\text{def}}{=} (\varphi : \mathbb{F})(u : [\varphi] \to (i : \mathbb{I}) \to A\,i)\\ \to \{u_0 : A\,e \mid \varphi \Rightarrow u\,e = u_0\}\\ \to \{u_1 : A\,\overline{e} \mid \varphi \Rightarrow u\,\overline{e} = u_1\}\end{aligned}$$

where $\overline{e}$ sends $0 : \mathbb{I}$ to 1 and vice versa.

Notice $\mathsf{Comp}\,0\,A$ closely matches the signature of the composition operation from CTT.

We can then build the following Category with Families of fibrant types, which we call Fib:

- A context Γ is a global element of U_ω.
- A family over Γ is a global CCHM fibration over Γ.
- An element t of a family (A, α) over Γ is a global element of $(\gamma : \Gamma) \to A\,\gamma$.

it then follows that Fib models the type formers of Cubical Type Theory.

Remark 2. *Cubical Type Theory as presented by Cohen et al. [20] includes specific judgemental equalities for the composition operator* comp^i_A *according to the shape of A, matching the behaviour of the given fibration structures in their model. The constructions by Orton and Pitts [43] do not necessarily produce fibration structures that satisfy the same equalities. One way to deal with this is to use alternative formulations of these rules [30, 53]. Since these equalities are mainly relevant for operational properties of CTT, we will ignore this issue in this paper.*

A.6.2 Modelling Clocked Cubical Type Theory. In the previous section we saw how to model Cubical Type Theory in any category of the form $\mathsf{PSh}(\mathscr{C} \times \mathbb{D})$. We now define the category $\mathscr{T}$ of *time objects* and extend the model to a model of CCTT in $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$. The objects of $\mathscr{T}$ are pairs $(\mathscr{E}; \delta)$, where $\mathscr{E}$ is a finite set (to be though of as a set of semantic clocks), and $\delta : \mathscr{E} \to \mathbb{N}$ is a map associating to each clock a finite amount of time steps that are left on that clock. A morphism $\sigma : (\mathscr{E}; \delta) \to (\mathscr{E}'; \delta')$ is a map $\sigma : \mathscr{E} \to \mathscr{E}'$ such that $\delta'\sigma \leq \delta$ in the pointwise order. This can be understood as a generalisation of the topos of trees model $\mathsf{PSh}(\omega^{\mathrm{op}})$ of guarded recursion [10], where the indexing category is restricted to objects where the first component $\mathscr{E}$ is a singleton. The category $\mathsf{PSh}(\mathscr{T})$ has previously been used to model type theories with guarded recursion and multiple clocks [12, 35], and (in a slight variation) Guarded Computational Type Theory [48].

The category $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$ has an object of clocks Clk defined as $\mathsf{Clk}(I, (\mathscr{E}; \delta)) = \mathscr{E}$. The can be used to model assumptions of the form κ : clock in contexts, and the types $\forall\kappa.A$ can be modelled as Π-types. To model $\triangleright$, recall that this

is a Fitch-style modal type operator and that these can be modelled using dependent right adjoint types [19].

Definition A.14. Let $\mathbb{C}, \mathbb{D}$ be CwFs and let $\mathsf{L} : \mathbb{C} \to \mathbb{D}$ be a functor between the underlying categories. A dependent right adjoint to L is a mapping associating to a family A over $\mathsf{L}\Gamma$ a family $\mathsf{R}A$ over Γ and a bijective correspondence between elements of A and elements of $\mathsf{R}A$, both natural in Γ.

The naturality requirement for R means that if $\gamma : \Gamma' \to \Gamma$ and A is a family over $\mathsf{L}\Gamma$, then $(\mathsf{R}A)[\sigma] = \mathsf{R}(A[\mathsf{L}\sigma])$. Writing $\overline{(-)}$ for both directions of the bijective correspondence on elements, the naturality condition for this means that if t is an element of A over $\mathsf{L}\Gamma$, then $\overline{t}[\gamma] = \overline{t[\mathsf{L}\gamma]}$, and similarly for the opposite direction.

Given an endofunctor L on a CwF $\mathbb{C}$ as well as a dependent right adjoint R to L, one can model a Fitch-style modal operator by modelling extensions of contexts with ticks by L and the modal operator by R. In the case of Clocked Type Theory, the ticks, as well as the modal operators are indexed over an object Clk in the model: Semantically, the hypothesis $\Gamma \vdash \kappa$: clock of the context extension rule for $\Gamma, \alpha : \kappa \vdash$ corresponds to an element of the slice category over Clk. By (15) the slice category over Clk is itself a presheaf category and therefore carries a natural CwF structure. In fact, if $\chi : \Gamma \to \mathrm{Clk}$ is an object in the slice category, then families over χ in the slice category correspond bijectively to families in $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$ over Γ. Exploiting this, Mannaa et al. [35] describe how to model ticks on clocks using a dependent right adjoint $\blacktriangleright$ to an endofunctor $\blacktriangleleft$ on the slice category over Clk. In this model $\Gamma, \alpha : \kappa \vdash$ is interpreted as the domain of $\blacktriangleleft(\llbracket\Gamma\rrbracket, \llbracket\kappa\rrbracket)$ and $\triangleright\,(\alpha : \kappa).A$ as the dependent right adjoint $\blacktriangleright$ applied to $\llbracket A\rrbracket$. The bijective correspondence on elements then models the bijective correspondence between terms $\Gamma, \alpha : \kappa \vdash t : A$ and terms $\Gamma \vdash u : \triangleright\,(\alpha : \kappa).A$ given by tick abstraction and application. Tick weakening, which syntactically corresponds to context projections $\Gamma, \alpha : \kappa \to \Gamma$ can be modelled using a natural transformation from $\blacktriangleleft$ to the identity.

A.6.3 Composition structure for dependent right adjoints.

Before recalling the dependent right adjoint structure on the slice category $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})/\mathrm{Clk}$ we now give general conditions ensuring that dependent right adjoint types carry composition structure. The conditions are all on the left adjoint functor.

Definition A.15. Let $\mathbb{C}$ be a cubical model with interval object $\mathbb{I}$ and let $\mathsf{L} : \mathbb{C} \to \mathbb{C}$ be a finite product preserving functor.

1. We say L preserves the interval if there is an isomorphism $\mathsf{L}(\mathbb{I}) \cong \mathbb{I}$ preserving endpoints, i.e., such that the following commutes for $e = 0, 1$.

$$\begin{array}{ccc} 1 & \xrightarrow{e} & \mathbb{I} \\ \downarrow\cong & & \downarrow\cong \\ \mathsf{L}1 & \xrightarrow{\mathsf{L}e} & \mathsf{L}\mathbb{I} \end{array}$$

2. We say L *preserves cofibrations* if $\mathsf{L}i : \mathsf{L}A \to \mathsf{L}B$ is a cofibration whenever $i : A \to B$ is, and if the diagram on the right below is a pullback whenever i is a cofibration and the diagram on the left is a pullback

$$\begin{array}{ccc} C & \xrightarrow{a} & A \\ \downarrow j & & \downarrow i \\ D & \xrightarrow{b} & B \end{array} \qquad \begin{array}{ccc} \mathsf{L}C & \xrightarrow{\mathsf{L}a} & \mathsf{L}A \\ \downarrow \mathsf{L}j & & \downarrow \mathsf{L}i \\ \mathsf{L}D & \xrightarrow{\mathsf{L}b} & \mathsf{L}B \end{array}$$

The condition of preserving cofibrations corresponds to giving an operation mapping cofibrations $\Gamma \vdash \varphi : \mathbb{F}$ on Γ to cofibrations $\mathsf{L}\Gamma \vdash \mathsf{L}_{\mathbb{F}}(\varphi) : \mathbb{F}$ such that $\mathsf{L}\Gamma.[\mathsf{L}_{\mathbb{F}}(\varphi)] \cong \mathsf{L}(\Gamma.[\varphi])$ as subobjects of $\mathsf{L}\Gamma$ and satisfying $\mathsf{L}_{\mathbb{F}}(\varphi[\sigma]) = \mathsf{L}_{\mathbb{F}}(\varphi)[\mathsf{L}\sigma]$. Here $[\varphi]$ is the family classified by φ.

Theorem A.16. *Let $\mathbb{C}$ be a cubical model, let $\mathsf{L} : \mathbb{C} \to \mathbb{C}$ be a functor preserving finite products, the interval and cofibrations, and let* R *be a dependent right adjoint to* L. *If a family A over $\mathsf{L}\Gamma$ carries a global composition structure, so does $\mathsf{R}A$ over Γ. Moreover, this assignment is natural in Γ.*

Note that this is a statement about *global* composition structures in the model. The theorem can not be proved in the internal logic of the topos, but can be proved in an extension of this using crisp type theory, similarly to the construction of universes for cubical type theory in crisp type theory [34]. The reason is that the proof uses the bijective correspondence of Definition A.14 which only applies to global terms.

To prove Theorem A.16, we need the following lemma giving an alternative description of the data of a composition structure. The lemma uses standard CwF notation writing $\mathsf{p} : \Delta.A \to \Delta$ for the projection out of a comprehension object, and (less standard) notation $\sigma.A : \Delta.A[\sigma] \to \Gamma.A$ if $\sigma : \Delta \to \Gamma$ for the functoriality of comprehension in the first component defined in the language of CwFs as $(\sigma\mathsf{p}, \mathsf{q})$.

Lemma A.17. *Let Γ be global element of U_ω and let $A : \Gamma \to \mathsf{U}_\omega$. To give a composition structure on A corresponds to giving, for each global element Δ of U_ω and map $\sigma : \Delta \times \mathbb{I} \to \Gamma$, an assignment expressed as the rule*

$$\frac{\Delta \vdash \varphi : \mathbb{F} \qquad \Delta.\mathbb{I}.[\varphi[\mathsf{p}]] \vdash u : A[\sigma\mathsf{p}] \qquad \Delta \vdash u_e : A[\sigma \circ (\mathrm{id}, e)] \qquad \Delta.[\varphi] \vdash u_e[\mathsf{p}] = u[(\mathrm{id}_\Delta, e).[\varphi]]}{\Delta \vdash c_\sigma\, \varphi\, u\, u_e : A[\sigma \circ (\mathrm{id}, 1 - e)]}$$

for each $e \in \{0, 1\}$, natural in Δ satisfying

$$\Delta.[\varphi] \vdash (c_\sigma\, \varphi\, u\, u_e)[\mathsf{p}] = u[(\mathrm{id}, 1 - e).[\varphi]]$$

Proof. Given e, to give the part of isFib corresponding to e corresponds to giving $p : \mathbb{I} \to \Gamma \vdash c : \mathrm{Comp}\, e\, (A \circ p)$

in the model. This in turn corresponds to an assignment of morphisms $\tau : \Delta \to (\mathbb{I} \to \Gamma)$ to terms $\delta : \Delta \vdash c_\tau : \mathrm{Comp}\, e\, (A \circ \tau(\delta))$ natural in Δ. By uncurrying, the latter corresponds to an assignment mapping $\sigma : \Delta \times \mathbb{I} \to \Gamma$ to $\delta : \Delta \vdash c_\sigma : \mathrm{Comp}\, e\, (A \circ \sigma(\delta, -))$, also natural in Δ. By further uncurrying the arguments to the composition operator and using similar naturality arguments in each case, we arrive at the description in the lemma. □

Proof of Theorem A.16. By Lemma A.17 it suffices to give an assignment mapping $\sigma : \Delta \times \mathbb{I} \to \Gamma$ to a rule

$$\frac{\Delta \vdash \varphi : \mathbb{F} \qquad \Delta.\mathbb{I}.[\varphi[\mathsf{p}]] \vdash u : (\mathsf{R}A)[\sigma\mathsf{p}] \qquad \Delta \vdash u_e : (\mathsf{R}A)[\sigma \circ (\mathrm{id}, e)] \qquad \Delta.[\varphi] \vdash u_e[\mathsf{p}] = u[(\mathrm{id}_\Delta, e)\,.[\varphi]]}{\Delta \vdash c_\sigma^{\mathsf{R}A}\, \varphi\, u\, u_e : (\mathsf{R}A)[\sigma \circ (\mathrm{id}, 1-e)]}$$

natural in Δ and satisfying the equality of Lemma A.17. Using $(\mathsf{R}A)[\sigma\mathsf{p}] = \mathsf{R}(A[\mathsf{L}(\sigma\mathsf{p})])$ the assumptions correspond to

$$\mathsf{L}(\Delta.\mathbb{I}.[\varphi[\mathsf{p}]]) \vdash \overline{u} : A[\mathsf{L}(\sigma\mathsf{p})] \quad \mathsf{L}\Delta \vdash \overline{u_e} : A[\mathsf{L}(\sigma \circ (\mathrm{id}, e))]$$

satisfying

$$\mathsf{L}(\Delta.[\varphi]) \vdash \overline{u_e}[\mathsf{Lp}] = \overline{u}[\mathsf{L}((\mathrm{id}_\Delta, e)\,.[\varphi])] \tag{16}$$

Since L preserves cofibrations, by the notation introduced after Definition A.15, $\mathsf{L}(\Delta.\mathbb{I}.[\varphi[\mathsf{p}]] \cong \mathsf{L}(\Delta.\mathbb{I}).[\mathsf{L}_\mathbb{F}(\varphi[\mathsf{p}])]$ as subobjects of $\mathsf{L}(\Delta.\mathbb{I})$. For simplicity we will leave this isomorphism implicit. Moreover, since L preserves finite products and the interval, there is an isomorphism $\xi_\Delta : \mathsf{L}\Delta.\mathbb{I} \cong \mathsf{L}(\Delta.\mathbb{I})$ natural in Δ such that

$$\begin{array}{ccc} & \mathsf{L}\Delta & \\ {\scriptstyle (\mathrm{id},e)}\swarrow & & \searrow{\scriptstyle \mathsf{L}(\mathrm{id},e)} \\ \mathsf{L}\Delta.\mathbb{I} & \xrightarrow{\xi_\Delta} & \mathsf{L}(\Delta.\mathbb{I}) \end{array} \tag{17}$$

Then, since $\mathsf{L}(\sigma\mathsf{p}) \circ \xi_\Delta.[\mathsf{L}_\mathbb{F}(\varphi[\mathsf{p}])] = \mathsf{L}(\sigma) \circ \xi_\Delta \circ \mathsf{p}$

$$\mathsf{L}(\Delta).\mathbb{I}.[\mathsf{L}_\mathbb{F}(\varphi)[\mathsf{p}]] \vdash \overline{u}[\xi_\Delta.[\mathsf{L}_\mathbb{F}(\varphi[\mathsf{p}])]] : A[\mathsf{L}(\sigma) \circ \xi_\Delta][\mathsf{p}]$$

and by (17)

$$\mathsf{L}\Delta \vdash \overline{u_e} : A[\mathsf{L}(\sigma) \circ \xi_\Delta][(\mathrm{id}, e)]$$

We can therefore apply the composition structure for A as in Lemma A.17 in the case of $L(\sigma) \circ \xi_\Delta : \mathsf{L}\Delta.\mathbb{I} \to \mathsf{L}\Gamma$, the cofibration $\mathsf{L}\Delta \vdash \mathsf{L}_\mathbb{F}(\varphi) : \mathbb{F}$, and the terms $\overline{u}[\xi_\Delta.[\mathsf{L}_\mathbb{F}(\varphi[\mathsf{p}])]]$ and $\overline{u_e}$, if only we can prove that

$$\mathsf{L}\Delta.[\mathsf{L}_\mathbb{F}(\varphi)] \vdash \overline{u_e}[\mathsf{p}] = \overline{u}[\xi_\Delta.[\mathsf{L}_\mathbb{F}(\varphi[\mathsf{p}])]][(\mathrm{id}_{\mathsf{L}\Delta}, e)\,.[\mathsf{L}_\mathbb{F}(\varphi)]]$$

Up to the isomorphism $\mathsf{L}\Delta.[\mathsf{L}_\mathbb{F}(\varphi)] \cong \mathsf{L}(\Delta.[\varphi])$ the context projection p equals Lp, and so in context $\mathsf{L}(\Delta.[\varphi])$ the left hand side reduces to $\overline{u_e}[\mathsf{Lp}]$. The right hand side reduces using (17) to

$$\overline{u}[\mathsf{L}(\mathrm{id}_\Delta, e)\,.\mathsf{L}_\mathbb{F}(\varphi)]$$

which up to the isomorphism $\mathsf{L}\Delta.[\mathsf{L}_\mathbb{F}(\varphi)] \cong \mathsf{L}(\Delta.[\varphi])$ equals $\overline{u}[\mathsf{L}((\mathrm{id}_\Delta, e)\,.[\varphi])]$, and so the required equality follows from (16). The composition structure for A therefore gives

$$\mathsf{L}\Delta \vdash c^A_{L(\sigma)\circ\xi_\Delta}\, \mathsf{L}_\mathbb{F}(\varphi)\, \overline{u}[\dots]\, \overline{u_e} : A[L(\sigma) \circ \xi_\Delta \circ (\mathrm{id}, 1-e)]$$

which, since $A[L(\sigma) \circ \xi_\Delta \circ (\mathrm{id}, 1-e)] = A[L(\sigma \circ (\mathrm{id}, 1-e))]$, corresponds to a term

$$\Delta \vdash c_\sigma^{\mathsf{R}A}\, \varphi\, u\, u_e : (\mathsf{R}A)[\sigma \circ (\mathrm{id}, 1-e)]$$

To show the equality

$$\Delta.\varphi \vdash (c_\sigma^{\mathsf{R}A}\, \varphi\, u\, u_e)[\mathsf{p}] = u[(\mathrm{id}, 1-e)\,.[\varphi]]$$

is equivalent to showing

$$\mathsf{L}(\Delta.\varphi) \vdash \overline{(c_\sigma^{\mathsf{R}A}\, \varphi\, u\, u_e)[\mathsf{p}]} = \overline{u[(\mathrm{id}, 1-e)\,.[\varphi]]}$$

which up to the isomorphism $\mathsf{L}(\Delta.\varphi) \cong \mathsf{L}\Delta.[\mathsf{L}_\mathbb{F}(\varphi)]$ corresponds to showing that the term

$$\mathsf{L}\Delta.\mathsf{L}_\mathbb{F}(\varphi) \vdash (c^A_{L(\sigma)\circ\xi_\Delta}\, \mathsf{L}_\mathbb{F}(\varphi)\ \overline{u}[\xi_\Delta.[\mathsf{L}_\mathbb{F}(\varphi[\mathsf{p}])]]\ \overline{u_e})[\mathsf{p}] \tag{18}$$

equals

$$\mathsf{L}\Delta.\mathsf{L}_\mathbb{F}(\varphi) \vdash \overline{u}[\mathsf{L}((\mathrm{id}, 1-e)).[\mathsf{L}_\mathbb{F}(\varphi)]] \tag{19}$$

By the equality rule for the composition structure on A, (18) equals

$$\overline{u}[\xi_\Delta.[\mathsf{L}_\mathbb{F}(\varphi[\mathsf{p}])]][(\mathrm{id}, 1-e)\,.[\mathsf{L}_\mathbb{F}(\varphi)]]$$

which equals (19) by (17). □

A.6.4 The dependent right adjoint. We now recall the structure of the dependent right adjoint in details. Mannaa et al. [35] define this structure for $\mathsf{PSh}(\mathscr{T})$ but the constructions carry over directly to $\mathsf{PSh}(\mathscr{C} \times \mathscr{T})$. The structure arises as an extension of an endo-adjunction on the slice category as in the following lemma slightly generalised from Clouston et al. [19], to which we refer for details.

Lemma A.18. *Let $\mathbb{C}$ and $\mathbb{D}$ be CwFs and let $\mathsf{L} : \mathbb{C} \to \mathbb{D}$ be a functor between the underlying categories with a right adjoint R. Suppose R extends to families and elements as in the following data*

1. *An operation mapping families A over Γ in $\mathbb{D}$ to families $\mathsf{R}_{\mathsf{Fam}}(A)$ over $\mathsf{R}\Gamma$ satisfying $\mathsf{R}_{\mathsf{Fam}}(A[\gamma]) = (\mathsf{R}_{\mathsf{Fam}}(A))[R\gamma]$*
2. *An operation mapping elements t of A to elements $\mathsf{R}_{\mathsf{El}}(t)$ of $\mathsf{R}_{\mathsf{Fam}}(A)$ satisfying $\mathsf{R}_{\mathsf{El}}(t[\gamma]) = (\mathsf{R}_{\mathsf{El}}(t))[R\gamma]$.*

Then L has a dependent right adjoint mapping families A over $\mathsf{L}\Gamma$ to $\mathsf{R}A = (\mathsf{R}_{\mathsf{Fam}}A)[\eta]$ where $\eta : \Gamma \to \mathsf{RL}\Gamma$ is the unit of the adjunction.

The endoadjunction on the slice category is best described by using the equivalent description of the slice category as $\mathsf{PSh}(\int \mathrm{Clk})$. The right adjoint is the simplest to describe and is similar to the functor $\blacktriangleright$ on the topos-of-trees [10]:

$$\blacktriangleright \Gamma(I, (\mathscr{C}; \delta), \lambda) = \begin{cases} \Gamma(I, (\mathscr{C}; \delta[\lambda \mapsto n]), \lambda) & \text{if } \delta(\lambda) = n{+}1 \\ 1 & \text{if } \delta(\lambda) = 0 \end{cases}$$

Here $\delta[\lambda \mapsto n](\lambda) = n$ and $\delta[\lambda \mapsto n](\lambda') = \delta(\lambda')$ for $\lambda' \neq \lambda$ and 1 in the second clause is a singleton set. This lifts

to families and elements in the sense of Lemma A.18, for example, if A is a family over Γ then

$$\blacktriangleright_{\mathsf{Fam}}(A)(I,(\mathscr{E};\delta),\lambda)(\gamma) = \begin{cases} A(\gamma) & \text{if } \delta(\lambda) = n{+}1 \\ 1 & \text{if } \delta(\lambda) = 0 \end{cases}$$

The left adjoint can be concretely described as $\blacktriangleleft\,\Gamma(I,(\mathscr{E};\delta),\lambda)$ having as elements pairs (σ, x) such that $(\mathsf{id}_I, \sigma) : (I,(\mathscr{E}';\delta'),\lambda') \to (I,(\mathscr{E};\delta),\lambda)$ with $\delta'(\lambda') > \delta(\lambda)$ and $x \in \Gamma(I,(\mathscr{E}';\delta'),\lambda')$ considered up to the equivalence relation $\sim$ generated by $(\sigma\tau, x) \sim (\sigma, (\mathsf{id}, \tau)\cdot x)$. We refer to Mannaa et al. [35] for the details of the adjunction structure as well as an abstract description of the left adjoint.

There is a natural transformation $\mathsf{p}_{\blacktriangleleft} : \blacktriangleleft \to \mathsf{id}$ defined as $\mathsf{p}_{\blacktriangleleft}(\sigma, x) = (\mathsf{id}, \sigma)\cdot x$. This is used in our model to interpret tick-weakening.

Theorem A.19. *Let Γ be an object of $\mathsf{PSh}(\mathscr{C}\times\mathscr{T})$ and $\kappa : \Gamma \to \mathrm{Clk}$. Suppose A is a family over the domain of $\blacktriangleleft(\Gamma,\kappa)$ which carries a composition structure, then $\blacktriangleright A$ carries a composition structure as a family over Γ.*

Lemma A.20. *The category $\int \mathrm{Clk}$ for Clk considered an object in $\mathsf{PSh}(\mathscr{T})$ has coproducts.*

Proof. The coproduct of $((\mathscr{E};\delta),\lambda)$ and $((\mathscr{E}';\delta'),\lambda')$ is the object $((\mathscr{E}'';\delta''),\lambda'')$ where λ'' is fresh, $\mathscr{E}''$ is the disjoint union of $\mathscr{E}\setminus\{\lambda\}$ and $\mathscr{E}'\setminus\{\lambda'\}$ and $\{\lambda''\}$ and δ'' agrees with δ on $\mathscr{E}\setminus\{\lambda\}$, with δ' on $\mathscr{E}'\setminus\{\lambda'\}$ and maps λ'' to the minimum of $\delta(\lambda)$ and $\delta'(\lambda')$. □

Proof of Theorem A.19. Recall that families and terms in the CwF of the slice category $\mathsf{PSh}(\int \mathrm{Clk})$ over an object corresponding to an element $\kappa : \Gamma \to \mathrm{Clk}$ correspond bijectively to families and elements of the CwF of $\mathsf{PSh}(\mathscr{C}\times\mathscr{T})$ over Γ. Since this correspondence also respects the interpretation of the internal dependent type theory, it suffices to show that $\blacktriangleright A$ carries a composition structure as expressed on the slice category, when A does. By Theorem A.16 this reduces to showing that $\blacktriangleleft$ preserves finite products, the interval and cofibrations.

Each component $\blacktriangleleft 1(I,(\mathscr{E};\delta),\lambda)$ of $\blacktriangleleft 1$ is easily seen to be inhabited. If $(\sigma,\star)$ and $(\tau,\star)$ are two elements of $\blacktriangleleft 1(I,(\mathscr{E};\delta),\lambda)$, then both of these are related under the equivalence relation used in the definition of $\blacktriangleleft$ to $([\sigma,\tau],\star)$, where $[\sigma,\tau]$ is the copairing of σ and τ out of the coproduct of their domains, which exists by Lemma A.20. So $\blacktriangleleft$ preserves the terminal object.

Writing $[(\sigma,(x,y))]$ for the equivalence class represented by $(\sigma,(x,y))$, the map

$$\blacktriangleleft(A\times B)(I,(\mathscr{E};\delta),\lambda) \to (\blacktriangleleft A \times \blacktriangleleft B)(I,(\mathscr{E};\delta),\lambda)$$

maps $[(\sigma,(x,y))]$ to $([(\sigma,x)],[(\sigma,y)])$, and the inverse maps $([(\sigma,x)],[(\tau,y)])$ to $[([\sigma,\tau],(\mathsf{inl}(x),\mathsf{inr}(y)))]$.

For the interval, the map $\mathsf{p}_{\blacktriangleleft} : \blacktriangleleft\mathbb{I} \to \mathbb{I}$, which (as decribed in the main text) is defined as $\mathsf{p}_{\blacktriangleleft}(\sigma,x) = (\mathsf{id},\sigma)\cdot x$ has an inverse which at $(I,(\mathscr{E};\delta),\lambda)$ maps $x \in \mathbb{I}(I)$ to (σ,x) where $\sigma : (\mathscr{E};\delta') \to (\mathscr{E};\delta)$ is tracked by the identity and δ' agrees with δ everywhere except at λ where it is one higher. This clearly preserves endpoints.

For cofibrations, we first show that $\blacktriangleleft$ preserves pullbacks of cofibrations. Suppose that the diagram on the left below is a pullback with i and j cofibrations.

$$\begin{array}{ccc} C & \xrightarrow{a} & A \\ \downarrow{\scriptstyle j} & & \downarrow{\scriptstyle i} \\ D & \xrightarrow{b} & B \end{array} \qquad \begin{array}{ccc} \blacktriangleleft C & \xrightarrow{\blacktriangleleft a} & \blacktriangleleft A \\ \downarrow{\scriptstyle \blacktriangleleft j} & & \downarrow{\scriptstyle \blacktriangleleft i} \\ \blacktriangleleft D & \xrightarrow{\blacktriangleleft b} & \blacktriangleleft B \end{array}$$

We must show that also the diagram in the right is a pullback. Since $\mathbb{F}$ is constant in the time dimension, it follows, since i is a fibration that any naturality square of the form

$$\begin{array}{ccc} A(I,(\mathscr{E};\delta),\lambda) & \xrightarrow{(\mathsf{id},\sigma)\cdot(-)} & A(I,(\mathscr{E}';\delta'),\lambda') \\ \downarrow{\scriptstyle i} & & \downarrow{\scriptstyle i} \\ B(I,(\mathscr{E};\delta),\lambda) & \xrightarrow{(\mathsf{id},\sigma)\cdot(-)} & B(I,(\mathscr{E}';\delta'),\lambda') \end{array}$$

is a pullback. From this it follows that the square on the right below is a pullback diagram.

$$\begin{array}{ccccc} \blacktriangleleft C & \xrightarrow{\blacktriangleleft a} & \blacktriangleleft A & \xrightarrow{\mathsf{p}_{\blacktriangleleft}} & A \\ \downarrow{\scriptstyle \blacktriangleleft j} & & \downarrow{\scriptstyle \blacktriangleleft i} & & \downarrow{\scriptstyle i} \\ \blacktriangleleft D & \xrightarrow{\blacktriangleleft b} & \blacktriangleleft B & \xrightarrow{\mathsf{p}_{\blacktriangleleft}} & B \end{array}$$

By the pullback pasting diagram it therefore suffices to show that the outer diagram is a pullback, which follows again by the pullback pasting diagram applied to

$$\begin{array}{ccccc} \blacktriangleleft C & \xrightarrow{\mathsf{p}_{\blacktriangleleft}} & C & \xrightarrow{a} & A \\ \downarrow{\scriptstyle \blacktriangleleft j} & & \downarrow{\scriptstyle j} & & \downarrow{\scriptstyle i} \\ \blacktriangleleft D & \xrightarrow{\mathsf{p}_{\blacktriangleleft}} & D & \xrightarrow{b} & B \end{array}$$

and naturality of $\mathsf{p}_{\blacktriangleleft}$.

Suppose now that $\chi_A : B \to \mathbb{F}$ classifies the cofibration $i : A \to B$. A similar argument to the one above for $\mathbb{I}$ shows that $\mathsf{p}_{\blacktriangleleft} : \blacktriangleleft\mathbb{F} \to \mathbb{F}$ is an isomorphism, which preserves truth. Then $\mathsf{p}_{\blacktriangleleft}\circ\blacktriangleleft(\chi_A)$ classifies $\blacktriangleleft i$, so also $\blacktriangleleft i$ is a cofibration. □

A.6.5 Interpreting ticks and tick application.

A simple tick judgement $\Gamma \vdash \kappa : u \rightsquigarrow \Gamma'$ is interpreted as a map in $\mathsf{PSh}(\int \mathrm{Clk})$ from Γ to $\blacktriangleleft\Gamma'$. Here, for simplicity, we keep the clock κ implicit. An element t of the family $\blacktriangleright A$ over Γ' corresponds bijectively to an element $\bar{t}$ of A over $\blacktriangleleft\Gamma'$, which can then be reindexed along the tick u to interpret tick application. A forcing tick judgement $\Gamma \vdash (\kappa, u) \rightsquigarrow \Gamma'$ is interpreted as a map from Γ to $\blacktriangleleft(\Gamma'.\mathrm{Clk})$ in $\mathsf{PSh}(\int \mathrm{Clk})$, where the domain is considered an element of the slice category over Clk with map given by κ, and $\Gamma'.\mathrm{Clk}$ an object with map given by the second projection. The assumption of the rule for forcing tick application corresponds semantically to an element of a family $\blacktriangleright A$ over an object $\Gamma'.\mathrm{Clk}$, which corresponds bijectively to an element of A over $\blacktriangleleft(\Gamma'.\mathrm{Clk})$. Tick

application is then interpreted by reindexing this element along the interpretation of the forcing tick.

To define the interpretation of ticks, say that family A over an object Γ is *timeless* if the map

$$(\blacktriangleleft(\mathrm{p}), \mathrm{q}[\mathrm{p}_\blacktriangleleft]) : \blacktriangleleft(\Gamma.A) \to (\blacktriangleleft\Gamma).A[\mathrm{p}_\blacktriangleleft]$$

is an isomorphism.

Lemma A.21. *The families $\mathbb{I}$ and* Clk*, as well as any cofibration, are timeless. As a consequence*

$$\blacktriangleleft([\![\Gamma, \mathsf{TL}(\Gamma')]\!]) \cong \blacktriangleleft([\![\Gamma]\!]).[\![\mathsf{TL}(\Gamma')]\!]$$

in $\mathrm{PSh}(\int \mathrm{Clk})$.

Proof. As noted in the proof of Theorem A.19, cofibrations A over Γ are constant in the time dimension in the sense that for any $\sigma : ((\mathscr{E};\delta),\lambda) \to ((\mathscr{E}';\delta'),\lambda')$ the map

$$(\mathrm{id},\sigma)\cdot(-) : A(\gamma) \to A((\mathrm{id},\sigma)\cdot\gamma)$$

is an isomorphism. This means that the inverse to the map of the lemma can be defined to map the element $((\sigma,\gamma),a) \in ((\blacktriangleleft\Gamma).A[\mathrm{p}_\blacktriangleleft])(I,(\mathscr{E};\delta),\lambda)$ to $[(\sigma,(\gamma,a'))] \in \blacktriangleleft(\Gamma.A)(I,(\mathscr{E};\delta),\lambda)$ where a' is the unique element such that $(\mathrm{id}_I,\sigma)\cdot a' = a$. A similar argument applies in the case of $\mathbb{I}$.

In the case of Clk, first note that

$$\blacktriangleleft(\Gamma.\mathrm{Clk}) = \blacktriangleleft(\Gamma \times \mathrm{Clk}) \cong \blacktriangleleft\Gamma \times \blacktriangleleft\mathrm{Clk}$$

since, as we saw in the proof of Theorem A.19, $\blacktriangleleft$ preserves finite products. It thus suffices to show that $\mathrm{Clk} \cong \blacktriangleleft\mathrm{Clk}$. In order to prove this, we first prove that any element in $\blacktriangleleft(\mathrm{Clk})(I,(\mathscr{E};\delta),\lambda)$ can be represented on the form (id,κ), for $\mathrm{id} : ((\mathscr{E};\delta+1),\lambda) \to ((\mathscr{E};\delta),\lambda)$ and $\kappa \in \mathscr{E}$. Here $\delta+1$ is the composition of δ with the successor function. This is true, because an element represented by (σ,κ) where $\sigma : ((\mathscr{E}';\delta'),\lambda') \to ((\mathscr{E};\delta),\lambda)$ and $\kappa \in \mathscr{E}'$ is equivalent to the element given by (σ,κ) where σ now is considered a map from $((\mathscr{E}';\delta'+1),\lambda')$ to $((\mathscr{E};\delta),\lambda)$. This element is in turn equivalent to $(\mathrm{id},\sigma(\kappa))$ as required. As a consequence, we can define an inverse to $\mathrm{p}_\blacktriangleleft$ to map $\kappa \in \mathrm{Clk}((\mathscr{E};\delta),\lambda)$ to $[(\mathrm{id},\kappa)]$.

The last statement of the lemma now follows by induction on Γ'. □

For $\Gamma \vdash \kappa : u \rightsquigarrow \Gamma'$, let n be the number of ticks on clock κ in $\Gamma \setminus \Gamma'$. Then the interpretation of u factors through $\blacktriangleleft^n\Gamma'$, and a tick assumption is interpreted as the appropriate projection $\blacktriangleleft^n \to \blacktriangleleft$. Likewise if $\Gamma \vdash (\kappa,u) \rightsquigarrow \Gamma'$ let n be the number of ticks on κ in $\Gamma \setminus \Gamma'$, then the interpretation of u factors through $\blacktriangleleft^n(\Gamma'.\mathrm{Clk})$ with tick variables interpreted as projections. The construction tirr is interpreted using the following lemma.

Lemma A.22. *For any $n \geq 1$ there exists a unique natural transformation $\alpha : \blacktriangleleft^n \to \blacktriangleleft$ such that $\mathrm{p}_\blacktriangleleft \circ \alpha = \mathrm{p}^n_\blacktriangleleft$, and a unique $\beta : \blacktriangleleft^n(-.\mathrm{Clk}) \to \blacktriangleleft(-.\mathrm{Clk})$ commuting with the projection to $-.\mathrm{Clk}$.*

In particular, this means that tirr is interpreted using constant paths in our model. Before proving Lemma A.22 we first give an alternative description of $\blacktriangleleft^n$ similar to the explicit description for $\blacktriangleleft$ itself.

Lemma A.23. *Let $n \geq 0$. Up to isomorphism, the functor $\blacktriangleleft^n$ maps Γ to the presheaf given at $(I,(\mathscr{E};\delta),\lambda)$ by equivalence classes of pairs (σ,x) such that $(\mathrm{id}_I,\sigma) : (I,(\mathscr{E}';\delta'),\lambda') \to (I,(\mathscr{E};\delta),\lambda)$ with $\delta'(\lambda') \geq \delta(\lambda)+n$ and $x \in \Gamma(I,(\mathscr{E}';\delta'),\lambda')$ considered up to the equivalence relation $\sim$ generated by $(\sigma\tau,x) \sim (\sigma,(\mathrm{id},\tau)\cdot x)$. The projection $\mathrm{p}_\blacktriangleleft : \blacktriangleleft^{n+1} \to \blacktriangleleft^n$ maps $[(\sigma,x)]$ to $[(\sigma,x)]$.*

We omit the routine proof of this, which is by induction on n.

Proof of Lemma A.22. Suppose $\alpha : \blacktriangleleft^n \to \blacktriangleleft$ is such that $\mathrm{p}_\blacktriangleleft \circ \alpha = \mathrm{p}^n_\blacktriangleleft$. We will show that $\alpha([(\sigma,x)]) = ([(\sigma,x)])$ for any $[(\sigma,x)] \in (\blacktriangleleft^n\Gamma)(I,(\mathscr{E};\delta),\lambda)$.

First consider the case of a representable object $\Gamma = y(I,(\mathscr{E}';\delta'),\lambda')$, and an element of the form

$$([(\sigma,\mathrm{id})]) \in \blacktriangleleft^n(y(I,(\mathscr{E}';\delta'),\lambda'))(I,(\mathscr{E};\delta),\lambda).$$

That is, $(\mathrm{id}_I,\sigma) : (I,(\mathscr{E}';\delta'),\lambda') \to (I,(\mathscr{E};\delta),\lambda)$ and $\delta'(\lambda') \geq \delta(\lambda)+n$ and id is the identity on $(I,(\mathscr{E}';\delta'),\lambda')$ which is an element of $y(I,(\mathscr{E}';\delta'),\lambda')$. Then $\alpha([(\sigma,\mathrm{id})])$ must be on the form $([(\tau,\rho)])$, and since $\mathrm{p}_\blacktriangleleft([(\tau,\rho)]) = (\mathrm{id}_I,\tau)\circ\rho$ and $\mathrm{p}^n_\blacktriangleleft([(\sigma,\mathrm{id})]) = (\mathrm{id}_I,\sigma)$, the assumption implies $(\mathrm{id}_I,\tau)\circ\rho = (\mathrm{id}_I,\sigma)$, and in particular that ρ is on the form (id_I,ρ'). Therefore

$$\begin{aligned}\alpha([(\sigma,\mathrm{id})]) &= [(\tau,(\mathrm{id}_I,\rho'))] = [(\tau,(\mathrm{id}_I,\rho')\cdot\mathrm{id})]\\ &= [(\tau\rho',\mathrm{id})] = [(\sigma,\mathrm{id})]\end{aligned}$$

Now, consider a general $[(\sigma,x)] \in (\blacktriangleleft^n\Gamma)(I,(\mathscr{E};\delta),\lambda)$. That is, $(\mathrm{id},\sigma) : (I,(\mathscr{E}';\delta'),\lambda') \to (I,(\mathscr{E};\delta),\lambda)$ and $\delta'(\lambda') \geq \delta(\lambda)+n$ and $x \in \Gamma(I,(\mathscr{E}';\delta'),\lambda')$. Let $\gamma : y(I,(\mathscr{E}';\delta'),\lambda') \to \Gamma$ be the element corresponding to x by the yoneda lemma, i.e., $\gamma(f,\sigma) = (f,\sigma)\cdot x$. Then

$$\begin{aligned}\alpha([(\sigma,x)]) &= \alpha([(\sigma,\gamma(\mathrm{id}))])\\ &= \alpha(\blacktriangleleft^n(\gamma)[(\sigma,\mathrm{id})])\\ &= \blacktriangleleft(\gamma)(\alpha([(\sigma,\mathrm{id})]))\\ &= \blacktriangleleft(\gamma)([(\sigma,\mathrm{id})])\\ &= [(\sigma,x)]\end{aligned}$$

For the second statement of the theorem, recall that [35, Proposition 7.1] states that $\mathrm{p}_\blacktriangleleft : \blacktriangleleft(\Gamma.\mathrm{Clk}) \to \Gamma.\mathrm{Clk}$ is an isomorphism. Therefore, also the projection $\mathrm{p}^n_\blacktriangleleft : \blacktriangleleft^n(\Gamma.\mathrm{Clk}) \to \Gamma.\mathrm{Clk}$ is an isomorphism, from which the statement follows. □

A.6.6 Semantics for HITs.

We provide semantics for any HIT definable in the schema with a direct generalization of the method employed in [21]. To each HIT signature given we associate a notion of algebra. An initial such algebra is then constructed and shown to support the necessary

structure to model the HIT. For this section we denote objects of the time category by $x, x', \ldots$ to avoid notational clashes.

Reflecting the syntax of higher inductive types, we will work with homogeneous composition and transport separately. The semantic counterpoint to hcomp is the notion of a fibrancy structure. A fibrancy structure for a type global type A is an operation taking a face $\varphi : \mathbb{F}$, an element $u_0 : A$ and a partial path $p : A^{\mathbb{I}}$ defined on the extent φ and equal to u_0 at the endpoint $0 : \mathbb{I}$. It then produces an element $h_A(\varphi, u_0, u) : A$ which is equal to $u(1)$ on extent φ.

Consider a list of constructors $\Delta \vdash \mathscr{K}$ constrs, consisting of constructors $(\ell_i, (\Gamma_i; \overline{\Xi_i}; \Psi_i; \varphi_i; e_i)) \in \mathscr{K}$ for each $1 \le i \le n$, and let $\Delta' \vdash \delta : [\![\Delta]\!]$. A $\mathscr{K}, \delta$-algebra is a presheaf $A \in \mathsf{PSh}(\mathscr{C} \times \mathscr{T})/\Delta'$ with a fibrancy structure h_A, and for each $(\ell_i, (\Gamma_i; \overline{\Xi_i}; \Psi_i; \varphi_i; e_i))$ semantic constructors c_i^A. A semantic constructor is a map $c_i : \Pi(\gamma : [\![\Gamma_i]\!](\delta)).\overline{([\![\Xi_i]\!](\delta, \gamma) \to A)} \to [\![\Psi]\!] \to A$. Given such c_i^A's, we can interpret the boundary conditions given by the constructors, and the semantic constructors are subject to the boundary coherence conditions generated by this interpretation.

As in the syntax, we define the semantic boundary interpretation $\|-\|$ by induction over the grammar for these terms. We denote the list of constructors in $\mathscr{K}$ before the i'th constructor by $\mathscr{K}_{<i}$, and the list of semantic constructors for these by $C_{<i}$. The boundary coherence condition states that $c_i^A(\gamma, \theta, \bar{i}) = \|M_{i,j}\|_{C_{<i}}(\gamma, \theta, \bar{i})$ when $\varphi_{i,j}(\bar{i})$, which is the direct interpretation of $[\varphi_{i,0}\ \|M_{i,0}\|, \ldots, \varphi_{i,n}\ \|M_{i,n}\|]$. We define the boundary interpretation for the remaining grammar as follows:

$$\|x_i(\overline{u}(\gamma, \theta, i))\|_{C_{<i}} = [\![\theta_i]\!]_{\mathsf{id}, [\![\overline{u}]\!]}$$
$$\|\mathsf{con}_j(\bar{t}, \overline{\lambda\xi.N}, \bar{i})\|_{C_{<i}} = c_j^A([\![\overline{t}]\!], \overline{\lambda\xi.\|N\|_{C_{<i}}}, [\![\bar{i}]\!])$$
$$\|\mathsf{hcomp}[\varphi \mapsto u]\, u_0\|_{C_{<i}} = h_A(\varphi, \|u_0\|_{C_{<i}}, \|u\|_{C_{<i}})$$

A morphism of $\mathscr{K}, \delta$-algebras $f : A \to B$ is a natural transformation preserving all $\mathscr{K}, \delta$-algebra structure up to equality. For constructors this means that $f(c_i^A(\gamma, \theta, r)) = c_i^B(\gamma, f \circ \theta, r)$.

We will now define a presheaf H^{pre}, and carve out the initial $\mathscr{K}, \delta$-algebra as a subpresheaf $H \subset H^{pre}$ following the construction exemplified in [21]. Let $\rho \in [\![\Delta]\!](I, x)$. The presheaf H^{pre} consists of formal constructor elements, and formal solutions to composition problems. Concretely the presheaf $H^{pre}(I, x, \rho)$ will contain elements of two forms: Firstly formal constructors $con_i(\gamma, \theta, r)$, where $\gamma \in [\![\Gamma_i]\!](I, x, \rho)$, $r \in [\![\Psi]\!](I, x, \rho)$, $[\![\varphi_i]\!](I, r) \neq 1_{\mathbb{F}}$, and θ is a family indexed by $f : (I, x) \to (J, x')$ and an element $\xi \in [\![\Xi_{i,j}]\!](J, x', f \cdot \rho, f \cdot \gamma)$ of elements of $H^{pre}(J, x', f \cdot \rho, f \cdot \gamma)$. Secondly hcomp elements of the form $\mathsf{hcomp}[\psi \mapsto u]\, u_0$ where $\psi \neq 1 \in \mathbb{F}(I), u_0 \in H^{pre}(I, x, \rho)$ and u is a family indexed by pairs of a map $f : I \to J$ in $\mathscr{C}$, such that $f \cdot \varphi = 1_{\mathbb{F}}$, and $r \in \mathbb{I}(J)$ of elements $u_{f,r} \in H^{pre}(J, x, (f, \mathsf{id}) \cdot \rho)$. To define the action of H^{pre} on morphisms, we first need to give interpretations of boundary terms. While this is not formally the same case as for general algebras, the interpretation follows the same structure and so we leave out the definition here. We denote this boundary interpretation in H^{pre} again by $\|-\|$, owing to the fact that when the definition is complete, the two will coincide. Let $g : (I, x) \to (J, x')$. We define the action on elements on H^{pre} as follows:

$$H^{pre}(g)(con_i(\bar{t}, \bar{a}, \bar{r})) :=$$
$$\begin{cases} \|M(g \cdot \bar{t}, g.\bar{a}, g \cdot \bar{r})\|_C & \text{if } \varphi_i(g \cdot \bar{r}) \\ con_i(g \cdot \bar{t}, g.\bar{a}, g \cdot \bar{r}) & \text{else} \end{cases}$$
$$H^{pre}(g)(\mathsf{hcomp}[\psi \mapsto u]u_0) :=$$
$$\begin{cases} u_{g,1} & \text{if } \psi(g \cdot \bar{r}) \\ \mathsf{hcomp}[g \cdot \psi \mapsto g.u](g \cdot u_0) & \text{else} \end{cases}$$

Here $g.\bar{a}$ is given by $(g.\bar{a})_{i,f,\xi} = a_{i,fg,\xi}$, $g.u$ is u similarly reindexed by the cubical component of g and C is list of constructors con_i on which the boundary interpretation is based.

Now we carve out those elements where the families taken as input are natural. This means that we include elements $con_i(\gamma, \theta, r)$ where each $\theta_{f,\xi}$ is in H and $\theta_{gf,g\cdot\xi} = g \cdot \theta_{f,\xi}$. Additionally, we include those $\mathsf{hcomp}[\psi \mapsto u]\, u_0$ elements where u_0 and each $u_{f,r}$ is in H, $u_{gf,(g,\mathsf{id})\cdot r} = (g, \mathsf{id}) \cdot u_{f,r}$ and $(f, \mathsf{id}) \cdot u_0 = u_{f,0}$ for suitable maps f such that $f \cdot \psi = 1_{\mathbb{F}}$.

Lemma A.24. *The presheaf H is the initial $\mathscr{K}, \delta$-algebra for each $\delta : \Delta$. As a consequence, it has a unique map to any cubical type A with the structure of a $\mathscr{K}, \delta$-algebra.*

Proof. Let A be a $\mathscr{K}, \delta$-algebra with fibrancy structure h_A and constructors semantic c_i. We are tasked with producing a natural transformation $e : H \to A$, and we define $e(u)$ by induction on the structure of u: In the case where $u = con_i(\bar{t}, \bar{a}, \bar{r})$, we define $e(u) := c_i(\bar{t}, \overline{e(a)}, \bar{r})$, and in the case $u = \mathsf{hcomp}[\varphi \mapsto u]u_0$ we define $e(u) := h_A(\varphi, e(u), e(u_0))$. In both cases, $e(x)$ is e applied levelwise to the family x. While making this definition, we have to verify naturality at each stage, which follows in each case from the conditions on the c_i's or the computational behaviour of the fibrancy structures. □

Lemma A.25. *The presheaf H carries a composition structure.*

The proof of this lemma uses the fact that Δ, Γ_i and $\Xi_{i,j}$ all carry transport structures. Given those we define a transport structure on H by initiality, following the syntactic description of the action of trans on constructors in Section A.5. Combining this with the homogeneous composition structure we conclude the proof by Lemma 5 of [21].

Proposition A.26. *The presheaf H supports the dependent elimination principle of H.*

This follows from Lemma A.24 as in [21].

In addition to Lemma A.25 and Proposition A.26, we need to verify that H supports the formation and introduction rules, as well as the judgemental equalities These verifications are routine.

A.6.7 Verifying the principle of induction under clocks. The principle of induction under clocks is verified in the model by the following theorem:

Theorem A.27. *Consider a HIT* $\Delta \vdash H$ type*, a type family* $\Gamma, \forall\overline{\kappa}.H\,(\delta\,[\overline{\kappa}]) \vdash D$ type *and a term* $\Gamma \vdash \delta : \forall\overline{\kappa}.\Delta$*. Suppose that we semantically have an elimination list, i.e., for each* i *we have a term* $u_i : [\![D[\lambda\overline{\kappa}.\mathsf{con}_i(\gamma\,[\overline{\kappa}], \overline{x\,[\overline{\kappa}]}, \psi)]]\!]$ *over the context*

$$\left[\!\!\left[\begin{array}{l}\Gamma, \gamma : \forall\overline{\kappa}.\Gamma_i[\delta\,[\overline{\kappa}]], \overline{x} : \overline{\forall\overline{\kappa}.\Xi_i[\delta\,[\overline{\kappa}], \gamma\,[\overline{\kappa}]] \to H\,(\delta\,[\overline{\kappa}])},\\ \overline{y} : \overline{(\forall\overline{\kappa}.\Xi_i[\delta\,[\overline{\kappa}], \gamma\,[\overline{\kappa}]]) \to D[\lambda\overline{\kappa}.\lambda\xi.x\,[\overline{\kappa}]\,\xi\,[\overline{\kappa}]]}, \psi : \Psi_i \vdash\end{array}\right]\!\!\right].$$

Then we can construct a term of $[\![D]\!]$ *over* $[\![\Gamma.\forall\overline{\kappa}.H\,(\delta\,[\overline{\kappa}])]\!]$*.*

Proof. The semantics of clock quantification of A is given by the semantic Π over the clock object, $\Pi\mathsf{Clk}.[\![A]\!]$. It is shown in [12] this is equivalent at I, x, ρ to the limit

$$\lim_n [\![A]\!](I, x + \lambda_n, \iota \cdot \rho, \lambda)$$

with structure maps given by the map $\mathsf{tick}^\lambda : (I, x + \lambda_{n+1}) \to (I, x + \lambda_n)$. Here $(\mathscr{C}, f) + \lambda_n = (\mathscr{C} \cup \{\lambda\}, f[\lambda \mapsto n])$. The map tick^λ is given by the identity maps, so it can be viewed as letting a single time step pass on the clock λ. Nominally such a family depends on the choice of fresh clock λ, but of course this dependence goes away since we have isomorphisms $x + \lambda_n \cong x + \lambda'_n$ in the time category. Hence we will suppress the change of λ, as long as these are fresh meaning that the environment variable is of the form $\iota \cdot \rho$.

The action of morphisms on HIT's in the model can only change the constructor type by acting on the cube component. As a consequence of this, the structure of elements in $[\![\forall\kappa.\mathsf{H}\,(\delta\,[\kappa])]\!]$ are determined at each level by their structure at 0.

It is sufficient to work with $d : [\![\forall\kappa.\Delta]\!](I, x)$, with the proof for general Γ following by pulling back along the term $\Gamma \vdash d : \forall\kappa.\Delta$. We start by making explicit the data of the assumed terms u_i. The input to a u_i is as follows:

(a) We have a family $t = (t_n) \in [\![\forall\kappa.\Gamma_i\,[\kappa]]\!](I, x, d)$.
(b) For each j we have a family

$$a = (a_n^j) \in [\![\forall\kappa.\Xi_{i,j}[\delta\,[\kappa], \gamma\,[\kappa]] \to \mathsf{H}\,(\delta\,[\kappa])]\!](I, x, d, t).$$

This can be further unfolded as follows: We have for each n, $f : (I, x + \lambda_n) \to (J, x')$ and

$$z \in [\![\Xi_{i,j}[\delta\,[\kappa], \gamma\,[\kappa]]]\!](J, x', f \cdot d_n, f \cdot t_n, f(\lambda))$$

an element $a^j_{n,f,z} \in [\![\mathsf{H}\,(\delta\,[\kappa])]\!](J, x', f \cdot d_n, f(\lambda))$.

(c) The next family is indexed by j as above, a morphism $g : (I, x) \to (J, x')$ and a family

$$z := (z_n) \in [\![\forall\kappa.\Xi_{i,j}[\delta\,[\kappa], \gamma\,[\kappa]]]\!](J, x', g \cdot d, g \cdot t).$$

Note here that this means we have a family $h := (h_n) \in [\![\forall\kappa.\mathsf{H}\,(\delta\,[\kappa])]\!](J, x', g \cdot d)$ given by $h_n := a^j_{n,g+\lambda_n,z_n}$. We then have an element $b^j_{n,z} \in [\![D]\!](J, x', g \cdot d, h)$, and we denote the family by b.

(d) An element $r \in \Psi_i(I)$.

When given this, we have an element

$$(\mathsf{con}_i(t_n, a_n, r))_n \in [\![\forall\kappa.\mathsf{H}\,(\delta\,[\kappa])]\!](I, x, d)$$

and we assume that we can inhabit D over this element, i.e., that we have

$$u_i(t, a, b, r) \in [\![D]\!](I, x, d, (\mathsf{con}_i(t_n, a_n, r))_n).$$

We need to inhabit $[\![D]\!]$ at an arbitrary element

$$h \in [\![\forall\kappa.\mathsf{H}\,(\delta\,[\kappa])]\!](I, x, d).$$

Observe that the structure of the family h is completely determined by the structure of h_0. This holds because the family is compatible with the tick maps, and these are trivial in the cube component which means in particular that they cannot change the outer constructor of an element of the HIT. This means that we only have to inhabit it at families either of the form $(\mathsf{con}_i(t_n, a_n, r))$ or of the form $(\mathsf{hcomp}[\varphi \mapsto u^n]\,u_0^n)$. We can do this by induction on the structure, since it is the same for each component of the family, and can therefore assume that we can inhabit $[\![D]\!]$ at structurally simpler families. Explicitly, the induction is done on the structure of the 0 component in the family, so that the inductive hypothesis says that whenever we have a family (h'_n) such that h'_0 is structurally simpler than h_0, we can inhabit $[\![D]\!]$ over this family. We denote the element over a family h by $\alpha(h)$.

We consider first the case where we are given a family

$$(\mathsf{con}_i(t_n, a_n, r))_n \in [\![\forall\kappa.\mathsf{H}\,(\delta\,[\kappa])]\!](I, x, d).$$

Note that (t_n) is forced to be compatible with tick maps and that r is constant. This means that we have $(t_n)_n \in [\![\forall\kappa.\Gamma_i[\delta\,[\kappa]]]\!](I, x, d)$ and $r \in \Psi(I)$, and we have the data of (a) and (d). The implied typing of a^j_n is a family over $f : (I, x+\lambda_n) \to (J, x')$ and $z \in [\![\Xi_{i,j}[\delta\,[\kappa], \gamma\,[\kappa]]]\!](J, x', f \cdot d_n, f \cdot t_n, f(\lambda))$ of elements $a^j_{n,f,z} \in [\![\mathsf{H}\,(\delta\,[\kappa])]\!](J, x', f \cdot d_n, f(\lambda))$. This is exactly the typing of (b), so we need only to construct the family in (c). We need this to be a family over $g : (I, x) \to (J, x')$ and $z := (z_n) \in [\![\forall\kappa.\Xi_{i,j}[\delta\,[\kappa], \gamma\,[\kappa]]]\!](J, x', g \cdot d, g \cdot t)$. Given this, we again define the family

$$h := (h_n) \in [\![\forall\kappa.\mathsf{H}\,(\delta\,[\kappa])]\!](J, x', g \cdot d)$$

by $h_n := a^j_{n,g+\lambda_n,z_n}$. Note that this family is compatible with ticks, and hence we obtain $\alpha((h_n))\ in [\![D]\!](J, x', g \cdot d, h)$ by inductive hypothesis, since h_0 is given by a component of the structurally simpler a. We define

$$\alpha((\mathsf{con}_i(t_n, a_n, r))_n) = u_i(t, a, \alpha((h_n)), r).$$

Consider now the hcomp case. Since $[\![D]\!]$ is a type, it carries a composition structure. The strategy for inhabiting

$[\![D]\!]$ will be to construct an appropriate composition problem and apply the inductive hypothesis. Consider a family $h = (\mathsf{hcomp}[\varphi \mapsto u^n]\, u_0^n) \in [\![\forall\kappa.\mathsf{H}\,(\delta\,[\kappa])]\!](I, x, d)$. Here the implicit typing is as follows: For each n, we have for each $f : I \to J$ such that $f \cdot \varphi = \top$ and $s \in \Psi(J)$ an element $u^n_{f,s} \in [\![\mathsf{H}\,(\delta\,[\kappa])]\!](J, x + \lambda_n, f \cdot d_n, \lambda)$ and an element $u_0^n \in [\![\mathsf{H}\,(\delta\,[\kappa])]\!](J, x + \lambda_n, d_n, \lambda)$. The u_0^n family has the right shape for us to employ our induction hypothesis to inhabit $[\![D]\!]$ over it. Because the shape of f and s does not depend on the time component, we also have for each f a family in n given by $(u^n_{f,s})$. All these families have 0 component structurally simpler, hence we can inhabit $[\![D]\!]$ over each of them by $\alpha((u_0^n))$ and $\alpha((u^n_{f,i}))$. We will write $\alpha((u^n))$ for the family given at f, i by $\alpha((u^n_{f,i}))$. We can then define $\alpha(h) = \mathsf{comp}_{D[v]}[\varphi \mapsto \alpha((u^n))]\, \alpha((u_0^n))$ where $v = (\mathsf{hfill}_{[\![\mathsf{H}\,\delta\,[\kappa]]\!]}[\varphi \mapsto u^n]\, u_0^n)$.

It must be verified that α is natural in morphisms of the category while defining it. Both because we need α to be natural for the desired conclusion and because the recursive calls used to define α require it. Let $f : (I, x) \to (J, x')$. Naturality means that $f \cdot \alpha(h) = \alpha(f \cdot h)$, which unfolds as follows:

- Consider a family $h = (\mathsf{con}_i(t_n, a_n, r))$. By definition $\alpha(h) = u_i(t, a, b, r)$ where $b_{g,z} = \alpha((a^j_{n,g+\lambda_n,z_n}))$. If f does not trigger a boundary condition for con_i we have

$$f \cdot \alpha(h) = u_i(f \cdot t, f.a, f.b, f \cdot r) \text{ and}$$

$$(f \cdot h)_n = \mathsf{con}_i((f + \lambda_n) \cdot t_n, (f + \lambda_n).a_n, (f + \lambda_n) \cdot r).$$

  Recall here that $g.a_n$ is defined to be the family a_n is given by composing the map index with g and acting on the input index. Applying α to this yields $u_i(((f + \lambda_n) \cdot t_n), ((f+\lambda_n).a_n), b', f \cdot r)$ where b' is defined from $((f + \lambda_n).a_n)$ as above, i.e.,

$$b'_{g,z} = \alpha((((f + \lambda_n).a)_{n,g+\lambda_n,z_n})).$$

  By definition of the morphism action on a limit we have $((f+\lambda_n)\cdot t_n) = f\cdot(t_n)$ and $((f+\lambda_n).a_n) = f.(a_n)$. The action of f and $f+\lambda_n$ are the same on r, since they have the same cubical component. It remains to argue that $f.b$ coincides with the b' defined from the family $f.a$, which already a part of the naturality of the u_i's. We check this componentwise for the family, so we have to show that $(f.b)_{g,z} = b'_{g,z}$. The left hand side is by definition equal to $b_{gf,f\cdot z} = \alpha((a^j_{n,gf+\lambda_n,(f+\lambda_n)\cdot z_n}))$, while the right hand side is equal to $\alpha((((f+\lambda_n).a)^j_{g+\lambda_n,z_n}))$. Unfolding the right hand side we get $\alpha(((a)^j_{(g+\lambda_n)(f+\lambda_n),(f+\lambda_n)\cdot z_n}))$, which is the same as the left hand side since $(g+\lambda_n)(f+\lambda_n) = gf + \lambda_n$.

  If f triggers a boundary condition of con_i we have by assumption that

$$f \cdot \alpha(h) = f \cdot u_i(t, a, b, r) = [\![\langle\!|e|\!\rangle^{\delta}_{\vec{\mathscr{E}},\vec{x}\mapsto\vec{y}}]\!](f \cdot t, f \cdot a, f \cdot b, f \cdot r)$$

  where e is the boundary term of con_i. On the other hand, the structure of H means that $\alpha(f\cdot h) = \alpha((\|e\|((f+\lambda_n)\cdot t_n, (f+\lambda_n).a_n, (f+\lambda_n)\cdot r))$ where $\|e\|$ is the boundary evaluation for $\mathsf{H}\,(\delta\,[\kappa])$. Equality of these two can be shown by induction on the structure of e.

- Consider a family $h = (\mathsf{hcomp}[\varphi \mapsto u^n]\, u_0^n)$. By definition $\alpha(h) = \mathsf{comp}_{D[v]}[\varphi \mapsto \alpha((u^n))]\, \alpha((u_0^n))$. If f does not make φ true we have to verify that

  $f \cdot \mathsf{comp}_{D[v]}[\varphi \mapsto \alpha((u^n))]\, \alpha((u_0^n)) = \alpha(f \cdot h)$.

  We unfold the right hand side:

$$\begin{aligned}
&\alpha(f \cdot h) = \\
&\quad \alpha((\mathsf{hcomp}[f \cdot \varphi \mapsto (f + \lambda_n).u^n]\, (f + \lambda_n) \cdot u_0^n)) \\
&= \mathsf{comp}_{D[f\cdot v]}[f \cdot \varphi \mapsto \alpha(((f + \lambda_n).u^n))] \\
&\qquad\qquad \alpha(((f + \lambda_n) \cdot u_0^n)) \\
&= \mathsf{comp}_{D[f\cdot v]}[f \cdot \varphi \mapsto f.\alpha((u^n))]\, f \cdot \alpha((u_0^n)) \\
&= f \cdot \mathsf{comp}_{D[v]}[\varphi \mapsto \alpha((u^n))]\, \alpha((u_0^n))
\end{aligned}$$

  The second to last step above follows by the inductive hypothesis that α is natural on simpler families, and the last line is exactly the left hand side of the original equation.

  If f makes φ true, both composition in D and hcomp reduce to appropriate components of the input, i.e., we have $\alpha(f \cdot h) = \alpha((u^n_{(f+\lambda),1}))$ and $f \cdot \alpha(h) = \alpha((u^n))_{f,1}$ for the right and left hand sides of the equality we are trying to show. These agree by definition of $\alpha((u^n))_{f,1}$.

This means that α assembles into a term of $[\![D]\!]$ over $[\![\Gamma.\forall\kappa.\mathsf{H}\,(\delta\,[\kappa])]\!]$ as desired. It is moreover clear from the definition of α that it will validate the computation rules of the induction principle, since the value on constructor families was given directly by the appropriate u_i and the value on hcomp families was given by the appropriate composition in D. □

3. UNIFYING CUBICAL AND MULTIMODAL TYPE THEORY

UNIFYING CUBICAL AND MULTIMODAL TYPE THEORY

FREDERIK LERBJERG AAGAARD, MAGNUS BAUNSGAARD KRISTENSEN, DANIEL GRATZER, AND LARS BIRKEDAL

[a] Aarhus University
e-mail address: aagaard@cs.au.dk

[b] IT University of Copenhagen
e-mail address: mbkr@itu.dk

[c] Aarhus University
e-mail address: gratzer@cs.au.dk

[d] Aarhus University
e-mail address: birkedal@cs.au.dk

Abstract. In this paper we combine the principled approach to programming with modalities of multimodal type theory (MTT) with the computationally well-behaved identity types of cubical type theory (CTT). The result—cubical modal type theory (Cubical MTT)—has the desirable features of both systems. In fact, the whole is more than the sum of its parts: Cubical MTT validates desirable extensionality principles for modalities that MTT only supported through ad hoc means.

We investigate the semantics of Cubical MTT and provide an axiomatic approach to producing models of Cubical MTT based on the internal language of topoi and use it to construct presheaf models. Finally, we demonstrate the practicality and utility of this axiomatic approach to models by constructing a model of (cubical) guarded recursion in a cubical version of the topos of trees. We then use this model to justify an axiomatization of Lob induction and thereby use Cubical MTT to smoothly reason about guarded recursion.

1. Introduction

Type theorists have produced a plethora of variants of type theory since the introduction of Martin-Löf type theory (MLTT), each of which attempts to refine MLTT to enhance its expressivity or convenience. Unfortunately, even extensions of type theory which appear orthogonal cannot be carelessly combined and so expert attention is frequently necessary to trust a type theory with the correct suite of extensions. We are particularly interested in two families of extensions to MLTT: *cubical type theories* [CCHM18, ABC$^+$21] and *(Fitch-style) modal type theories* [Clo18, BCM$^+$20, GKNB20].

Both of these lines of research aim to increase the expressivity of type theory, but along different axes. Cubical type theory gives a higher-dimensional interpretation to the identity

Key words and phrases: Dependent type theory, modal type theory, cubical type theory, guarded recursion, categorical semantics.

Preprint submitted to
Logical Methods in Computer Science

type and thereby obtains a more flexible notion of equality along with a computational interpretation of *univalent foundations* [Uni13]. Meanwhile, modal dependent type theory (MTT) extends MLTT with connectives which need not commute with substitution, allowing for type theory to model phenomena such as guarded recursion, axiomatic cohesion, or parametricity [BMSS12, Shu18, ND18].

While combining two complex type theories like this is not a task to be undertaken frivolously, experience has shown that a principled mixture of these two type theories would be useful. Indeed, modalities naturally appear in synthetic homotopy theory [Shu18] but without an apparatus like MTT, these extensions must be handled in an ad-hoc way, which can easily disrupt the desirable properties of type theory. Moreover, cubical type theory's version of equality validates invaluable principles like function extensionality and a cubical variant of MTT would thus eliminate the need to postulate such principles [BBC$^+$19, KMV21].

Prior to discussing how MTT$_\square$ fuses these systems, we set the stage by introducing both cubical type theory and multimodal type theory MTT separately.

Cubical type theory. Cubical type theory originates from the broader class of *homotopy type theories* whose study was instigated by Voevodsky's observation that the intensional identity type could be realized as a (homotopical) path space [KL21]. This shift in perspective justifies the inclusion of the univalence axiom which postulates an equivalence between equalities between elements of a universe and equivalences of the denoted types. While univalence has many pleasant consequences (function extensionality, effectivity of quotients, etc.), the addition of such an axiom disrupts crucial properties of the type theory. In particular, it is not possible to compute in such a theory. In order to rectify this issue, cubical type theory was introduced and shown to simultaneously support computation and validate the univalence axiom.

Cubical type theory extends MLTT with an interval object $\mathbb{I}$ along with a function space—a path space—to hypothesize over it. Intuitively, $\mathbb{I}$ abstracts the interval $[0, 1]$ and this connection is enhanced by the addition of operations, *e.g.* $0, 1 : \mathbb{I}$. Accordingly, $\mathbb{I} \to A$ classifies lines in A and by iterating we obtain squares, cubes, or arbitrary n-cubes in A.

While homotopy type theory recasts the identity type from MLTT as a path in a space, cubical type theory begins with paths and forces them to behave like an identity type. We therefore isolate a subtype $\mathsf{Path}_A(a_0, a_1)$ of paths $\mathbb{I} \to A$ whose values at 0 and 1 are a_0 and a_1. By further equipping A with *Kan operations*, $\mathsf{Path}_A(a_0, a_1)$ becomes a new model for the identity type. The Kan operations are subtle and complex but without them $\mathsf{Path}_A(-, -)$ is not even an equivalence relation. The flexibility afforded by these Kan operations, however, allows cubical type theory to support a computational effective interpretation of univalence.

MTT and Fitch-style modal type theories. We now turn from cubical type theory to modal type theory. Like cubical type theory, the motivations for modalities—a type constructor which does not necessarily respect substitution—are semantic; many models of type theory have further structure which does not directly commute with substitution, but would still be useful to internalize. For instance, the global sections comonad of a presheaf model is frequently essential for working internally to the model [CBGB15, LOPS18, Shu18], but it almost never commutes with substitution.

Unfortunately, much of the convenience of MLTT hinges on the fact that all operators do commute with substitution, so introducing a modality tends to disrupt nearly every

property of importance. In order to cope with this contradiction, modal type theories like MTT [GKNB20] have carefully isolated classes of modalities which can be safely incorporated into type theory while preserving properties such as canonicity and normalization [Gra21]. In fact, MTT can be instantiated with an arbitrary collection of (weak) dependent right adjoints [BCM+20]. The metatheory of MTT applies irrespective of the choice of mode theory, and therefore MTT can be seen as a *general* modal type theory, suitable for instantiation with a wide variety of different modalities to specialize the type theory to capture particular models.

In practice, MTT is parameterized by a mode theory, a 2-category used to describe the modalities in play. The objects of the mode theory correspond to individual copies of MLTT linked together by the modalities, the morphisms of the mode theory. The 2-cells of the mode theory induce natural transformations between modalities. For instance, in order to model the global sections comonad we pick a mode theory with one mode m, one modality μ, and a collection of 2-cells shaping this modality into a comonad, *e.g.*, 2-cells $\mu \longrightarrow \mu \circ \mu$ and $\mu \longrightarrow \mathsf{id}_m$ subject to several equations. Upon instantiating MTT with this mode theory we obtain a type theory with a comonad already known to satisfy many important metatheorems.

Towards Cubical MTT. In [GKNB20], each mode of MTT contains a copy of MLTT. Unfortunately, the type theory therefore inherits the well-known limitations of MLTT: the identity type is difficult to work with, function extensionality is not satisfied. One can resolve these issues by adding equality reflection to MTT, but this disrupts the decidability of type-checking. Moreover, several modalities arise in the context of homotopy type theory [Shu18] and adapting MTT to these models requires simply postulating univalence, thereby conferring the same set of difficulties.

We introduce $\mathsf{MTT}_\square$, a unification of Cubical type theory and MTT. To a first approximation, $\mathsf{MTT}_\square$ replays the theory of MTT, after replacing MLTT with cubical type theory. One thereby obtains a modal type theory with different modes—now copies of cubical type theory—connected by arbitrary dependent right adjoints. Moreover, each mode now satisfies univalence and function extensionality.

Beyond this, a computation rule for Kan operations in modal types is needed for computation (and thus for normalization), but it is not immediately well-typed. Indeed, a key challenge in combining MTT with CTT is exactly to capture sufficient interactions between modal and cubical aspects for this rule to be well-typed, whilst not making greater demands than the intended models can bear.

The switch from using MLTT to using CTT in MTT also improves modal types. For instance, in [Gra21] special care is taken to include *crisp induction* in order to validate the modal counterpart to function extensionality. While this addition preserves normalization and canonicity, modal extensionality is independent of MTT. In $\mathsf{MTT}_\square$, by contrast, the corresponding principle is simply provable (Theorem 3.1).

We show that models of $\mathsf{MTT}_\square$ can be assembled from certain models of cubical type theory connected by right adjoints. In particular, given coherent functors $f_\mu : \mathcal{C}_n \longrightarrow \mathcal{C}_m$ there is a model of $\mathsf{MTT}_\square$ which realizes context categories as $\mathbf{PSh}_{\mathbf{cSet}}(\mathcal{C}_m)$ and modalities as right Kan extension $(f_\mu)_*$. This ensures, for example, that despite the complexity of both MTT and cubical type theory, it is easily shown that $\mathsf{MTT}_\square$, appropriately instantiated, models cubical guarded recursion [BBC+19, KMV21]. Indeed, we show that the cubical underpinnings of $\mathsf{MTT}_\square$ improve the presentation of guarded recursion in MTT [Gra21].

The development of this theory of models also implies the soundness of $\mathsf{MTT}_\square$. We further conjecture, but do not prove, that the normalization results of [Gra21] for MTT and [SA21] for cubical type theory can be appropriately combined into a normalization proof for $\mathsf{MTT}_\square$.

Contributions. We contribute $\mathsf{MTT}_\square$, a synthesis of cubical type theory and MTT, and a foundation for multimodal and higher-dimensional type theories. In section 2 we recapitulate the basics of cubical type theory and MTT and in section 3 we present the definition of $\mathsf{MTT}_\square$ and further prove the aforementioned modal extensionality principle. Finally, in section 4 we introduce the model theory of $\mathsf{MTT}_\square$ and further show that cubical presheaves and certain essential geometric morphisms assemble into models. We then apply this theory to cubical guarded recursion in section 5 and explore the improved presentation of guarded recursion.

2. Cubical and multimodal type theory

We now recall the essential details of cubical type theory [CCHM18] and MTT [GKNB21]. We focus mostly on the portions relevant for $\mathsf{MTT}_\square$ and refer the reader to the existing literature for a more thorough introduction. Readers familiar with both systems may safely proceed to section 3.

2.1. Cubical type theory.

CTT begins by extending MLTT with a primitive interval $\mathbb{I}$ and algebraic structure to mimic the real interval $[0, 1]$. A term of type A which assumes *dimension variables* $i, j, k : \mathbb{I}$ corresponds to n-cubes (lines, squares, cubes) in A. Concretely, we add a new context formation rule $\Gamma, i : \mathbb{I}$ and a new syntactic class of *dimension terms* $\Gamma \vdash r : \mathbb{I}$:

$$\textit{(Abstract interval)} \quad r, s : \mathbb{I} \quad ::= \quad i \mid 0 \mid 1 \mid 1 - r \mid r \vee s \mid r \wedge s$$

We further identify dimension terms by the equations of De Morgan algebras.

Next, we add *path types*: a dependent product over the interval. The rules for this new connective are given in Figure 1 and—just as with dependent products—path types enjoy β and η rules stating *e.g.* $(\lambda i.\ p)(r) = p[r/i]$. In addition to β and η, paths are equipped with further equalities reducing them at endpoints, *e.g.*, given $p : \mathsf{Path}_A(a, b)$ then $p(0) = a : A$.

Out of the box, paths define a relation on types which is reflexive and symmetric and which validates extensionality principles such as function extensionality. They are not, however, transitive and it is this deficiency that leads to the *Kan composition operation* which forms the backbone of CTT. Intuitively, this composition operation lets us complete an open box (an n-cube missing certain faces) to an n-cube. In order to formalize this geometric intuition we add the face lattice $\mathbb{F}$, a class of propositions, which we use to codify the open boxes to be filled. We therefore add another syntactic class $\Gamma \vdash \phi : \mathbb{F}$:

$$\textit{(Face lattice)} \quad \phi, \psi : \mathbb{F} \quad ::= \quad \bot \mid \top \mid (r = 0) \mid (r = 1) \mid \phi \vee \psi \mid \phi \wedge \psi$$

Elements of $\mathbb{F}$ are identified by the equations of distributive lattices as well as the additional equation $(r = 0) \wedge (r = 1) = \bot$.

Elements $\phi : \mathbb{F}$ are used to *restrict* a context by assuming them, denoted Γ, ϕ. This allows us to locally force ϕ to hold so that, *e.g.*, $i : \mathbb{I}, (i = 0) \vdash i = 0 : \mathbb{I}$. We can take advantage of an assumption ϕ in our context through *systems*. The rules for systems are

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash \mathsf{Path}_A(a,b)} \qquad \frac{\Gamma, i : \mathbb{I} \vdash p : A}{\Gamma \vdash \lambda i.\ p : \mathsf{Path}_A(p(0), p(1))} \qquad \frac{\Gamma \vdash r : \mathbb{I} \qquad \Gamma \vdash p : \mathsf{Path}_A(a,b)}{\Gamma \vdash p(r) : A}$$

$$\frac{\Gamma, \phi_i \vdash a_i : A \qquad \Gamma, \phi_0 \wedge \phi_1 \vdash a_0 = a_1 : A}{\Gamma, \phi_0 \vee \phi_1 \vdash \{\ \phi_0\ a_0, \phi_1\ a_1\ \} : A} \qquad \frac{}{\Gamma, \bot \vdash \{\ \} : A}$$

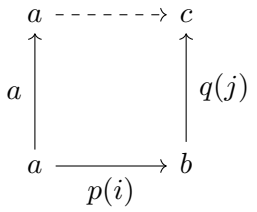
$$\frac{\Gamma \vdash \phi : \mathbb{F} \qquad \Gamma, i : \mathbb{I} \vdash A \qquad \Gamma, \phi, i : \mathbb{I} \vdash u : A \qquad \Gamma \vdash u_0 : A[0/i]\,[\phi \mapsto u[0/i]]}{\Gamma \vdash \mathsf{comp}^i_A\,[\phi \mapsto u]\,u_0 : A[1/i]\,[\phi \mapsto u[1/i]]}$$

Figure 1: Selected rules from CTT

given in Figure 1; intuitively they state that to construct an element in $\Gamma, \bigvee_i \phi_i \vdash u : A$, it suffices to construct elements $\Gamma, \phi_i \vdash u_i : A$ that agree on the overlap. An element constructed through this amalgamation restricts appropriately *e.g.*, $\Gamma, \phi_i \vdash u = u_i : A$.

We are frequently concerned with the behavior of a term after some assumption ϕ—its *boundary*—and therefore introduce notation for it. We write $\Gamma \vdash a : A\,[\phi \mapsto u]$ as shorthand for (1) a being a term of A and (2) under the assumption ϕ, $a = u : A$. With this machinery, we can now formulate the Kan composition rule, shown in Figure 1. This one principle is sufficient to prove the properties we expect of identity types, including J (path induction).[1]

We review the proof that path equality is transitive to give the reader a sense of the rule. Let A be a type that does not depend on any interval variables, and suppose $a, b, c : A$, $p : \mathsf{Path}_A(a,b)$, and $q : \mathsf{Path}_A(b,c)$. We form three lines in A: The paths p and q as well as the constant a line. Using these we form a system depending on i and j given by $\{\ (i = 0)\ a, (i = 1)\ q(j)\ \}$. The path p forms an extension of this system at $j = 0$, and so we can form the path $\lambda i.\ \mathsf{comp}^j_A\,[(i = 0) \mapsto a, (i = 1) \mapsto q(j)]\,p(i)$, which will reduce to the $j = 1$ part of our designed system, i.e., a at $i = 0$ and $q(1) = c$ at $i = 1$, thus proving transitivity. We think of the input data as an open box with bottom p and sides given by the system; in this analogy the composition forms a lid completing the outer square.



Remark 2.1. Given $x, y : A$, we write $x \equiv y$ when there exists an element of $\mathsf{Path}_A(x,y)$.

2.2. **Multimodal type theory.** To a first approximation, MTT is a collection of copies of MLTT for each $m \in \mathcal{M}$, connected by dependent adjunctions [BCM+20]. MTT is parameterized by a mode theory $\mathcal{M}$ [LS16], a strict 2-category. Each object $m, n, o : \mathcal{M}$ is assigned to a distinct *mode*: a copy of MLTT complete with its own judgments ($\Gamma\ \mathsf{cx} \mathbin{@} m$, $\Gamma \vdash M : A \mathbin{@} m$, ...). Many of the rules of MTT (dependent sums, inductive types, etc.) are *mode-local* and taken as-is from MLTT; the interesting features of MTT arise from allowing modes to interact with each other.

[1] Unlike in MLTT, however, path induction reduces on reflexivity only up to a path.

$$\frac{\Gamma, \{\mu\} \vdash A \mathbin{@} n}{\Gamma \vdash \langle \mu \mid A \rangle \mathbin{@} m} \qquad \frac{\Gamma, \{\mu\} \vdash a : A \mathbin{@} n}{\Gamma \vdash \mathsf{mod}_\mu(a) : \langle \mu \mid A \rangle \mathbin{@} m}$$

$$\frac{\Gamma, \{\mu\} \vdash A \mathbin{@} n}{\Gamma, x : (\mu \mid A)\ \mathsf{cx} \mathbin{@} m \qquad \Gamma, x : (\mu \mid A), \{\mu\} \vdash x : A \mathbin{@} n}$$

$$\frac{\begin{array}{c}\mu : n \to m \qquad \nu : o \to n \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\}, \{\nu\} \vdash A \mathbin{@} o \qquad \Gamma, \{\mu\} \vdash a : \langle \nu \mid A \rangle \mathbin{@} n \\ \Gamma, x : (\mu \mid \langle \nu \mid A \rangle) \vdash B \mathbin{@} m \qquad \Gamma, y : (\mu \circ \nu \mid A) \vdash b : B[\mathsf{mod}_\nu(y)/x] \mathbin{@} m\end{array}}{\Gamma \vdash \mathsf{let}_\mu\ \mathsf{mod}_\nu(y) \leftarrow a\ \mathsf{in}\ b : B[a/x] \mathbin{@} m}$$

Figure 2: Selected rules from MTT

Modes are connected to each other by *modalities*: a morphism $\mu : n \longrightarrow m$ induces a modality sending types A from mode n to types $\langle \mu \mid A \rangle$ in mode m. The actual presentation of modalities is necessarily complex because of dependence: given a type $\Gamma \vdash A \mathbin{@} n$, there is no obvious choice of context in mode m for $\langle \mu \mid A \rangle$. MTT resolves this tension in Fitch-style [Clo18] and pairs each modality with an adjoint action on contexts. In particular, given a modality $\mu : n \longrightarrow m$, we can obtain a new context $\Gamma, \{\mu\}\ \mathsf{cx} \mathbin{@} n$ from $\Gamma\ \mathsf{cx} \mathbin{@} m$. Further rules turn $-, \{\mu\}$ into a functor between categories of contexts and substitutions at modes n and m; intuitively a left adjoint to the modal type former $\langle \mu \mid - \rangle$. See the introduction and formation rules for $\langle \mu \mid - \rangle$ recorded in Figure 2.

The elimination rule for $\langle \mu \mid - \rangle$ is complex because we cannot 'reverse' the introduction rules without violating the admissibility of substitution. Instead, MTT annotates each variable in the context and replaces $\Gamma, x : A$ with $\Gamma, x : (\mu \mid A)$. One can access a variable annotated with μ if and only if it appears behind precisely $-, \{\mu\}$. The elimination rule uses these annotations to implement a *modal induction principle* and allows one to reduce the process of constructing an element of $B[a/x]$ for some $\Gamma.\{\nu\} \vdash a : \langle \mu \mid A \rangle \mathbin{@} m$ to the case $B[\mathsf{mod}_\mu(y)/x]$ for some fresh $y : (\nu \circ \mu \mid A)$; see the precise rule in Figure 2.

Thus far we have not mentioned the (2-)categorical structure of $\mathcal{M}$, but it is this additional structure which allows us to control the behavior of modalities. In fact, the operation sending a modality μ to $-, \{\mu\}$ is 2-functorial so that, *e.g.*, $\Gamma, \{\mu\}, \{\nu\} = \Gamma, \{\mu \circ \nu\}$. This fact is reflected into types; the assignment $\mu \mapsto \langle - \mid - \rangle$ is essentially pseudo-functorial. Consequently, a 2-cell $\alpha : \mu \longrightarrow \nu$ in $\mathcal{M}$ induces a substitution $\Gamma, \{\nu\} \vdash \{\alpha\}_\Gamma : \Gamma, \{\mu\} \mathbin{@} m$ which in turn produces a collection of functions $\langle \mu \mid - \rangle \to \langle \nu \mid - \rangle$. By modifying the equalities and 2-cells of $\mathcal{M}$, we can force $\langle \mu \mid - \rangle$ to become, *e.g.*, a comonad, a right adjoint, etc.

MTT also extends dependent products to hypothesize over types annotated with modalities other than id, i.e., to abstract over $x : (\mu \mid A)$ [GKNB21]. While these *modal dependent products* are convenient, we refrain from discussing them here for simplicity.

3. MTT$_\square$

Cubical multimodal type theory (MTT$_\square$) enhances MTT with a better behaved identity type and univalence by combining it with CTT. Like MTT, MTT$_\square$ is parameterised by a mode theory, i.e., a 2-category of modes, modalities, and 2-cells. Whereas MTT has a copy of MLTT at each mode, MTT$_\square$ has a copy of CTT. A guiding principle in the design of MTT$_\square$ is that cubical and modal aspects should be *orthogonal* to each other. In particular,

$$\textsc{int/exc}\quad \frac{\Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma, \{\mu\} \vdash r^\mu : \mathbb{I}_n \mathbin{@} n} \qquad \textsc{face/exc}\quad \frac{\Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma, \{\mu\} \vdash \phi^\mu : \mathbb{F}_n \mathbin{@} n}$$

$$\textsc{sb/exc-int}\quad \frac{}{\Gamma, i : \mathbb{I}_m, \{\mu\} \vdash \sigma_\mu : \Gamma, \{\mu\}, i : \mathbb{I}_n \mathbin{@} m} \qquad \textsc{sb/exc-face}\quad \frac{\Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma, \phi, \{\mu\} \vdash \tau_\mu : \Gamma, \{\mu\}, \phi^\mu \mathbin{@} m}$$

$$\textsc{sb/exc-int-inv}\quad \frac{}{\Gamma, \{\mu\}, i : \mathbb{I}_n \vdash \sigma^\mu : \Gamma, i : \mathbb{I}_m, \{\mu\} \mathbin{@} n} \qquad \textsc{sb/exc-face-inv}\quad \frac{\Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma, \{\mu\}, \phi^\mu \vdash \tau^\mu : \Gamma, \phi, \{\mu\} \mathbin{@} n}$$

$$\textsc{term-eq/comp-mod}\quad \frac{\begin{array}{c}\Gamma, i : \mathbb{I}_m, \{\mu\} \vdash A \mathbin{@} n \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma, \phi, i : \mathbb{I}_m, \{\mu\} \vdash u : A \mathbin{@} n \\ \Gamma, \{\mu\} \vdash u_0 : A[0/i] \mathbin{@} n \qquad \Gamma, \phi, \{\mu\} \vdash u[0/i] = u_0 : A[0/i] \mathbin{@} n\end{array}}{\Gamma \vdash \mathsf{comp}^i_{\langle \mu \mid A \rangle}\, [\phi \mapsto \mathsf{mod}_\mu(u)]\, \mathsf{mod}_\mu(u_0) = \mathsf{mod}_\mu(\mathsf{comp}^i_A\, [\phi^\mu \mapsto u[\sigma^\mu \circ \tau^\mu]]\, u_0) : \langle \mu \mid A \rangle[1/i] \mathbin{@} m}$$

Figure 3: Selected rules of $\mathsf{MTT}_\square$, presupposing $\mu : n \longrightarrow m$ and $\Gamma\ \mathsf{cx} \mathbin{@} m$.

the primitives of each system should interact as little as possible with primitives from the other. To realize this, we add certain exchange principles in subsection 3.1 which are then applied in subsection 3.2 to define composition for modal types.

We detail the novel aspects of $\mathsf{MTT}_\square$ and refer to Appendix A for an exhaustive account.

3.1. **Cubical exchange.** The orthogonality principle of $\mathsf{MTT}_\square$ dictates that the interval should be minimally impacted by the action of a modality on the context. Accordingly, we add *exchange operations*. Concretely, given a dimension term $\Gamma \vdash r : \mathbb{I}_m \mathbin{@} m$, we add a new dimension term $\Gamma, \{\mu\} \vdash r^\mu : \mathbb{I}_n \mathbin{@} n$, see INT/EXC in Figure 3. We demand that this operation is a morphism of De Morgan algebras and is lax natural, e.g. $(r \wedge s)^\mu = r^\mu \wedge s^\mu$ and $r^{\mu \circ \nu} = (r^\mu)^\nu$. Using this operation, it is possible to derive the *exchange substitution* $\Gamma, i : \mathbb{I}_m, \{\mu\} \vdash \sigma_\mu : \Gamma, \{\mu\}, i : \mathbb{I}_n \mathbin{@} m$, see SB/EXC-INT. Finally, we add an inverse to this, see SB/EXC-INT-INV, making $\Gamma, i : \mathbb{I}_m, \{\mu\}$ isomorphic to $\Gamma, \{\mu\}, i : \mathbb{I}_n$, once again in accordance with the orthogonality principle.

The case is entirely symmetrical for elements of the face lattice and the corresponding restricted contexts. Concretely, given a face $\Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m$, we add a new face $\Gamma, \{\mu\} \vdash \phi^\mu : \mathbb{F}_n \mathbin{@} n$, see FACE/EXC. Similarly to before, we require this operation to be a morphism of bounded lattices and be lax natural, but we further require that it matches with the corresponding operation on the interval via the equation $(r = 0)^\mu = (r^\mu = 0)$. Thus, $-^\mu$ commutes with everything but dimension variables, meaning that ϕ^μ is precisely $-^\mu$ applied to every dimension variable in ϕ. Just as before, we can derive a substitution, to which we add an inverse, making $\Gamma, \phi, \{\mu\}$ and $\Gamma, \{\mu\}, \phi^\mu$ isomorphic, see SB/EXC-FACE and SB/EXC-FACE-INV.

As we will see shortly, these rules are sufficient for composition in modal types, but one may still wonder if there would be merit in the addition of inverses to r^μ and ϕ^μ; after all, this would be in line with our orthogonality principle. It turns out, however, that such an addition would lead to a significant restriction of what models are valid, in particular,

it would invalidate our model of guarded recursion in section 5, and we thus refrain from making such an addition.

As mentioned, the exchange operations respect the 2-categorical structure of the mode theory, and since the exchange substitutions are derived from the simpler exchange operations, they inherit this property. We now record one such coherence explicitly for future use. $\mathsf{MTT}_\square$ inherits a *weakening substitution* from CTT: $\uparrow^\phi\colon \Gamma, \phi \longrightarrow \Gamma$. On may show that the two canonical substitutions $\Gamma, \{\mu\}, \phi^\mu \longrightarrow \Gamma, \{\mu\}$ are equal. Explicitly, we equate the direct restriction substitution $\uparrow^{\phi^\mu}$ to $\uparrow^\phi . \{\mu\} \circ \tau^\mu$.

3.2. **Composition in modal types.** Now we can tackle the problem of composition in $\mathsf{MTT}_\square$. Composition is, as the other cubical rules, added to the system locally and satisfies the same computation rules familiar from CTT for standard type formers. Modal types will support a computation principle similar to that of inductive types, allowing us to commute $\mathsf{mod}_\mu(-)$ with comp. Thus the status of composition in modal types is similar to that of natural numbers, where composition is a formal operation that reduces on canonical forms, as opposed to *e.g.* dependent sums.

The desired 'reduction' is TERM-EQ/COMP-MOD. We take a moment to show that the conclusion of this rule is well-typed and that the result has the expected boundary.

Inspecting the assumptions of this rule, we note that all but one are completely equivalent to a composition problem in $\langle \mu \mid A \rangle$ where the input terms are of form $\mathsf{mod}_\mu(u)$ and $\mathsf{mod}_\mu(u_0)$—with the exception that the assumption $u[0/i] = u_0$ is slightly stronger than necessary—so it is clear that the left-hand side of this equality is well-typed. That the right-hand side is well-typed is more subtle. We would like to show that $\mathsf{mod}_\mu(\mathsf{comp}_A\, [\phi^\mu \mapsto u[\sigma_\mu \circ \tau_\mu]]\, u_0)$ is well-typed. Inspecting the rule for composition in Figure 1, we see that we must first verify the following:

(1) $\Gamma, \{\mu\} \vdash \phi^\mu : \mathbb{F}_n \mathbin{@} n$
(2) $\Gamma, \{\mu\}, \phi^\mu, i : \mathbb{I}_m \vdash u[\sigma^\mu \circ \tau^\mu] : A[\sigma^\mu] \mathbin{@} n$
(3) $\Gamma, \{\mu\} \vdash u_0 : A[0/i] \mathbin{@} n$
(4) $\Gamma, \phi, \{\mu\} \vdash u[0/i] = u_0 : A[0/i] \mathbin{@} n$

All of these are immediate results of the premises of TERM-EQ/COMP-MOD. In particular, item 4 is precisely the aforementioned stronger premise.

Assured that both sides of TERM-EQ/COMP-MOD are well-typed, we show that that the right-hand side of this equality satisfies the same boundary condition as the left-hand side, i.e., that the right-hand side is equal to $\mathsf{mod}_\mu(u)[1/i]$ under ϕ.[2]

First, we observe that in context $\Gamma, \{\mu\}, \phi^\mu\ \mathsf{cx} \mathbin{@} n$ we have the following:

$$\mathsf{comp}_A\, [\phi^\mu \mapsto u[\sigma^\mu \circ \tau^\mu]]\, u_0 = u[\sigma^\mu \circ \tau^\mu][1/i] = u[1/i][\tau^\mu]$$

Next, we recall that weakening by an assumption of the face lattice commutes with face exchange. Given that the former is silent in our notation and the latter is not, this leads to the somewhat odd equation $\Gamma, \phi \vdash \mathsf{mod}_\mu(m) = \mathsf{mod}_\mu(m[\tau_\mu]) : \langle \mu \mid A \rangle \mathbin{@} m$ when $\Gamma, \{\mu\} \vdash m : A \mathbin{@} m$. Combining these two equations, we have the following in context Γ, ϕ:

$$\mathsf{mod}_\mu(\mathsf{comp}_A\, [\phi^\mu \mapsto u[\sigma^\mu \circ \tau^\mu]]\, u_0) = \mathsf{mod}_\mu((\mathsf{comp}_A\, [\phi^\mu \mapsto u[\sigma^\mu \circ \tau^\mu]]\, u_0)[\tau_\mu]) = \mathsf{mod}_\mu(u[1/i])$$

[2]This requirement is a further sanity check on the rule; without this equality the right-hand side would not solve the same composition problem as the left and the equation would be highly suspect.

3.3. **Extensionality principles in $\mathsf{MTT}_\Box$.** Function extensionality in $\mathsf{MTT}_\Box$ follows directly from function extensionality in CTT, since the rules used in CTT are all available mode-locally in $\mathsf{MTT}_\Box$. We will now prove *modal extensionality*, which cannot be proven in MTT and cannot be stated in CTT:

Theorem 3.1. *Given $\mu : n \to m$ and a pair of terms $\Gamma \vdash a, b : A \mathbin{@} n$, where A is the elements of some term of the universe, there is an equivalence $\mathsf{modext}_{a,b} : \langle \mu \mid \mathsf{Path}_A(a,b)\rangle \simeq \mathsf{Path}_{\langle \mu | A\rangle}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b))$.*

Proof. We define the map $\mathsf{modext}_{a,b}$ and show it to be an equivalence by constructing an inverse. Fix $\Gamma \vdash m : \langle \mu \mid \mathsf{Path}_A(a,b)\rangle \mathbin{@} m$ and $\Gamma \vdash r : \mathbb{I}_m \mathbin{@} m$. We wish to construct $\mathsf{modext}_{a,b}(m)(r)$. By modal induction it suffices to consider the case where $m = \mathsf{mod}_\mu(p)$ for some $\Gamma, \{\mu\} \vdash p : \mathsf{Path}_A(a,b) \mathbin{@} n$. Because p lives in a locked context whereas r does not, we need an exchange operation. We form $\Gamma, \{\mu\} \vdash r^\mu : \mathbb{I}_n \mathbin{@} n$, and define $\mathsf{modext}_{a,b}(m)(r) = \mathsf{mod}_\mu(p(r^\mu))$. Towards verifying that we obtain an inverse using path induction, note that for $\Gamma, \{\mu\} \vdash c : A \mathbin{@} n$ we have that

$$\mathsf{modext}_{c,c}(\mathsf{mod}_\mu(\mathsf{refl}(c))) = \mathsf{refl}(\mathsf{mod}_\mu(c)).$$

Next we define a map $\mathsf{modext}^{-1}_{a,b}$ in the inverse direction.[3] By based path induction along with careful modal induction, it suffices to define only $\mathsf{modext}^{-1}_{a,a}(\mathsf{refl}(\mathsf{mod}_\mu(a)))$. In this case we define $\mathsf{modext}^{-1}_{a,a}(\mathsf{refl}(\mathsf{mod}_\mu(a))) = \mathsf{mod}_\mu(\mathsf{refl}(a))$. Furthermore, by calculation:

$$\mathsf{modext}^{-1}_{c,c}(\mathsf{refl}(\mathsf{mod}_\mu(c))) \equiv \mathsf{mod}_\mu(\mathsf{refl}(c)).$$

Note that we obtain only a path rather than a judgmental equality because path induction computes only up to a higher path in cubical type theory.

Lastly, we prove that these maps form an equivalence. Let $\Gamma \vdash m : \langle \mu \mid \mathsf{Path}_A(a,b)\rangle \mathbin{@} m$. We are to find a path between $\mathsf{modext}^{-1}_{a,b}(\mathsf{modext}_{a,b}(m))$ and m. It suffices to do so when $m = \mathsf{mod}_\mu(\mathsf{refl}(c))$ for some $\Gamma, \{\mu\} \vdash c : A \mathbin{@} n$ where we compute:

$$\mathsf{modext}^{-1}_{c,c}(\mathsf{modext}_{c,c}(m)) = \mathsf{modext}^{-1}_{c,c}(\mathsf{refl}(\mathsf{mod}_\mu(c))) \equiv m.$$

For the reverse direction, let $\Gamma \vdash p : \mathsf{Path}_{\langle \mu | A\rangle}(\mathsf{mod}_\mu(a), \mathsf{mod}_\mu(b)) \mathbin{@} m$. We need a path between p and $\mathsf{modext}_{a,b}(\mathsf{modext}^{-1}_{a,b}(p))$. We again reduce to the case where $p = \mathsf{refl}(\mathsf{mod}_\mu(c))$ and compute from there: $\mathsf{modext}_{c,c}(\mathsf{modext}^{-1}_{c,c}(p)) \equiv \mathsf{modext}_{c,c}(\mathsf{mod}_\mu(\mathsf{refl}(c))) = p$. □

4. Semantics of $\mathsf{MTT}_\Box$

Section 3 toured through $\mathsf{MTT}_\Box$ informally, but in fact, $\mathsf{MTT}_\Box$ can be presented as a particular *generalized algebraic theory* [Car78]. This automatically gives rise to a category of models—a variant of the standard *categories with families* [Dyb96]—with several desirable properties such as the initiality of syntax. However, $\mathsf{MTT}_\Box$ is complex and the induced definition of model is nearly intractable to manipulate, let alone construct.

We fracture the definition of model into more manageable pieces, making heavy use of the natural models of MTT [Awo18, GKNB21]. In order to construct these models, we introduce *cubical MTT cosmoi*. This is a more compact structure supplementing MTT cosmoi [Gra21] with the ingredients necessary to internally construct a model of CTT [OP18, LOPS18]. In

[3] We will only need path induction and modal induction rather than path abstraction to define $\mathsf{modext}^{-1}_{a,b}$, meaning that it can also be defined in MTT.

practice, cosmoi are easier to obtain and suffice for the most important models *e.g.* those in cubical presheaves.

4.1. **Models of $\mathsf{MTT}_\square$.** We now present the definition of a model of $\mathsf{MTT}_\square$ with mode theory $\mathcal{M}$. To begin with, we require a strict 2-functor $[\![-]\!] : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$, known as the *modal context structure*. Intuitively, this 2-functor assigns each mode to a category of contexts. From this viewpoint, the functor $[\![\mu]\!] : [\![m]\!] \longrightarrow [\![n]\!]$ sends a morphism $\mu : n \longrightarrow m$ to the adjoint action $-.\{\mu\}$ contexts and the 2-cells $[\![\alpha]\!]$ interpret the natural transformations $\{\alpha\}$. We now specify the remaining structure on top of this functor.

Mode-local structure. Each mode $[\![m]\!]$ should contain a complete model of CTT, and we specify this in the language of natural models [Awo18] which provides a concise description of the connectives of type theory.

As a model of CTT, $[\![m]\!]$ has an interval object $\mathbf{I}_m : [\![m]\!]$. Just as in CTT, we require that $\mathbf{I}_m$ is a De Morgan algebra and that all products $- \times \mathbf{I}_m$ exist.

Next, we require a pair of presheaves $\widetilde{\mathcal{T}}_m, \mathcal{T}_m : \mathbf{PSh}([\![m]\!])$ representing respectively the collection of terms and types in a given context. Moreover, there is a projection map $\tau_m : \widetilde{\mathcal{T}}_m \longrightarrow \mathcal{T}_m$ which sends a term to its type. This universe is closed under dependent sums, products, etc. Each mode also contains an interpretation of the face lattice $\mathbb{F}_m : \mathbf{PSh}([\![m]\!])$ and face restriction which is used to specify the composition operations. While complex, this piece of the model is unchanged from CTT so we omit further details.

Modal types. Next we turn to the modal aspect of a model: modal context extension and modal types. Both of these structures are specified exactly as in MTT [GKNB21], with the small caveat that we require an additional equality for composition operation on modal types.

Cubical exchange. Finally, we must address the interaction of the functors $[\![\mu]\!]$ and the intervals and face lattices. Mirroring the syntax, we require natural transformations $\mathbb{I}_\mu : \mathbf{y}(\mathbf{I}_m) \longrightarrow [\![\mu]\!]^* \mathbf{y}(\mathbf{I}_n)$ that are pointwise morphisms of De Morgan algebras and that assemble with $\mathbf{y}(\mathbf{I}_m)$ into a lax natural transformation. From this, we can define a morphism, which we require to be have an inverse:

$$([\![\mu]\!]\pi_1, \mathbb{I}_{\mu,\Gamma\times\mathbf{I}_m}(\pi_2)) : [\![\mu]\!](\Gamma \times \mathbf{I}_m) \longrightarrow [\![\mu]\!](\Gamma) \times \mathbf{I}_n$$

The above is replayed for face lattices: We require natural transformations $\mathbb{F}_\mu : \mathbb{F}_m \longrightarrow [\![\mu]\!]^* \mathbb{F}_n$ that are pointwise morphisms of bounded lattices and that assemble with $\mathbb{F}_m$ into a lax natural transformation. From this can be defined a canonical morphism $[\![\mu]\!](\Gamma.[\phi]_m) \longrightarrow [\![\mu]\!].[\mathbb{F}_{\mu,\Gamma}(\phi)]_n$, which we require has an inverse.

4.2. **Cubical MTT cosmoi.** Even after the repackaging of models detailed in subsection 4.1, a model of $\mathsf{MTT}_\square$ is still a complex object. There are two orthogonal aspects to this complexity: (1) constructing the models of cubical type theory in each mode and (2) constructing the network of modalities and their actions on contexts. Fortunately, there already exists a technique to simplify (1); rather than construct a model of cubical type theory directly, [OP18] and [LOPS18] have shown that any topos satisfying a handful of axioms supports a model of cubical type theory. Moreover, (2) is partially addressed in [Gra21] by the notion of an MTT cosmos which abstracts several of the difficulties of

constructing a model of MTT. We now unify these two ideas to define *cubical MTT cosmoi* and prove that they induce a model of $\mathsf{MTT}_\square$.

MTT cosmoi. We will first recall the definition of MTT cosmoi and prove that they induce models of MTT.

Definition 4.1. A cosmos is a pseudofunctor $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ that takes objects to locally cartesian closed categories and morphisms to right adjoints. We denote the left adjoint to $F(\mu)$ by $F_!(\mu)$.

A cosmos abstracts from the basic situation we encountered in subsection 4.1: a 2-functor F picking out categories of contexts and the actions of modalities between them. In this case, we were primarily concerned not with the category $F(m)$, but with *presheaves over* $F(m)$. After all, it is the category of presheaves which hosts types and terms and where we formulate structures like context extension. Careful inspection reveals that we only require the locally Cartesian closed structure of $\mathbf{PSh}(F(m))$ when formulating the rest of the structure of a model, so it is natural to require only that each mode of a cosmos is locally Cartesian closed. Indeed, on top of this skeleton we can transport more of the structure of a model to cosmoi:

Definition 4.2. An extensional MTT cosmos is a cosmos F such that each mode is equipped with a morphism $\tau_m : \widetilde{\mathcal{T}}_m \longrightarrow \mathcal{T}_m$ representing inducing a universe closed under dependent products, sums, booleans, and extensional identity types. We further require that each map $F(\mu) : F(n) \longrightarrow F(m)$ induce a *dependent right adjoint* [BCM+20].[4]

We have leveraged same intuition as natural models to regard $\mathcal{T}_m$ (respectively $\widetilde{\mathcal{T}}_m$) as the collection of types (resp. terms), but without any representability requirements (they cannot be stated in LCCCs). Requiring closure of these universes under the connectives of MTT then ensures that an MTT cosmos induces a model of MTT in the sense of [GKNB21]. Prior to proving this, however, we require the following standard category-theoretic fact:

Lemma 4.3. *Let $\mathcal{C}$ be a 2-category and $F : \mathcal{C} \longrightarrow \mathbf{Cat}$ be a pseudofunctor such that each $F(f)$ is a right adjoint $F_!(f) \dashv F(f)$ then the left adjoints extend to a pseudofunctor $F_! : \mathcal{C}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$.*

Theorem 4.4. *An extensional MTT cosmos induces a model of extensional MTT with modal context structure is pseudonaturally equivalent to the pseudofunctor of left adjoints induced by Lemma 4.3. If the pseudofunctor of left adjoints is a strict 2-functor, it is equal to the modal context structure.*

Proof. Fix an extensional MTT cosmos $F : \mathcal{M} \longrightarrow \mathbf{Cat}$. By Lemma 4.3, the left adjoints $F_!(\mu)$ assemble into a pseudofunctor $F_! : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$. We may strictify this functor to get a strict 2-functor $\widehat{F_!} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ and a pseudonatural equivalence of categories $\alpha : F_! \longrightarrow \widehat{F_!}$. We claim that $\widehat{F_!}$ models extensional MTT.

For each mode m, we define the universe of types and terms as the Yoneda embedding of the universe already present in $\widehat{F_!}(m)$: $\widehat{\tau_m} = \mathbf{y}(\alpha_m(\tau_m))$. Because $\widehat{F_!}(m)$ is finitely complete, this is a representable natural transformation. Moreover, since both α_m and $\mathbf{y}$ preserve LCCC structure, this universe is closed under the types in the types cosmos: dependent

[4]These are essentially the same as modal types in MTT, further equipped with a syntactically ill-behaved but semantically convenient elimination rule.

right adjoints for each μ, dependent sums, products, booleans, and extensional identity types etc. Thus by [GKNB21, Theorem 7.1] $\widehat{F_!}$ models extensional MTT. □

Cubical cosmoi. An MTT cosmos F interprets each mode as an LCCC $F(m)$ because locally Cartesian closed structure is sufficient to specify the connectives of MTT. Unfortunately, it is not sufficient to apply the techniques of [OP18] and [LOPS18] and internally construct a model of CTT. We therefore isolate the notion of *a LOPS topos*,[5] containing precisely the required structure. We further define cubical cosmoi as particularly well-behaved networks of LOPS topoi.

Definition 4.5. A LOPS topos is an elementary topos $\mathcal{E}$ with a hierarchy of universes, an object of cofibrations $\mathbf{F}_\mathcal{E} \hookrightarrow \Omega$ and a tiny interval object $\mathbf{I}_\mathcal{E}$ subject to the Orton-Pitts axioms.[6]

Theorem 4.6 [LOPS18]**.** *There exists a model of CTT in every LOPS topos.*

Consider a cosmos $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ such that each $F(m)$ is a LOPS topos. Theorem 4.6 then implies that each mode is a model of cubical type theory, but on its own this is insufficient to conclude that F assembles into a model of $\mathsf{MTT}_\square$; we must ensure that each $F(\mu)$ properly preserves interval objects and face lattices. In order to isolate what further properties we must impose on F, we briefly revisit how one interprets constructs in cubical type theory such as systems and face restrictions in a LOPS topos.

Extending a context $X : \mathcal{E}$ by an interval variable is given product: $X \times \mathbf{I}_\mathcal{E}$. The structure of this context extension and of dimension terms more generally follow directly from the universal property of products along with the De Morgan algebra structure on $\mathbf{I}_\mathcal{E}$; a dimension term in context X is realized by a morphism $X \longrightarrow \mathbf{I}_\mathcal{E}$. Similarly, an element of the face lattice context X is interpreted by a morphism. $X \longrightarrow \mathbf{F}_\mathcal{E}$. Restricting a context by such a face is given by pullback:[7]

$$\begin{array}{ccc} \{X \mid \phi\} & \longrightarrow & \mathbf{1} \\ \downarrow & \lrcorner & \downarrow \top \\ X & \xrightarrow[\phi]{} & \mathbf{F}_\mathcal{E} \end{array}$$

Returning to our original question, we can now isolate some of the additional structure required by a cosmos valued in LOPS topoi to induce a model of $\mathsf{MTT}_\square$. In particular, a right adjoint between LOPS topoi will correctly model a dependent right adjoint which appropriately respects cubical structure when its left adjoint satisfies the following conditions:

Definition 4.7. A morphism of LOPS topoi is a geometric morphism $F_! \dashv F : \mathcal{E} \longrightarrow \mathcal{E}'$ along with an isomorphism of De Morgan algebras $\alpha_F : F_!(\mathbf{I}_{\mathcal{E}'}) \cong \mathbf{I}_\mathcal{E}$ and of bounded distributive lattices $\beta_F : F_!(\mathbf{F}_{\mathcal{E}'}) \cong \mathbf{F}_\mathcal{E}$. The maps α_F and β_F are required to be compatible in the sense that $\beta_F \circ F_!(- = 0) = (- = 0) \circ \alpha_F$.

[5]Named after the authors of [LOPS18]

[6]In fact, we make use of a slight strengthening of axioms presented by [OP18] in order to ensure that $\mathbf{I}_\mathcal{E}$ is an internal De Morgan algebra rather than a connection algebra.

[7]We have used the familiar set-comprehension notation for restriction by a face. Because $\mathbf{F}_\mathcal{E}$ is a subobject of Ω, this coincides with the standard interpretation of this notation in a topos.

Remark 4.8. Whilst $F(\mathbf{I}_{\mathcal{E}})$ is a De Morgan algebra, we make no assumption that it be isomorphic to $\mathbf{I}_{\mathcal{E}'}$. Doing so would correspond to adding an inverse to INT/EXC, but as mentioned in subsection 3.1, this is not valid in the model of guarded recursion in section 5; explicity, the right adjoint *later* does not preserve the interval. The case is the same for the object of cofibrations.

We now have built up the machinery necessary to define the desired fusion of LOPS topoi and MTT cosmoi:

Definition 4.9. A cubical MTT cosmos $F : \mathcal{M} \longrightarrow \mathbf{Cat}$ is an extensional MTT cosmos satisfying the following additional restrictions:

- $F(m)$ is a LOPS topos for each mode m,
- $F(\mu)$ is a morphism of LOPS topoi for each modality μ,
- The interval and face lattice isomorphisms are pseudonatural.

Theorem 4.10. *Any cubical MTT cosmos F induces a model of MTT$_\square$ with modal context structure pseudonaturally equivalent to the pseudofunctor of left adjoints induced by Lemma 4.3. If the pseudofunctor of left adjoints is a strict 2-functor, it is equal to the modal context structure.*

Proof. A cubical MTT cosmos is in particular an extensional MTT cosmos, meaning that all the rules from extensional MTT can be modelled with the strictified 2-functor $\widehat{F_!}$ by Theorem 4.4. Equivalence preserves being a LOPS topos, and thus $\widehat{F_!}(m) \simeq F_!(m) = F(m)$ is a LOPS topos, implying we can model all the mode-local rules added from CTT (including composition structures for all non-modal types) with Theorem 4.6. It thus remains to construct the exchange principles and composition structures on modal types.

We claim that $\widehat{F_!}$ (or rather, the pseudofunctor of right adjoints $\widehat{F}$ induced by the dual of Lemma 4.3) also has the cubical components of being a cubical MTT cosmoi. We have already argued that $\widehat{F}(m)$ is a LOPS topos since it is equivalent to $F(m)$, the fact the naturality squares of these equivalences commute up to natural isomorphism is enough to show that $\widehat{F}(\mu)$ is a morphism of LOPS topoi, and the pseudonatural coherence of the isomorphisms is preserved since the equivalences cohere pseudonaturally.

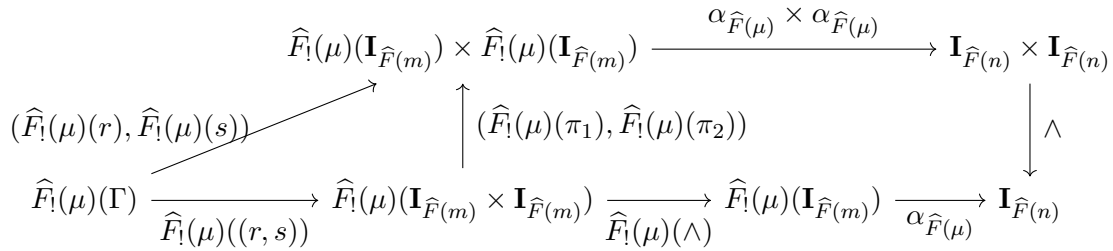
As a consequence of this, we have at each mode $m : \mathcal{M}$ a De Morgan algebra and a bounded distributive lattice $\mathbf{I}_{\widehat{F}(m)}, \mathbf{F}_{\widehat{F}(m)} : \widehat{F}(m)$ and for each modality $\mu : n \longrightarrow m$ coherent structure-preserving isomorphisms $\alpha_{\widehat{F}(\mu)} : \widehat{F_!}(\mu)(\mathbf{I}_{\widehat{F}(m)}) \cong \mathbf{I}_{\widehat{F}(n)}$ and $\beta_{\widehat{F}(\mu)} : \widehat{F_!}(\mu)(\mathbf{F}_{\widehat{F}(m)}) \cong \mathbf{F}_{\widehat{F}(n)}$.

To define the interval exchange operation, take a dimension term $r : \Gamma \longrightarrow \mathbf{I}_{\widehat{F}(m)}$, and define $\mathbb{I}_{\mu,\Gamma}(r)$ as the composite:

$$\widehat{F_!}(\mu)(\Gamma) \xrightarrow{\widehat{F_!}(\mu)(r)} \widehat{F_!}(\mu)(\mathbf{I}_{\widehat{F}(m)}) \xrightarrow{\alpha_{\widehat{F}(\mu)}} \mathbf{I}_{\widehat{F}(n)}$$

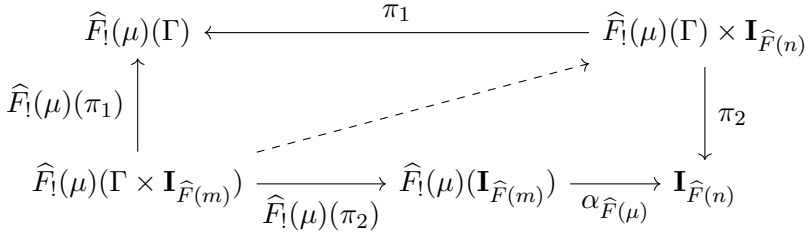
The naturality of $\mathbb{I}_\mu$ follows from the functoriality of $\widehat{F_!}(\mu)$, and they cohere lax naturally since the isomorphisms cohere. To see that it defines morphisms of De Morgan algebras, consider the concretely the case of $\wedge$. Preservation is then the commutativity of the following

diagram:



The right rectangle commutes since the $\alpha_{\widehat{F}(\mu)}$ preserves $\wedge$, and the left triangle commutes by the uniqueness of morphisms to products. Preservation of the other connectives follow similarly.

The final thing to verify for intervals is that the uniquely determined dashed arrow in the following diagram has an inverse:



This follows from $\alpha_{\widehat{F}(\mu)}$ being invertible and $\widehat{F_!}(\mu)$ preserving finite limits.

Replaying these arguments for the face lattices completes the construction of the exchange principles.

Lastly, we will construct the compositions structures on modal types. For this, we note that the model of extensional MTT obtained from Theorem 4.4 supports an inverse operation to $\mathsf{mod}_\mu(-)$ such that every element of a modal type is of the form $\mathsf{mod}_\mu(a)$. Therefore, the equation TERM-EQ/COMP-MOD can be taken as-is to fully define a composition structure. □

4.3. **Cubical presheaves.** The intended model of cubical type theory is a variant on the standard presheaf mode with types interpreted as a variant of Kan cubical sets [CCHM18]—particular presheaves on the cube category □ realized as the Lawvere theory of De Morgan algebras. One immediate benefit of the internal construction of a model of CTT is to generalize this result from cubical sets to *presheaves valued in cubical sets* [OP18]. Meanwhile, networks of presheaf categories connected by the essential geometric morphisms induced by functors between base categories are known to induce models of MTT [GKNB21, Section 8]. In fact, a consequence of Theorem 4.10 is that these two results can be essentially combined, thereby giving rise to the most important models of $\mathsf{MTT}_\square$.

Proposition 4.11. *Let $\mathcal{C}$ and $\mathcal{D}$ be small categories, let $F : \mathcal{C} \longrightarrow \mathcal{D}$ be a functor, and write F^*, $F_!$, and F_* for precomposition respectively left and right Kan extensions of $F \times \mathsf{id}_\square$.*

(1) *The presheaf categories $\mathbf{PSh}(\mathcal{C} \times \square)$ and $\mathbf{PSh}(\mathcal{D} \times \square)$ are LOPS topoi.*
(2) *The adjunction $F^* \dashv F_*$ induces a morphism of LOPS topoi.*
(3) *If $F_!$ is lex the adjunction $F_! \dashv F^*$ induces a morphism of LOPS topoi.*

Before proving the above proposition we recall some standard lemmas.

Lemma 4.12. *Let $\mathcal{C}$ and $\mathcal{D}$ be small categories and $F : \mathcal{C} \longrightarrow \mathcal{D}$ a functor. Left Kan extension of functors sending $X : \mathcal{C} \longrightarrow \mathbf{Set}$ to $F_!(X) : \mathcal{D} \longrightarrow \mathbf{Set}$ is lex iff $(F \downarrow d)$ is filtered for each $d : \mathcal{D}$. We note that this implies in particular that $(F \downarrow d)$ is connected.*

Proof. Fix a finite diagram of functors $X : \mathcal{I} \longrightarrow [\mathcal{C}, \mathbf{Set}]$. We can identify $F_!(\mathsf{lim}\, X)(d)$ with the colimit of the diagram $(F \downarrow d) \longrightarrow \mathcal{C} \longrightarrow \mathbf{Set}$ sending $f : F(c) \longrightarrow d$ to $(\mathsf{lim}\, X)(c)$. By assumption, this is a filtered colimit and thus it commutes with finite limits. Since limits are computed pointwise in $[\mathcal{C}, \mathbf{Set}]$, we may identify $(\mathsf{lim}\, X)(c) = \mathsf{lim}_i\, X_i(c)$ and compute:

$$\begin{aligned} F_!(\mathsf{lim}\, X)(d) &= \mathsf{colim}_{(c,f):(F\downarrow d)}\, \mathsf{lim}_{i:\mathcal{I}}\, X_i(c) \\ &= \mathsf{lim}_{i:\mathcal{I}}\, \mathsf{colim}_{(c,f):(F\downarrow d)}\, X_i(c) \\ &= \mathsf{lim}_{i:\mathcal{I}}\, F_!(X_i)(d) \end{aligned}$$

□

Lemma 4.13. *Let $F : \mathcal{C} \longrightarrow \mathcal{D}$ and $F' : \mathcal{C}' \longrightarrow \mathcal{D}'$. Then $(F \times F' \downarrow (d, d'))$ is equivalent to $(F \downarrow d) \times (F' \downarrow d')$ for each $d : \mathcal{D}$ and $d' : \mathcal{D}'$.*

Lemma 4.14. *Consider a diagram $F : A \times B \longrightarrow \mathcal{C}$ where B has a terminal object b_1. Then the colimit of $F(a, b)$ over $A \times B$ is isomorphic to the colimit of $F(a, b_1)$ over A, naturally in A.*

We can now prove proposition 4.11:

Proof. Note first that $\mathbf{PSh}(\mathcal{C} \times \square) = [(\mathcal{C} \times \square)^{\mathsf{op}}, \mathbf{Set}] \cong [\mathcal{C}^{\mathsf{op}}, \mathbf{cSet}] = \mathbf{PSh}_{\mathbf{cSet}}(\mathcal{C})$. Letting $\mathbb{I}, \mathbb{F} : \mathbf{cSet}$ be the interval respectively face lattice from [CCHM18, Section 8.1], we define $\mathbf{I}_{\mathcal{C}}(c, I) = \mathbb{I}(I)$ and $\mathbf{F}_{\mathcal{C}}(c, I) = \mathbb{I}(I)$ for $c : \mathcal{C}$ and $I : \square$.

For (1) these topoi satisfy the Orton-Pitts axioms as noted in [CRS21]. To see that the intervals defined above are tiny we proceed as follows: Using the Yoneda lemma along with the fact that $\mathbb{I}$ is naturally isomorphic to $[-, \{i\}]_{\square}$ shows that $\mathbf{y}(c, I) \times \mathbf{I} \cong \mathbf{y}(c, I + \{i\})$, and we thus calculate:

$$\begin{aligned} X^{\mathbf{I}}(c, I) &\cong [\mathbf{y}(c, I), X^{\mathbf{I}}] \\ &\cong [\mathbf{y}(c, I) \times \mathbf{I}, X] \\ &\cong [\mathbf{y}(c, I + \{i\}), X] \\ &\cong X(c, I + \{i\}) \\ &\cong (\mathsf{id}_{\mathcal{C}} \times (- + \{i\}))^*(X)(c, I) \end{aligned}$$

The above is natural in X, and thus exponentiation by $\mathbf{I}$ is (naturally isomorphic to) the precomposition functor $(\mathsf{id}_{\mathcal{C}} \times (- + \{i\}))^*$. As this functor has a right adjoint, we have shown that $\mathbf{I}$ is tiny.

We write $\pi_{\mathcal{C}}, \pi_{\mathcal{D}}$ the projections $\mathcal{C} \times \square \longrightarrow \square$ and $\mathcal{D} \times \square \longrightarrow \square$ respectively. For the second (respectively third) requirement we show the following:

- F^* (resp. $F_!$) preserves finite limits.
- $\iota : F^* \circ \pi_{\mathcal{D}}^* \cong \pi_{\mathcal{C}}^*$ ($F_! \circ \pi_{\mathcal{C}}^* \cong \pi_{\mathcal{D}}^*$)
- This isomorphism is an isomorphism of De Morgan algebras at $\mathbb{I}$ and of distributive lattices at $\mathbb{F}$.

The remaining conditions of a morphism of LOPS topoi follow automatically from the naturality α (resp. β).

For the first item, we note that F^* preserves all limits since it is a right adjoint, and that $F_!$ preserves finite limits by assumption.

Next, the desired isomorphism $F^*(\pi^*_{\mathcal{D}}(X)) \cong \pi^*_{\mathcal{C}}(X)$ can be taken to be the identity, justified by the following computation:

$$F^*(\pi^*_{\mathcal{D}}(X))(c, I) = \pi^*_{\mathcal{D}}(X)(F(c), I) = X(I) = \pi^*_{\mathcal{C}}(X)(c, I)$$

It is clear that this isomorphism preserves the De Morgan algebra structure when $X = \mathbb{I}$ and the distributive lattice structure when $X = \mathbb{F}$.

It remains to consider these conditions for $F_!$. We construct $F_!(\pi^*_{\mathcal{C}}(X)) \cong \pi^*_{\mathcal{D}}(X)$ as the composite of a string of natural isomorphisms:

$$\begin{aligned}
F_!(\pi^*_{\mathcal{C}}(X))(d, I) &\cong \mathsf{colim}_{((c,I'),_):(F\times \mathsf{id}\downarrow(d,I))}\, \pi^*_{\mathcal{C}}(X)(c, I') \\
&= \mathsf{colim}_{((c,I'),_):(F\times \mathsf{id}\downarrow(d,I))}\, X(I') \\
&\cong \mathsf{colim}_{((c,_),(I',_)):(F\downarrow d)\times(\mathsf{id}\downarrow I)}\, X(I') && \text{lemma 4.13} \\
&\cong \mathsf{colim}_{(c,_):(F\downarrow d)}\, X(I) && \text{lemma 4.14} \\
&\cong X(I) \\
&= \pi^*_{\mathcal{D}}(X)(d, I)
\end{aligned}$$

In the above calculation, the fourth isomorphism is obtained by observing that $\mathsf{colim}_{(c,\_):(F\downarrow d)}\, X(I)$ is a constant diagram over $(F \downarrow d)$; because $F_!$ is lex lemma 4.12 ensures that $(\mathsf{id}_\square \times F \downarrow (d, I))$ is connected, and hence so is $(F \downarrow d)$. The isomorphism then follows from the observation that colimits of constant, connected diagrams are isomorphic to the value of the diagram.

We must argue that this is an isomorphism of De Morgan algebras when $X = \mathbb{I}$ and of distributive lattices when $X = \mathbb{F}$. Chasing an element through this string of isomorphisms, we send an element of the colimit $\mathsf{in}_{((c,I'),(f,g))}(x)$ to $X(g)(x)$. One can verify that this preserves the relevant structure when X is appropriately specialized. We illustrate the simple case of interval endpoints: The 0 endpoint of $F_!(\mathbf{I}_{\mathcal{C}})$ at (d, I) is given by $\mathsf{in}_{(f_0,\mathsf{id})}(0 : \mathbb{I}(I))$ where f_0 is an arbitrary object of the (necessarily non-empty) category $(F \downarrow d)$. It is clear that this pair is mapped to $0 : \mathbf{I}_{\mathcal{D}}(d, I)$ via the morphisms above. □

We can package all of the above results into the following:

Theorem 4.15. *Let $f : \mathcal{M} \longrightarrow \mathbf{Cat}$ be a strict 2-functor, write $F^*(\mu)$, $F_!(\mu)$, and $F_*(\mu)$ for the precomposition, left Kan extension, and right Kan extension respectively of $f(\mu) \times \mathsf{id}_\square$, and write F^*, $F_!$, and F_* for the induced pseudofunctors.*

- *The network of morphisms of LOPS topoi given by the adjunctions $F^*(\mu) \dashv F_*(\mu)$ induces a model of $\mathsf{MTT}_\square$ over $\mathcal{M}$ with modal context structure equal to F^*.*
- *The network of morphisms of LOPS topoi given by the adjunctions $F_!(\mu) \dashv F^*(\mu)$ induces a model of $\mathsf{MTT}_\square$ over $\mathcal{M}^{\mathsf{coop}}$ with modal context structure pseudonaturally equivalent to $F_!$ if each $F_!(\mu)$ is lex.*

Proof. By Theorem 4.10, it is sufficient to show that F_* (respectively F^*) is a cubical MTT cosmos. By Proposition 4.11, each $\mathbf{PSh}(f(m) \times \square)$ is a LOPS topos, and each adjunction $F^*(\mu) \dashv F_*(\mu)$ (respectively $F_!(\mu) \dashv F^*(\mu)$) is a morphism of LOPS topoi, and we thus need only verify that the interval and face lattice isomorphisms cohere pseudonaturally.

Consider first the case of the adjunctions $F^*(\mu) \dashv F_*(\mu)$. In this case, each interval (respectiely face lattice) isomorphism is the identity, and thus since the left adjoints form a strict 2-functor, the interval objects (respectively face lattice objects) form a strict 2-natural transformation, which in particular is also a pseudonatural transformation.

Consider next the case of the adjunctions $F_!(\mu) \dashv F^*(\mu)$. To prove pseudonaturality, we need to prove coherence with identity, composition, and 2-cells. The proofs are similar, and we illustrate them with the identity case. Since $F_!(\mu)$ is a pseudofunctor, there is a natural isomorphism $F_!(1_m) \cong \mathsf{id}_{\mathbf{PSh}(f(m)\times\square)}$. We must prove that at the interval (respectively face lattice) object, this is the same isomorphism as the one constructed in Proposition 4.11.

The isomorphism in the proof of Proposition 4.11, $F_!(\mu)(\pi^*_{\mathcal{C}_m}(X))(d, I) \cong \pi^*_{f(m)}(X)(d, I)$, may be specified as follows: It is uniquely defined by its value upon precomposition with the components of the colimit $\mathsf{colim}_{((c,I'),\_):(f(\mu)\times\mathsf{id}\downarrow(d,I))} \pi^*_{f(m)}(X)(c, I')$, and this value is for each $c : f(m)$, $I' : \square$, and $(g, \iota) : (d, I) \longrightarrow (f(\mu)(c), I')$ equal to

$$X(\iota) : \pi^*_{f(m)}(X)(c, I') = X(I') \longrightarrow X(I) = \pi^*_{f(n)}(X)(d, I).$$

The isomorphism $F_!(1_m) \cong \mathsf{id}$ is constructed from the fact that both $F_!(1_m)$ and id are left adjoints to $F^*(1_m) = \mathsf{id}_{\mathbf{PSh}(f(m)\times\square)}$. Concretely, it is the counit:

$$\epsilon_{1_m} : F_!(1_m) = F_!(1_m) \circ F^*(1_m) \longrightarrow \mathsf{id}_{\mathbf{PSh}(f(m)\times\square)}$$

Precomposing with the components of the colimit $\mathsf{colim}_{((c,I'),\_):(f(\mu)\times\mathsf{id}\downarrow(d,I))} \pi^*_{f(m)}(X)(c, I')$, we get $X(\iota)$ as before, which shows that the two isomorphisms are the same, and thus the identity condition for pseudonaturality is satisfied. □

5. Proving and programming with guarded recursion

We now turn from theory to practice[8] and consider *guarded* $\mathsf{MTT}_\square$. We briefly recall guarded recursion. The core idea of guarded recursion [Nak00] is to use a modality $\blacktriangleright$ (pronounced 'later') to isolate recursively produced data to prevent its use until work is done, thereby ensuring productivity. This modality is equipped with operations making it into an applicative functor [MP08] which satisfies *Löb induction*, a powerful guarded fixed-point principle:

$$\mathsf{next} : A \longrightarrow \blacktriangleright A \qquad (\circledast) : \blacktriangleright(A \to B) \longrightarrow ((\blacktriangleright A) \to (\blacktriangleright B)) \qquad \mathsf{lob} : (\blacktriangleright A \to A) \longrightarrow A$$

In particular, lob allows us to define an element of A recursively but because the recursively computed data is available only as $\blacktriangleright A$, the usual problems with fixed-points are avoided. We consider a variant of guarded recursion which further includes an idempotent comonad $\square$ along with an equivalence $\square \blacktriangleright A \simeq A$. This last property ensures that guarded type theory can construct *coinductive* types through Löb induction [CBGB15].

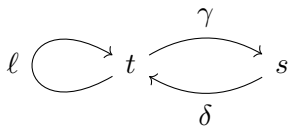
To encode guarded recursion in $\mathsf{MTT}_\square$, we instantiate the theory with a particular mode theory and extend it with a pair of axioms. As a result, we obtain a highly workable guarded type theory supporting the relevant modalities and operations. Similar work was done for *extensional* MTT in [GKNB21, Section 9]; here we show that the improved notion of equality in $\mathsf{MTT}_\square$ results in an improved experience.

Concretely, we work in the mode theory $\mathcal{M}_g$, a poset-enriched category which is concisely defined by Figure 4. Using the substitutions induced by 2-cells, we define:

$$\blacktriangleright A = \langle \ell \mid A[\{1 \le \ell\}] \rangle \qquad \mathsf{next}(x) = \mathsf{mod}_\ell(A[\{1 \le \ell\}])$$

While $\circledast$ is likewise definable, lob cannot be defined $\mathsf{MTT}_\square$ and must be axiomitized (Figure 5). In order to justify its inclusion, we provide a model of $\mathsf{MTT}_\square$ over $\mathcal{M}_g$ with Löb induction.

[8] Or at least, slightly more practice-adjacent theory!



$$\delta \circ \gamma \leq 1 \quad 1 = \gamma \circ \delta$$
$$1 \leq \ell \quad \gamma = \gamma \circ \ell$$

Figure 4: $\mathcal{M}_g$: a mode theory for guarded recursion. Reproduced from fig. 11 of [GKNB21]

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} t \qquad \Gamma \vdash A \mathbin{@} t}{\Gamma \vdash \mathsf{lob} : (\blacktriangleright A \to A) \to A \mathbin{@} t} \qquad \frac{\Gamma\ \mathsf{cx} \mathbin{@} t \qquad \Gamma \vdash A \mathbin{@} t \qquad \Gamma \vdash M : \blacktriangleright A \to A \mathbin{@} t}{\Gamma \vdash \mathsf{lob}(M) = M(\mathsf{next}(\mathsf{lob}(M))) : A \mathbin{@} t}$$

Figure 5: The rules of Löb induction.

5.1. **Soundness of Löb induction in $\mathsf{MTT}_\square$.** Letting ω be the poset category for the first infinite ordinal and 1 the terminal category, we define the strict 2-functor $f : \mathcal{M}_g \longrightarrow \mathbf{Cat}$ by

$$f(t) = \omega \qquad f(s) = 1 \qquad f(\delta)(*) = 0 \qquad f(\ell)(n) = n + 1 \qquad f(\gamma)(n) = *$$

From this, we define the pseudofunctor $F : \mathcal{M}_g \longrightarrow \mathbf{Cat}$ by $F(m) = \mathbf{PSh}(f(m) \times \square)$ and $F(\mu) = (f(\mu) \times \mathsf{id}_\square)_*$, which by Theorem 4.15 induces a model of $\mathsf{MTT}_\square$ $\widehat{F}$. This model is almost the same as the model defined in [GKNB21, Section 9.2], but $\widehat{F}$ uses **cSet**-valued presheaves. Since the cubical and modal aspects of $\mathsf{MTT}_\square$ are orthogonal, considerations in the **Set**-based model that do not involve identity types carry over to $\widehat{F}$. In particular, because Löb induction holds in the **Set**-based model, it also holds in $\widehat{F}$.[9]

5.2. **Programming with guarded $\mathsf{MTT}_\square$.** To see that $\mathsf{MTT}_\square$ can not merely replicate but also improve on work done in MTT, we now show that Löb induction not only gives a fixpoint but a unique one. In [GKNB20, Theorem 9.5] this is proven for extensional MTT (by introducing equality reflection), but because of modal extensionality, we can now prove it with nothing but $\mathsf{MTT}_\square$ and Löb induction. Similarly, the results from [GKNB21, Section 9.4] about guarded and coinductive streams in guarded MTT may also be proven in guarded $\mathsf{MTT}_\square$ without equality reflection.

Theorem 5.1. *$\mathsf{lob}(M)$ is the unique guarded fixpoint of $M : \blacktriangleright \mathsf{El}(A) \longrightarrow \mathsf{El}(A)$, i.e.*

$$(A : \mathsf{U})(x : \mathsf{El}(A)) \to \mathsf{Path}_{\mathsf{El}(A)}(M(\mathsf{next}(x)), x) \to \mathsf{Path}_{\mathsf{El}(A)}(\mathsf{lob}(M), x)$$

Proof. Supposing $A : \mathsf{U}$, we intend to use Löb induction to find a term of

$$(x : \mathsf{El}(A)) \to \mathsf{Path}_{\mathsf{El}(A)}(M(\mathsf{next}(x)), x) \to \mathsf{Path}_{\mathsf{El}(A)}(\mathsf{lob}(M), x)$$

To this end, given terms $f : \blacktriangleright((x : \mathsf{El}(A)) \to \mathsf{Path}_{\mathsf{El}(A)}(M(\mathsf{next}(x)), x) \to \mathsf{Path}_{\mathsf{El}(A)}(\mathsf{lob}(M), x))$, $x : \mathsf{El}(A)$, and $p : \mathsf{Path}_{\mathsf{El}(A)}(M(\mathsf{next}(x)), x)$, we must define a term of $\mathsf{Path}_{\mathsf{El}(A)}(\mathsf{lob}(M), x)$. We can construct the term

$$f \circledast \mathsf{next}(x) \circledast \mathsf{next}(p) : \blacktriangleright \mathsf{Path}_{\mathsf{El}(A)}(\mathsf{lob}(M), x).$$

By Theorem 3.1, this gives a term of $\mathsf{Path}_{\blacktriangleright \mathsf{El}(A)}(\mathsf{next}(\mathsf{lob}(M)), \mathsf{next}(x))$. Using that function application preserves paths and that $\mathsf{lob}(M)$ is a guarded fix point we then obtain the paths

$$\mathsf{lob}(M) \equiv M(\mathsf{next}(\mathsf{lob}M)) \equiv M(\mathsf{next}(x)) \equiv x. \qquad \square$$

[9] This can also be verified by hand as is done in [BBC+19].

6. Related work

$\mathsf{MTT}_\square$ builds upon two distinct strands of work: cubical and modal type theories. Even though both of these lines of research are ongoing, several proposals have already been made which combine elements of both.

Modal homotopy type theory. Several version of homotopy type theory extended with modalities have been proposed [Shu18, RFL21]. These type theories aim to increase the expressivity of HoTT and allow it to better capture some aspects of homotopy theory. Unlike $\mathsf{MTT}_\square$, however, these theories tend to be specialized to various modal situations. They build in the structure of one specific modality and are hand-crafted to have manageable syntax for that situation. In contrast, $\mathsf{MTT}_\square$ follows MTT and works for a *class* of modalities, and provides usable syntax in each case. Moreover, prior type theories in this tradition expand "book HoTT" [Uni13] and therefore do not enjoy the good computational properties we conjecture for $\mathsf{MTT}_\square$.

Modal cubical type theory. In order to extend cubical type theory with an *internal* notion of parametricity, Cavallo [Cav21, Part IV] has proposed a variant of (cartesian) cubical type theory extended with connectives and a handful of modalities to capture parametricity. Like $\mathsf{MTT}_\square$, this *cohesive parametric cubical type theory* combines cubical type theory with Fitch-style modalities. While morally the system is a specialization of $\mathsf{MTT}_\square$ to a cohesive collection of modalities, Cavallo takes advantage of several specifics of the intended model to add various equations to the theory.

Separately, another Fitch-style type theory, *clocked type theory*, has been extended to a cubical basis [KMV21]. This theory is used to present guarded recursion, similarly to section 5. Unlike guarded $\mathsf{MTT}_\square$, guarded clocked type theory includes several specialized axioms, a more sophisticated collection of guarded modalities, and an account of the interaction of HITs with parts of the modal machinery.

The extra equations and properties of the modalities in both systems prevent $\mathsf{MTT}_\square$ from directly recovering either parametric cubical type theory or clocked cubical type theory. The core aspects of both, however, are similar to $\mathsf{MTT}_\square$ and we believe that $\mathsf{MTT}_\square$ gives a means of systematically generalizing these calculi to other modal situations.

7. Conclusions

We contribute $\mathsf{MTT}_\square$, a general modal type theory based on cubical type theory and MTT. The system can be instantiated to a number of modal situations while still maintaining computationally effective interpretations of univalence and function extensionality.

While in this work we have introduced the theory and characterized a class of models for it, in the future we hope to investigate further metatheoretic properties of the system. In particular, both MTT and cubical type theory enjoy normalization [Gra21, SA21], and we conjecture that these proofs can be combined and generalized to apply to $\mathsf{MTT}_\square$. The introduction of cubical cosmoi takes the first step in this direction: cosmoi are a crucial ingredient of the proof of normalization for MTT. In a separate direction, we hope to investigate the behavior of more of the mode-local structure of cubical type theory such as *higher inductive types* and other novelties of cubical type theories.

References

[ABC+21] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Syntax and models of cartesian cubical type theory, 2021. To appear, Mathematical Structures in Computer Science. URL: https://www.cs.cmu.edu/~cangiuli/papers/abcfhl.pdf.

[Awo18] Steve Awodey. Natural models of homotopy type theory. *Mathematical Structures in Computer Science*, 28(2):241–286, 2018. arXiv:1406.3219, doi:10.1017/S0960129516000268.

[BBC+19] Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. Guarded cubical type theory. *Journal of Automated Reasoning*, (63):211–253, 2019.

[BCM+20] Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118–138, 2020. arXiv:1804.05236, doi:10.1017/S0960129519000197.

[BMSS12] Lars Birkedal, Rasmus Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4), 2012. doi:10.2168/LMCS-8(4:1)2012.

[Car78] John Cartmell. *Generalised Algebraic Theories and Contextual Categories*. PhD thesis, University of Oxford, 1978.

[Cav21] Evan Cavallo. *Higher inductive types and internal parametricity for cubical type theory*. PhD thesis, Carnegie Mellon University, 2021.

[CBGB15] Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. Programming and reasoning with guarded recursion for coinductive types. In Andrew Pitts, editor, *Foundations of Software Science and Computation Structures*, pages 407–421. Springer Berlin Heidelberg, 2015.

[CCHM18] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:34, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8475, doi:10.4230/LIPIcs.TYPES.2015.5.

[Clo18] Ranald Clouston. Fitch-Style Modal Lambda Calculi. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, pages 258–275. Springer International Publishing, 2018.

[CRS21] Thierry Coquand, Fabian Ruch, and Christian Sattler. Constructive sheaf models of type theory. *Mathematical Structures in Computer Science*, page 1–24, 2021. doi:10.1017/S0960129521000359.

[Dyb96] Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs*, pages 120–134, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. doi:10.1007/3-540-61780-9_66.

[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.

[GKNB21] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal Dependent Type Theory. *Logical Methods in Computer Science*, Volume 17, Issue 3, July 2021. URL: https://lmcs.episciences.org/7713, doi:10.46298/lmcs-17(3:11)2021.

[Gra21] Daniel Gratzer. Normalization for multimodal type theory. *CoRR*, abs/2106.01414, 2021. URL: https://arxiv.org/abs/2106.01414, arXiv:2106.01414.

[KL21] Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of Univalent Foundations (after Voevodsky). 23:2071–2126, 2021. arXiv:1211.2851, doi:10.4171/JEMS/1050.

[KMV21] Magnus Baunsgaard Kristensen, Rasmus Ejlers Møgelberg, and Andrea Vezzosi. Greatest hits: Higher inductive types in coinductive definitions via induction under clocks. 2021. URL: https://arxiv.org/abs/2102.01969, arXiv:2102.01969.

[LOPS18] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal Universes in Models of Homotopy Type Theory. In H. Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, Leibniz International Proceedings in

Informatics (LIPIcs), pages 22:1–22:17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. arXiv:1801.07664, doi:10.4230/LIPIcs.FSCD.2018.22.

[LS16] Daniel R. Licata and Michael Shulman. Adjoint Logic with a 2-Category of Modes. In Sergei Artemov and Anil Nerode, editors, *Logical Foundations of Computer Science*, pages 219–235. Springer International Publishing, 2016. doi:10.1007/978-3-319-27683-0_16.

[MP08] Conor McBride and Ross Paterson. Applicative programming with effects. *Journal of Functional Programming*, 18(1), 2008. URL: http://www.staff.city.ac.uk/~ross/papers/Applicative.pdf, doi:10.1017/S0956796807006326.

[Nak00] H. Nakano. A modality for recursion. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*, pages 255–266. IEEE Computer Society, 2000.

[ND18] Andreas Nuyts and Dominique Devriese. Degrees of relatedness: A unified framework for parametricity, irrelevance, ad hoc polymorphism, intersections, unions and algebra in dependent type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18. ACM, 2018. doi:10.1145/3209108.3209119.

[OP18] Ian Orton and Andrew M. Pitts. Axioms for Modelling Cubical Type Theory in a Topos. *Logical Methods in Computer Science*, 14(4), 2018. arXiv:1712.04864, doi:10.23638/LMCS-14(4:23)2018.

[RFL21] Mitchell Riley, Eric Finster, and Daniel R. Licata. Synthetic spectra via a monadic and comonadic modality, 2021. arXiv:2102.04099.

[SA21] Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '21, New York, NY, USA, 2021. ACM.

[Shu18] Michael Shulman. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *Mathematical Structures in Computer Science*, 28(6):856–941, 2018. doi:10.1017/S0960129517000147.

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

Appendix A. Rules of $\mathsf{MTT}_\square$

We here present the official syntax and rules of $\mathsf{MTT}_\square$. For the sake of brevity, we omit a number the rules, especially those lifted from MTT or CTT; in particular, we omit the following:

- The rules for dependent sums, booleans, universes.
- The equations stating that the interval is a De Morgan algebra.
- The equations stating that $(-)^\mu$ for interval terms is a morphism of De Morgan algebras.
- The equations stating that that face lattice is a bounded distributive lattice,
- The equations stating that $(-)^\mu$ for faces is a morphism of bounded lattices,
- Miscellaneous equations commuting substitutions past term formers or governing the composition of substitutions.

At the end there is a section on derived definitions some of which we will use throughout to ease notation.

Context formation.

$$\textsc{cx/emp}\ \frac{}{\mathbf{1}\ \mathsf{cx} \mathbin{@} m} \qquad \textsc{cx/lock}\ \frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu\}\ \mathsf{cx} \mathbin{@} n} \qquad \textsc{cx/ext-type}\ \frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n}{\Gamma.(\mu \mid A)\ \mathsf{cx} \mathbin{@} m}$$

$$\textsc{cx/ext-int}\ \frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\mathbb{I}_m\ \mathsf{cx} \mathbin{@} m} \qquad \textsc{cx/face-res}\ \frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.[\phi]\ \mathsf{cx} \mathbin{@} m}$$

Context equality.

CX-EQ/COMP-LOCK

$$\frac{\mu : n \longrightarrow m \qquad \nu : o \longrightarrow n \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu \circ \nu\} = \Gamma.\{\mu\}.\{\nu\}\ \mathsf{cx} \mathbin{@} o}$$

CX-EQ/ID-LOCK

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{1\} = \Gamma\ \mathsf{cx} \mathbin{@} m}$$

Substitution formation.

SB/COMP

$$\frac{\Gamma, \Delta, \Xi\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta \vdash \xi : \Xi \mathbin{@} m}{\Gamma \vdash \xi \circ \delta : \Xi \mathbin{@} m}$$

SB/ID

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma \vdash \mathsf{id} : \Gamma \mathbin{@} m}$$

SB/EMP

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma \vdash ! : \mathbf{1} \mathbin{@} m}$$

SB/WEAK-TYPE

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n}{\Gamma.(\mu \mid A) \vdash \uparrow : \Gamma \mathbin{@} m}$$

SB/WEAK-INT

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\mathbb{I}_m \vdash \uparrow^i : \Gamma \mathbin{@} m}$$

SB/WEAK-RES

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.[\phi] \vdash \uparrow^{\phi} : \Gamma \mathbin{@} m}$$

SB/LOCK

$$\frac{\mu : n \longrightarrow m \qquad \Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m}{\Gamma.\{\mu\} \vdash \delta.\{\mu\} : \Delta.\{\mu\} \mathbin{@} n}$$

SB/KEY

$$\frac{\mu, \nu : n \longrightarrow m \qquad \alpha : \nu \longrightarrow \mu \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu\} \vdash \{\alpha\}_{\Gamma} : \Gamma.\{\nu\} \mathbin{@} n}$$

SB/EXT-TYPE

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \begin{array}{c}\mu : n \longrightarrow m \\ \Delta.\{\mu\} \vdash A \mathbin{@} n\end{array} \qquad \Gamma.\{\mu\} \vdash a : A[\delta.\{\mu\}] \mathbin{@} n}{\Gamma \vdash \delta.a : \Delta.(\mu \mid A) \mathbin{@} m}$$

SB/EXT-INT

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash \delta.r : \Delta.\mathbb{I}_m \mathbin{@} m}$$

SB/FACE-RES

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \phi[\delta] = \top : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \delta.[\phi] : \Delta.[\phi] \mathbin{@} m}$$

SB/EXC-INT-INV

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu\}.\mathbb{I}_n \vdash \sigma^{\mu} : \Gamma.\mathbb{I}_m.\{\mu\} \mathbin{@} n}$$

SB/EXC-FACE-INV

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.\{\mu\}.[\phi^{\mu}] \vdash \tau^{\mu} : \Gamma.[\phi].\{\mu\} \mathbin{@} n}$$

Substitution equality.

SB-EQ/COMP-LOCK
$$\frac{\mu : n \longrightarrow m \qquad \nu : o \longrightarrow n \qquad \Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m}{\Gamma.\{\mu \circ \nu\} \vdash \delta.\{\mu \circ \nu\} = \delta.\{\mu\}.\{\nu\} : \Delta.\{\mu \circ \nu\} \mathbin{@} o}$$

SB-EQ/ID-LOCK
$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m}{\Gamma \vdash \delta.\{1\} = \delta : \Delta \mathbin{@} m}$$

SB-EQ/LOCK-COMP
$$\frac{\mu : n \longrightarrow m \qquad \Gamma, \Delta, \Xi\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta \vdash \xi : \Xi \mathbin{@} m}{\Gamma.\{\mu\} \vdash (\xi \circ \delta).\{\mu\} = \xi.\{\mu\} \circ \delta.\{\mu\} : \Xi.\{\mu\} \mathbin{@} n}$$

SB-EQ/LOCK-ID
$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu\} \vdash \mathsf{id}.\{\mu\} = \mathsf{id} : \Gamma.\{\mu\} \mathbin{@} n}$$

SB-EQ/ID-KEY
$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu\} \vdash \{1_\mu\}_\Gamma = \mathsf{id} : \Gamma.\{\mu\} \mathbin{@} n}$$

SB-EQ/NAT-KEY
$$\frac{\mu, \nu : n \longrightarrow m \qquad \alpha : \nu \longrightarrow \mu \qquad \Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m}{\Gamma.\{\mu\} \vdash \{\alpha\}_\Delta \circ \delta.\{\mu\} = \delta.\{\nu\} \circ \{\alpha\}_\Gamma : \Delta.\{\nu\} \mathbin{@} n}$$

SB-EQ/COMP-KEY
$$\frac{\mu, \nu, \rho : n \longrightarrow m \qquad \alpha : \nu \longrightarrow \mu \qquad \beta : \rho \longrightarrow \nu \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu\} \vdash \{\alpha \circ \beta\}_\Gamma = \{\alpha\}_\Gamma \circ \{\beta\}_\Gamma : \Gamma.\{\rho\} \mathbin{@} n}$$

SB-EQ/WHISK-KEY
$$\frac{\mu_0, \mu_1 : n \longrightarrow m \qquad \nu_0, \nu_1 : o \longrightarrow n \qquad \alpha : \mu_1 \longrightarrow \mu_0 \qquad \beta : \nu_1 \longrightarrow \nu_0 \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu_0 \circ \nu_0\} \vdash \{\alpha \star \beta\}_\Gamma = \{\alpha\}_\Gamma.\{\nu_1\} \circ \{\beta\}_{\Gamma.\{\mu_0\}} : \Gamma.\{\mu_1 \circ \nu_1\} \mathbin{@} o}$$

SB-EQ/EXT-TYPE-BETA
$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma.\{\mu\} \vdash a : A[\delta.\{\mu\}] \mathbin{@} n}{\Gamma \vdash \uparrow \circ \delta.a = \delta : \Delta \mathbin{@} m}$$

SB-EQ/EXT-TYPE-ETA
$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Delta.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma \vdash \delta : \Delta.(\mu \mid A) \mathbin{@} m}{\Gamma \vdash \delta = (\uparrow \circ \delta).\mathbf{v}_0[\delta] : \Delta.(\mu \mid A) \mathbin{@} m}$$

SB-EQ/EXT-INT-BETA
$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash \uparrow^i \circ \delta.r = \delta : \Delta \mathbin{@} m}$$

SB-EQ/EXT-INT-ETA
$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta.\mathbb{I}_m \mathbin{@} m}{\Gamma \vdash \delta = (\uparrow^i \circ \delta).\mathbf{v}_0^i[\delta] : \Delta.\mathbb{I}_m \mathbin{@} m}$$

SB-EQ/EXC-INT-LEFT-INV
$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\mathbb{I}_m.\{\mu\} \vdash \sigma^\mu \circ \sigma_\mu = \mathsf{id} : \Gamma.\mathbb{I}_m.\{\mu\} \mathbin{@} n}$$

SB-EQ/EXC-INT-RIGHT-INV
$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\{\mu\}.\mathbb{I}_m \vdash \sigma_\mu \circ \sigma^\mu = \mathsf{id} : \Gamma.\{\mu\}.\mathbb{I}_m \mathbin{@} n}$$

SB-EQ/EXC-FACE-LEFT-INV

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.[\phi].\{\mu\} \vdash \tau^\mu \circ \tau_\mu = \mathsf{id} : \Gamma.[\phi].\{\mu\} \mathbin{@} n}$$

SB-EQ/EXC-FACE-RIGHT-INV

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.\{\mu\}.[\phi^\mu] \vdash \tau_\mu \circ \tau^\mu = \mathsf{id} : \Gamma.\{\mu\}.[\phi^\mu] \mathbin{@} n}$$

SB-EQ/FACE-RES-UNIQ

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Delta \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \delta : \Gamma.[\phi] \mathbin{@} m}{\Gamma \vdash \delta = (\uparrow^\phi \circ \delta).[\phi] : \Gamma.[\phi] \mathbin{@} m}$$

SB-EQ/FACE-RES-BIN

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta, \xi : \Delta \mathbin{@} m \qquad \Gamma \vdash \phi, \psi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \phi \vee \psi = \top : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\phi] \vdash \delta \circ \uparrow^\phi = \xi \circ \uparrow^\phi : \Delta \mathbin{@} m \qquad \Gamma.[\psi] \vdash \delta \circ \uparrow^\psi = \xi \circ \uparrow^\psi : \Delta \mathbin{@} m}{\Gamma \vdash \delta = \xi : \Delta \mathbin{@} m}$$

SB-EQ/FACE-RES-NULL

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta, \xi : \Delta \mathbin{@} m \qquad \Gamma \vdash \bot = \top : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \delta = \xi : \Delta \mathbin{@} m}$$

Interval formation.

INT/JOIN

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r, s : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash r \vee s : \mathbb{I}_m \mathbin{@} m}$$

INT/MEET

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r, s : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash r \wedge s : \mathbb{I}_m \mathbin{@} m}$$

INT/BOT

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma \vdash 0 : \mathbb{I}_m \mathbin{@} m}$$

INT/TOP

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma \vdash 1 : \mathbb{I}_m \mathbin{@} m}$$

INT/INV

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash (1 - r) : \mathbb{I}_m \mathbin{@} m}$$

INT/EXC

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma.\{\mu\} \vdash r^\mu : \mathbb{I}_n \mathbin{@} n}$$

INT/VAR

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\mathbb{I}_m \vdash \mathbf{v}_0^i : \mathbb{I}_m \mathbin{@} m}$$

INT/SB

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash r[\delta] : \mathbb{I}_m \mathbin{@} m}$$

Interval equality.

INT-EQ/EXT-INT-BETA

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash \mathbf{v}_0^i[\delta.r] = r : \mathbb{I}_m \mathbin{@} m}$$

INT-EQ/RES-EQ

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma.[(r = 0)] \vdash r[\uparrow^{(r=0)}] = 0 : \mathbb{I}_m \mathbin{@} m}$$

INT-EQ/EXC-COMP

$$\frac{\mu : n \longrightarrow m \qquad \nu : o \longrightarrow n \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma.\{\mu\}.\{\nu\} \vdash r^{\mu \circ \nu} = (r^\mu)^\nu : \mathbb{I}_o \mathbin{@} o}$$

INT-EQ/EXC-ID

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash r^1 = r : \mathbb{I}_m \mathbin{@} m}$$

INT-EQ/EXC-KEY

$$\frac{\mu, \nu : n \longrightarrow m \qquad \alpha : \nu \longrightarrow \mu \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma.\{\mu\} \vdash r^\nu[\{\alpha\}_\Gamma] = r^\mu : \mathbb{I}_n \mathbin{@} n}$$

INTE-EQ/EXC-SUB

$$\frac{\nu : n \longrightarrow m \qquad \Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma.\{\mu\} \vdash r^\mu[\delta.\{\mu\}] = r[\delta]^\mu : \mathbb{I}_n \mathbin{@} n}$$

INT-EQ/FACE-RES-BIN

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r, s : \mathbb{I}_m \mathbin{@} m \qquad \Gamma \vdash \phi, \psi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \phi \vee \psi = \top : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\phi] \vdash r[\uparrow^\phi] = s[\uparrow^\phi] : \mathbb{I}_m \mathbin{@} m \qquad \Gamma.[\psi] \vdash r[\uparrow^\psi] = s[\uparrow^\psi] : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash r = s : \mathbb{I}_m \mathbin{@} m}$$

INT-EQ/FACE-RES-NULL

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r, s : \mathbb{I}_m \mathbin{@} m \qquad \Gamma \vdash \bot = \top : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash r = s : \mathbb{I}_m \mathbin{@} m}$$

Face formation.

FACE/EQ

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash (r = 0) : \mathbb{F}_m \mathbin{@} m}$$

FACE/JOIN

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi, \psi : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \phi \vee \psi : \mathbb{F}_m \mathbin{@} m}$$

FACE/MEET

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi, \psi : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \phi \wedge \psi : \mathbb{F}_m \mathbin{@} m}$$

FACE/BOT

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma \vdash \bot : \mathbb{F}_m \mathbin{@} m}$$

FACE/TOP

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma \vdash \top : \mathbb{F}_m \mathbin{@} m}$$

FACE/EXC

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.\{\mu\} \vdash \phi^\mu : \mathbb{F}_n \mathbin{@} n}$$

FACE/SB

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \phi[\delta] : \mathbb{F}_m \mathbin{@} m}$$

Face equality.

FACE-EQ/NON-CONTR
$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash (r = 0) \wedge ((1 - r) = 0) = \bot : \mathbb{F}_m \mathbin{@} m}$$

FACE-EQ/EXC-COMP
$$\frac{\mu : n \longrightarrow m \qquad \nu : o \longrightarrow n \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.\{\mu\}.\{\nu\} \vdash \phi^{\mu \circ \nu} = (\phi^{\mu})^{\nu} : \mathbb{F}_o \mathbin{@} o}$$

FACE-EQ/EXC-ID
$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \phi^1 = \phi : \mathbb{F}_m \mathbin{@} m}$$

FACE-EQ/EXC-KEY
$$\frac{\mu, \nu : n \longrightarrow m \qquad \alpha : \nu \longrightarrow \mu \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.\{\mu\} \vdash \phi^{\nu}[\{\alpha\}_{\Gamma}] = \phi^{\mu} : \mathbb{I}_n \mathbin{@} n}$$

FACE-EQ/EXC-EQ
$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash (r = 0)^{\mu} = (r^{\mu} = 0) : \mathbb{F}_n \mathbin{@} n}$$

FACE-EQ/EXC-SUB
$$\frac{\mu : n \longrightarrow m \qquad \Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.\{\mu\} \vdash \phi^{\mu}[\delta.\{\mu\}] = \phi[\delta]^{\mu} : \mathbb{F}_n \mathbin{@} n}$$

FACE-EQ/RES-EQ-TOP
$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.[\phi] \vdash \phi[\uparrow^{\phi}] = \top : \mathbb{F}_m \mathbin{@} m}$$

FACE-EQ/EQ-ZERO
$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma \vdash (0 = 0) = \top : \mathbb{F}_m \mathbin{@} m}$$

FACE-EQ/FACE-RES-BIN
$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi, \psi, \chi_0, \chi_1 : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \phi \vee \psi = \top : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\phi] \vdash \chi_0[\uparrow^{\phi}] = \chi_1[\uparrow^{\phi}] : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\psi] \vdash \chi_0[\uparrow^{\psi}] = \chi_1[\uparrow^{\psi}] : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \chi_0 = \chi_1 : \mathbb{F}_m \mathbin{@} m}$$

FACE-EQ/FACE-RES-NULL
$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \chi_0, \chi_1 : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \bot = \top : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \chi_0 = \chi_1 : \mathbb{F}_m \mathbin{@} m}$$

Type formation.

TYPE/PI
$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma.(\mu \mid A) \vdash B \mathbin{@} m}{\Gamma \vdash (\mu \mid A) \to B \mathbin{@} m}$$

TYPE/PATH
$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash A \mathbin{@} m \qquad \Gamma \vdash a : A[\mathsf{id}.0] \mathbin{@} m \qquad \Gamma \vdash b : A[\mathsf{id}.1] \mathbin{@} m}{\Gamma \vdash \mathsf{Path}_A(a, b) \mathbin{@} m}$$

TYPE/MOD

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n}{\Gamma \vdash \langle \mu \mid A \rangle \mathbin{@} m}$$

TYPE/SYS

$$\frac{\begin{array}{c}\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi, \psi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \phi \vee \psi = \top : \mathbb{F}_m \mathbin{@} m \\ \Gamma.[\phi] \vdash A \mathbin{@} m \qquad \Gamma.[\psi] \vdash B \mathbin{@} m \qquad \Gamma.[\phi \wedge \psi] \vdash A[\uparrow^{\phi \wedge \psi} .[\phi]] = B[\uparrow^{\phi \wedge \psi} .[\psi]] \mathbin{@} m\end{array}}{\Gamma \vdash \{\ \phi\ A, \psi\ B\ \} \mathbin{@} m}$$

TYPE/SB

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta \vdash A \mathbin{@} m}{\Gamma \vdash A[\delta] \mathbin{@} m}$$

Type equality.

TYPE-EQ/SYS-TOP

$$\frac{\begin{array}{c}\Gamma\ \mathsf{cx} \mathbin{@} m \\ \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\top] \vdash A \mathbin{@} m \qquad \Gamma.[\phi] \vdash B \mathbin{@} m \qquad \Gamma.[\phi] \vdash A[\uparrow^{\phi} .[\top]] = B \mathbin{@} m\end{array}}{\Gamma \vdash \{\ \top\ A, \phi\ B\ \} = A[\mathsf{id}.[\top]] \mathbin{@} m}$$

TYPE-EQ/FACE-RES-BIN

$$\frac{\begin{array}{c}\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi, \psi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \phi \vee \psi = \top : \mathbb{F}_m \mathbin{@} m \\ \Gamma \vdash A, B \mathbin{@} m \qquad \Gamma.[\phi] \vdash A[\uparrow^{\phi}] = B[\uparrow^{\phi}] \mathbin{@} m \qquad \Gamma.[\psi] \vdash A[\uparrow^{\psi}] = B[\uparrow^{\psi}] \mathbin{@} m\end{array}}{\Gamma \vdash A = B \mathbin{@} m}$$

TYPE-EQ/FACE-RES-NULL

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \bot = \top : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash A, B \mathbin{@} m}{\Gamma \vdash A = B \mathbin{@} m}$$

Term formation.

TERM/PI-LAM

$$\frac{\begin{array}{c}\mu : n \longrightarrow m \\ \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma.(\mu \mid A) \vdash B \mathbin{@} m \qquad \Gamma.(\mu \mid A) \vdash b : B \mathbin{@} m\end{array}}{\Gamma \vdash \lambda(b) : (\mu \mid A) \to B \mathbin{@} m}$$

TERM/PI-APP

$$\frac{\begin{array}{c}\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \\ \Gamma.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma.(\mu \mid A) \vdash B \mathbin{@} m \qquad \Gamma \vdash f : (\mu \mid A) \to B \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash a : A \mathbin{@} n\end{array}}{\Gamma \vdash f(a) : B[\mathsf{id}.a] \mathbin{@} m}$$

TERM/PATH-ABS

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash A \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash a : A \mathbin{@} m}{\Gamma \vdash \lambda(a) : \mathsf{Path}_A(a[\mathsf{id}.0], a[\mathsf{id}.1]) \mathbin{@} m}$$

TERM/PATH-APP

$$\frac{\Gamma \vdash a : A[\mathsf{id}.0] \mathbin{@} m \qquad \Gamma \vdash b : A[\mathsf{id}.1] \mathbin{@} m \qquad \Gamma \vdash p : \mathsf{Path}_A(a,b) \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m \qquad \begin{array}{c}\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash A \mathbin{@} m\end{array}}{\Gamma \vdash p(r) : A[\mathsf{id}.r] \mathbin{@} m}$$

TERM/MOD-MOD

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma.\{\mu\} \vdash a : A \mathbin{@} n}{\Gamma \vdash \mathsf{mod}_\mu(a) : \langle \mu \mid A \rangle \mathbin{@} m}$$

TERM/MOD-LET

$$\frac{\mu : n \longrightarrow m \qquad \nu : o \longrightarrow n \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\}.\{\nu\} \vdash A \mathbin{@} o \qquad \Gamma.\{\mu\} \vdash a : \langle \nu \mid A \rangle \mathbin{@} n \qquad \begin{array}{c}\Gamma.(\mu \mid \langle \nu \mid A \rangle) \vdash B \mathbin{@} m \qquad \Gamma.(\mu \circ \nu \mid A) \vdash b : B[\uparrow.\mathsf{mod}_\nu(\mathbf{v}_0)] \mathbin{@} m\end{array}}{\Gamma \vdash \mathsf{let}_\mu\ \mathsf{mod}_\nu(\_) \leftarrow a\ \mathsf{in}\ b : B[\mathsf{id}.a] \mathbin{@} m}$$

TERM/SYS-BIN

$$\frac{\Gamma \vdash A \mathbin{@} m \qquad \Gamma \vdash \phi, \psi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \phi \vee \psi = \top : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\phi] \vdash a : A[\uparrow^\phi] \mathbin{@} m \qquad \begin{array}{c}\Gamma\ \mathsf{cx} \mathbin{@} m\end{array} \qquad \begin{array}{c}\Gamma.[\psi] \vdash b : A[\uparrow^\psi] \mathbin{@} m \qquad \Gamma.[\phi \wedge \psi] \vdash a[\uparrow^{\phi\wedge\psi}.[\phi]] = b[\uparrow^{\phi\wedge\psi}.[\psi]] : A[\uparrow^{\phi\wedge\psi}] \mathbin{@} m\end{array}}{\Gamma \vdash \{\ \phi\ a, \psi\ b\ \} : A \mathbin{@} m}$$

TERM/SYS-NULL

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash A \mathbin{@} m \qquad \Gamma \vdash \bot = \top : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \{\ \} : A \mathbin{@} m}$$

TERM/COMP

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash A \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\phi].\mathbb{I}_m \vdash u : A[\uparrow^\phi.\mathbb{I}_m] \mathbin{@} m \qquad \begin{array}{c}\Gamma \vdash u_0 : A[\mathsf{id}.0] \mathbin{@} m \qquad \Gamma.[\phi] \vdash u[\mathsf{id}.0] = u_0[\uparrow^\phi] : A[\uparrow^\phi.0] \mathbin{@} m\end{array}}{\Gamma \vdash \mathsf{comp}\,[\phi \mapsto u]\,u_0 : A[\mathsf{id}.1] \mathbin{@} m}$$

TERM/VAR

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n}{\Gamma.(\mu \mid A).\{\mu\} \vdash \mathbf{v}_0 : A[\uparrow.\{\mu\}] \mathbin{@} n}$$

TERM/SB

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta \vdash a : A \mathbin{@} m}{\Gamma \vdash a[\delta] : A[\delta] \mathbin{@} m}$$

Term equality.

TERM-EQ/PI-BETA

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma.(\mu \mid A) \vdash B \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash a : A \mathbin{@} n \qquad \Gamma.(\mu \mid A) \vdash b : B \mathbin{@} m}{\Gamma \vdash \lambda(b)(a) = b[\mathsf{id}.a] : B[\mathsf{id}.a] \mathbin{@} m}$$

TERM-EQ/PI-ETA

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma.(\mu \mid A) \vdash B \mathbin{@} m \qquad \Gamma \vdash f : (\mu \mid A) \to B \mathbin{@} m}{\Gamma \vdash f = \lambda(f[\uparrow](\mathbf{v}_0)) : (\mu \mid A) \to B \mathbin{@} m}$$

TERM-EQ/PATH-BETA

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash A \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash a : A \mathbin{@} m \qquad \Gamma \vdash r : \mathbb{I}_m \mathbin{@} m}{\Gamma \vdash \lambda(a)(r) = a[\mathsf{id}.r] : A[\mathsf{id}.r] \mathbin{@} m}$$

TERM-EQ/PATH-ETA

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash A \mathbin{@} m \qquad \Gamma \vdash a_0 : A[\mathsf{id}.0] \mathbin{@} m \qquad \Gamma \vdash a_1 : A[\mathsf{id}.1] \mathbin{@} m \qquad \Gamma \vdash p : \mathsf{Path}_A(a_0, a_1) \mathbin{@} m}{\Gamma \vdash p = \lambda(p[\uparrow^i](\mathbf{v}_0^i)) : \mathsf{Path}_A(a_0, a_1) \mathbin{@} m}$$

TERM-EQ/MOD-BETA

$$\frac{\mu : n \longrightarrow m \qquad \nu : o \longrightarrow n \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\{\mu\}.\{\nu\} \vdash A \mathbin{@} o \qquad \Gamma.(\mu \mid \langle \nu \mid A \rangle) \vdash B \mathbin{@} m \qquad \Gamma.\{\mu\}.\{\nu\} \vdash a : A \mathbin{@} o \qquad \Gamma.(\mu \circ \nu \mid A) \vdash b : B[\uparrow.\mathsf{mod}_\nu(\mathbf{v}_0)] \mathbin{@} m}{\Gamma \vdash \mathsf{let}_\mu\ \mathsf{mod}_\nu(\_) \leftarrow \mathsf{mod}_\nu(a)\ \mathsf{in}\ b = b[\mathsf{id}.a] : B[\mathsf{id}.\mathsf{mod}_\nu(a)] \mathbin{@} m}$$

TERM-EQ/EXT-TYPE-BETA

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m \qquad \Delta.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma.\{\mu\} \vdash a : A[\delta.\{\mu\}] \mathbin{@} n}{\Gamma.\{\mu\} \vdash \mathbf{v}_0[\delta.a.\{\mu\}] = a : A[\delta.\{\mu\}] \mathbin{@} m}$$

TERM-EQ/SYS-TOP

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash A \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\top] \vdash a : A[\uparrow^\top] \mathbin{@} m \qquad \Gamma.[\phi] \vdash b : A[\uparrow^\phi] \mathbin{@} m \qquad \Gamma.[\phi] \vdash a[\uparrow^\phi .[\top]] = b : A \mathbin{@} m}{\Gamma \vdash \{\ \top\ a, \phi\ b\ \} = a[\mathsf{id}.[\top]] : A \mathbin{@} m}$$

TERM-EQ/COMP-FACE

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma.\mathbb{I}_m \vdash A \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\phi].\mathbb{I}_m \vdash u : A[\uparrow^\phi .\mathbb{I}_m] \mathbin{@} m \qquad \Gamma \vdash u_0 : A[\mathsf{id}.0] \mathbin{@} m \qquad \Gamma.[\phi] \vdash u[\mathsf{id}.0] = u_0[\uparrow^\phi] : A[\uparrow^\phi.0] \mathbin{@} m \qquad \Gamma \vdash \phi = \top : \mathbb{F}_m \mathbin{@} m}{\Gamma \vdash \mathsf{comp}\,[\phi \mapsto u]\, u_0 = u[\mathsf{id}.[\phi].1] : A[\mathsf{id}.1] \mathbin{@} m}$$

TERM-EQ/COMP-MOD

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \mu : n \longrightarrow m \qquad \Gamma.\mathbb{I}_m.\{\mu\} \vdash A \mathbin{@} n \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\phi].\mathbb{I}_m.\{\mu\} \vdash u : A[\uparrow^\phi .\mathbb{I}_m.\{\mu\}] \mathbin{@} n \qquad \Gamma.\{\mu\} \vdash u_0 : A[\mathsf{id}.0.\{\mu\}] \mathbin{@} n \qquad \Gamma.[\phi].\{\mu\} \vdash u[\mathsf{id}.0.\{\mu\}] = u_0[\uparrow^\phi .\{\mu\}] : A[\uparrow^\phi.0.\{\mu\}] \mathbin{@} n}{\Gamma \vdash \mathsf{mod}_\mu(\mathsf{comp}\,[\phi^\mu \mapsto u[\sigma_\mu \circ \tau_\mu]]\, u_0) = \mathsf{comp}\,[\phi \mapsto \mathsf{mod}_\mu(u)]\, \mathsf{mod}_\mu(u_0) : \langle \mu \mid A \rangle[\mathsf{id}.1] \mathbin{@} m}$$

TERM-EQ/COMP-PI

$$\frac{\begin{array}{c}\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \mu : n \longrightarrow m \\ \Gamma.\mathbb{I}_m \vdash (\mu \mid A) \to B \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma.[\phi].\mathbb{I}_m \vdash f : ((\mu \mid A) \to B)[\uparrow^{\phi}.\mathbb{I}_m] \mathbin{@} m \\ \Gamma \vdash f_0 : ((\mu \mid A) \to B)[\mathsf{id}.0] \mathbin{@} m \qquad \Gamma.[\phi] \vdash f[\mathsf{id}.0] = f_0[\uparrow^{\phi}] : ((\mu \mid A) \to B)[\uparrow^{\phi}.0] \mathbin{@} m \\ \Gamma.\{\mu\} \vdash a_1 : A[\mathsf{id}.1.\{\mu\}] \mathbin{@} m \qquad w := (\mathsf{hfill}^i\,[\,]\,a_1[\sigma_\mu])[\sigma^\mu] : A[i/1-i] \qquad v := w(1-i) : A\end{array}}{\Gamma \vdash (\mathsf{comp}\,[\phi \mapsto f]\,f_0)(a_1) = \mathsf{comp}\,[\phi \mapsto f(v[\uparrow^{\phi}.\uparrow^{i}.\{\mu\}])]\,f_0(v[\mathsf{id}.0.\{\mu\}]) : B[\mathsf{id}.1] \mathbin{@} m}$$

TERM-EQ/FACE-RES-BIN

$$\frac{\begin{array}{c}\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi, \psi : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash \phi \vee \psi = \top : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash A \mathbin{@} m \\ \Gamma \vdash a, b : A \mathbin{@} m \qquad \Gamma.[\phi] \vdash a[\uparrow^{\phi}] = b[\uparrow^{\phi}] : A[\uparrow^{\phi}] \mathbin{@} m \qquad \Gamma.[\psi] \vdash a[\uparrow^{\psi}] = b[\uparrow^{\psi}] : A[\uparrow^{\psi}] \mathbin{@} m\end{array}}{\Gamma \vdash a = b : A \mathbin{@} m}$$

TERM-EQ/FACE-RES-NULL

$$\frac{\Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \bot = \top : \mathbb{F}_m \mathbin{@} m \qquad \Gamma \vdash A \mathbin{@} m \qquad \Gamma \vdash a, b : A \mathbin{@} m}{\Gamma \vdash a = b : A \mathbin{@} m}$$

Derived Definitions.

SB/PLUS-INT

$$\frac{\Gamma, \Delta\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \delta : \Delta \mathbin{@} m}{\Gamma.\mathbb{I}_m \vdash \delta.\mathbb{I}_m \coloneqq (\delta \circ \uparrow^i).\mathbf{v}_0^i : \Delta.\mathbb{I}_m \mathbin{@} m}$$

SB/EXC-INT

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m}{\Gamma.\mathbb{I}_m.\{\mu\} \vdash \sigma_\mu \coloneqq \uparrow^i.\{\mu\}.\mathbf{v}_0^\mu : \Gamma.\{\mu\}.\mathbb{I}_n \mathbin{@} m}$$

SB/EXC-FACE

$$\frac{\mu : n \longrightarrow m \qquad \Gamma\ \mathsf{cx} \mathbin{@} m \qquad \Gamma \vdash \phi : \mathbb{F}_m \mathbin{@} m}{\Gamma.[\phi].\{\mu\} \vdash \tau_\mu \coloneqq \uparrow^{\phi}.\{\mu\}.[\phi^\mu] : \Gamma.\{\mu\}.[\phi^\mu] \mathbin{@} m}$$

Appendix B. Models of MTT$_\Box$

Definition B.1. A modal context structure on a mode theory $\mathcal{M}$ is a strict 2-functor $[\![-]\!] : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ such that for each mode $m : \mathcal{M}$, $[\![m]\!]$ has a terminal object.

Definition B.2. A modal natural model on a modal context structure consists of

- for each mode $m : \mathcal{M}$, a presheaf $\mathcal{T}_m : \mathbf{PSh}([\![m]\!])$,
- for each mode $m : \mathcal{M}$, a presheaf $\widetilde{\mathcal{T}}_m : \mathbf{PSh}([\![m]\!])$,
- for each mode $m : \mathcal{M}$, a natural transformation $\tau_m : \widetilde{\mathcal{T}}_m \longrightarrow \mathcal{T}_m$,

such that

- for any modes $m, n : \mathcal{M}$ and each modality $\mu : n \longrightarrow m$, it holds that $[\![\mu]\!]^* \tau_n : [\![\mu]\!]^* \widetilde{\mathcal{T}}_n \longrightarrow [\![\mu]\!]^* \mathcal{T}_n$ is a representable natural transformation.

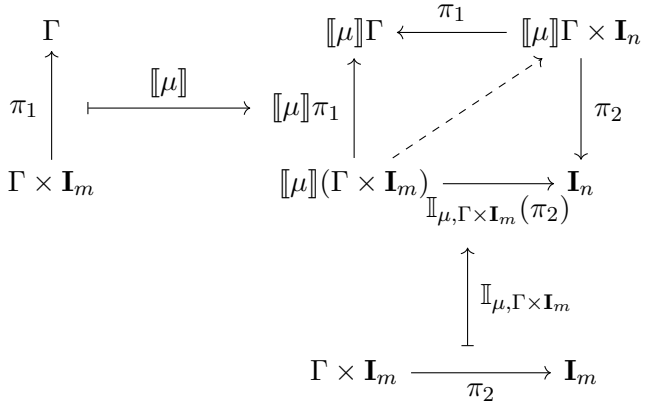
The type formers are the same as those in [GKNB21, Section 5.2] and [Awo18] except for identity types which we do not have and path types which will come later.

Definition B.3. A modal interval structure on a modal context structure consists of

- for each mode $m : \mathcal{M}$, a De Morgan algebra $\mathbf{I}_m : [\![m]\!]$,
- for any modes $m, n : \mathcal{M}$ and each modality $\mu : n \longrightarrow m$, a natural transformation of De Morgan algebras $\mathbb{I}_\mu : \mathbf{y}(\mathbf{I}_m) \longrightarrow [\![\mu]\!]^* \mathbf{y}(\mathbf{I}_n)$,

such that

- the preshaves $\mathbf{y}(\mathbf{I}_m)$ and morphisms $\mathbb{I}_\mu$ assemble into a lax natural transformation $[\![-]\!] \longrightarrow \mathbf{Set}^{\mathsf{op}} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$, where $\mathbf{Set}^{\mathsf{op}} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ is the functor constantly equal to $\mathbf{Set}^{\mathsf{op}}$,
- for each mode $m : \mathcal{M}$ and context $\Gamma : [\![m]\!]$, the product $\Gamma \times \mathbf{I}_m$ exists,
- for any modes $m, n : \mathcal{M}$, each modality $\mu : n \longrightarrow m$, and each context $\Gamma : [\![m]\!]$, the uniquely determined dashed arrow in the following diagram has an inverse:



Definition B.4. A modal face structure on a modal interval structure consists of

- a lax natural transformation $\mathbb{F} : [\![-]\!] \longrightarrow \mathbf{Set}^{\mathsf{op}} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$, where $\mathbf{Set}^{\mathsf{op}} : \mathcal{M}^{\mathsf{coop}} \longrightarrow \mathbf{Cat}$ is the functor constantly equal to $\mathbf{Set}^{\mathsf{op}}$,
- for each mode $m : \mathcal{M}$, a natural transformation $\mathsf{Eq}^0_m : \mathbf{y}(\mathbf{I}_m) \longrightarrow {\mathbb{F}_m}^{\mathsf{op}}$,

such that

- $\mathbb{F}$ factors through $\mathbf{BDisLat}^{\mathsf{op}}$, the functor constantly equal to the opposite of the category of bounded distributive lattices,
- for each mode $m : \mathcal{M}$, each context $\Gamma : [\![m]\!]$, and each interval term $r : \Gamma \longrightarrow \mathbf{I}_m$, it holds that $\mathbb{F}_{\mu,\Gamma}(\mathsf{Eq}^0_{m,\Gamma}(r)) = \mathsf{Eq}^0_{n,[\![\mu]\!](\Gamma)}(\mathbb{I}_{\mu,\Gamma}(r))$,
- for each mode $m : \mathcal{M}$ and each context $\Gamma : [\![m]\!]$, it holds that $\mathsf{Eq}^0_{m,\Gamma}(0) = \top$, where 0 is from $\mathbf{I}_m$ being a De Morgan algebra, and $\top$ is from $\mathbb{F}_m(\Gamma)$ being a bounded lattice,
- for each mode $m : \mathcal{M}$, each context $\Gamma : [\![m]\!]$, and each interval term $r : \Gamma \longrightarrow \mathbf{I}_m$, it holds that $\mathsf{Eq}^0_{m,\Gamma}(r) \wedge \mathsf{Eq}^0_{m,\Gamma}((1 - r)) = \bot$, where $(1 - r)$ is from $\mathbf{I}_m$ being a De Morgan algebra, and $\wedge$ and $\bot$ are from $\mathbb{F}_m(\Gamma)$ being a bounded lattice.

Definition B.5. A modal restriction structure on a modal face structure consists of

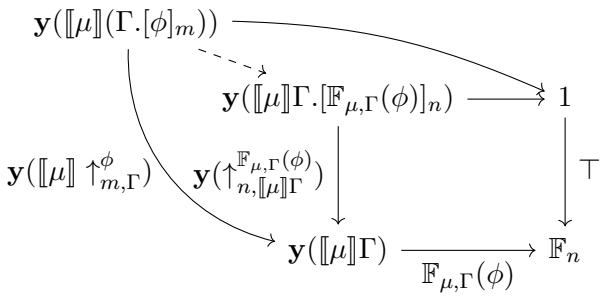
- for each mode $m : \mathcal{M}$, each context $\Gamma : [\![m]\!]$, and each face $\phi : \mathbf{y}(\Gamma) \longrightarrow \mathbb{F}_m$, a choice of pullback of the form:

$$\begin{array}{ccc} \mathbf{y}(\Gamma.[\phi]_m) & \longrightarrow & 1 \\ {\scriptstyle \mathbf{y}(\uparrow^{\phi}_{m,\Gamma})}\downarrow & & \downarrow{\scriptstyle \top} \\ \mathbf{y}(\Gamma) & \xrightarrow[\phi]{} & \mathbb{F}_m \end{array}$$

such that

- for all modes $m, n : \mathcal{M}$, each modality $\mu : n \longrightarrow m$, each context $\Gamma : [\![m]\!]$, and each face $\phi : \mathbf{y}(\Gamma) \longrightarrow \mathbb{F}_m$, the uniquely determined dashed arrow in the following diagram has an

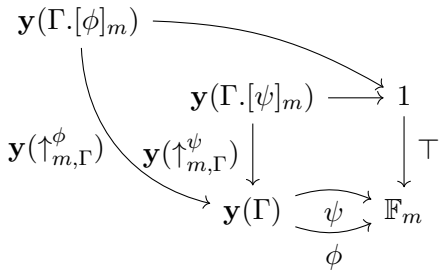
inverse:



Here, the commutativity of the outer square follows from the following calculation:

$$\begin{aligned}
\mathbb{F}_{\mu,\Gamma}(\phi) \circ \mathbf{y}([\![\mu]\!] \uparrow^{\phi}_{m,\Gamma}) &= \mathbb{F}_n([\![\mu]\!] \uparrow^{\phi}_{m,\Gamma})(\mathbb{F}_{\mu,\Gamma}(\phi)) \\
&= ((\mathbb{F}_n \circ [\![\mu]\!])(\uparrow^{\phi}_{m,\Gamma}) \circ \mathbb{F}_{\mu,\Gamma})(\phi) \\
&= (\mathbb{F}_{\mu,\Gamma.[\phi]_m} \circ \mathbb{F}_m(\uparrow^{\phi}_{m,\Gamma}))(\phi) \\
&= \mathbb{F}_{\mu,\Gamma.[\phi]_m}(\mathbf{y}(\uparrow^{\phi}_{m,\Gamma}) \circ \phi) \\
&= \mathbb{F}_{\mu,\Gamma.[\phi]_m}(\top) \\
&= \top.
\end{aligned}$$

Remark B.6. For each mode $m : \mathcal{M}$, each context $\Gamma : [\![m]\!]$, and any faces $\phi, \psi : \mathbf{y}(\Gamma) \longrightarrow \mathbb{F}_m$ with $\phi \leq \psi$, consider the following diagram:



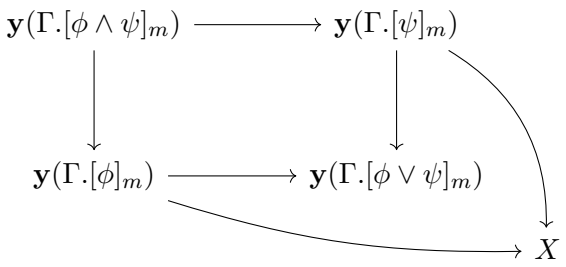
We can calculate

$$\begin{aligned}
\psi \circ \mathbf{y}(\uparrow^{\phi}_{m,\Gamma}) &= \mathbb{F}_m(\uparrow^{\phi}_{m,\Gamma})(\psi) \\
&\geq \mathbb{F}_m(\uparrow^{\phi}_{m,\Gamma})(\phi) \\
&= \phi \circ \mathbf{y}(\uparrow^{\phi}_{m,\Gamma}) \\
&= \top,
\end{aligned}$$

and thus $\psi \circ \mathbf{y}(\uparrow^{\phi}_{m,\Gamma}) = \top$, implying the outer square commutates, and we thus get a canonical morphism $\Gamma.[\phi]_m \longrightarrow \Gamma.[\psi]_m$.

Definition B.7. A modal face sturcture and a modal natural model (both on the same modal context sturcture) has systems if

- for each mode $m : \mathcal{M}$, each context $\Gamma : [\![m]\!]$, any faces $\phi, \psi : \mathbb{F}_m(\Gamma)$, each presheaf $X : \mathbf{PSh}([\![m]\!])$, and each commuting diagram



  where the arrows in the inner square are the canonical morphisms following from $\phi \wedge \psi \leq \phi$, $\phi \wedge \psi \leq \psi$, $\phi \leq \phi \vee \psi$, and $\psi \leq \phi \vee \psi$, if X is representable or $\mathbb{F}_m$ there exists at most one morphism $\mathbf{y}(\Gamma.[\phi \vee \psi]_m) \longrightarrow X$ such that the diagram commutes, and if X is $\mathcal{T}_m$ or $\widetilde{\mathcal{T}}_m$ there exists exactly one such morphism,
- for each mode $m : \mathcal{M}$, each context $\Gamma : [\![m]\!]$, and each presheaf $X : \mathbf{PSh}([\![m]\!])$, if X is representable, $\mathbb{F}_m$, or $\mathcal{T}_m$ there exists at most one morphism $\mathbf{y}(\Gamma.[\bot]_m) \longrightarrow X$, and if X is $\widetilde{\mathcal{T}}_m$ there exists exactly one such morphism.

Definition B.8. A path structure on modal interval structure and a modal natural model (both on the same modal context structure) is a direct translation of the rules for path types, and we will thus not give the details.

Definition B.9. A composition structure on modal restriction structure is a direct translation of the rules for composition, and we will thus not give the details.