

Contributions to Multimode and Presheaf Type Theory

Andreas Nuyts

Supervisors:
Prof. dr. ir. F. Piessens
Prof. dr. D. Devriese

Dissertation presented in partial
fulfillment of the requirements for the
degree of Doctor of Engineering
Science (PhD): Computer Science

August 4, 2020

Contributions to Multimode and Presheaf Type Theory

Andreas NUYTS

Examination committee:

Prof. dr. ir. B. Puers, chair

Prof. dr. ir. F. Piessens, supervisor

Prof. dr. D. Devriese, supervisor

Prof. dr. ir. B. Jacobs

Prof. dr. ir. T. Schrijvers

Prof. dr. D. R. Licata

(Wesleyan University, Middletown, CT, USA)


Prof. dr. L. Birkedal

(Aarhus University, Aarhus, Denmark)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

August 4, 2020

Uitgegeven in eigen beheer, Andreas Nuyts, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

This work is licensed under a Creative Commons "Attribution-NonCommercial-NoDerivatives 4.0 International" license.  You can even distribute this document on microfilm!

Contents

Contents	i
Acknowledgements	ix
Abstract (EN)	xi
Abstract (NL)	xiii
1 Introduction	1
1.1 Context	2
1.2 Higher Directed Type Theory in Practice	3
1.2.1 Example Problem	3
1.2.2 In Plain Dependent Type Theory	5
1.2.3 In Homotopy Type Theory	6
1.2.4 In Higher Directed Type Theory	7
1.3 ParamDTT: Parametric Quantifiers	8
1.4 RelDTT: Degrees of Relatedness	10
1.5 Transpension: The Right Adjoint to the Π -type	11
1.6 Robust Notions of Fibrancy	11
1.7 MTT: Well-Behaved Multimode Type Theory	13
1.8 Overview	13

I	Prerequisites	17
2	Mathematical Prerequisites	19
2.1	Set Theory	20
2.2	Category Theory	21
2.2.1	Categories, Functors and Natural Transformations	21
2.2.2	Mono- and Epimorphisms	22
2.2.3	Full and Faithful	22
2.2.4	Arrow Categories	23
2.2.5	Limits and Colimits	24
2.2.6	Ends and Co-ends	25
2.2.7	Dependent Ends and Co-ends	26
2.2.8	Slices and Elements	28
2.2.9	Adjunctions	29
2.2.10	Algebras and Coalgebras	31
2.2.11	Monads and Comonads	31
2.2.12	Subobject Classifier	36
2.2.13	Higher Category Theory	36
2.3	Presheaves	37
2.3.1	Definition and Yoneda-embedding	37
2.3.2	Examples of Base and Presheaf Categories	39
2.3.3	The Yoneda and Co-Yoneda Lemmas	43
2.3.4	Dependent Yoneda and Co-Yoneda Lemmas	44
2.3.5	Injective and Surjective Morphisms	45
2.3.6	Subobject Classifier	46
2.3.7	Base Pullbacks and Representable Morphisms	48
2.3.8	Lifting Functors	49
2.4	Factorization Systems	51
2.4.1	Introduction	51
2.4.2	Orthogonal Factorization Systems (OFSs)	52
2.4.3	Weak Factorization Systems (WFSs)	55
2.4.4	Functorial Weak Factorization Systems (FWFSs)	56
2.4.5	Natural Weak Factorization Systems (NWFSs)	58

2.4.6	Left Generation in Presheaf Categories	64
3	Formal Systems and Dependent Type Theory	71
3.1	Formal Systems, Syntax and Models	71
3.1.1	Motivation	71
3.1.2	Simple Algebraic Theories	73
3.1.3	Multisorted Algebraic Theories	74
3.1.4	Generalized Algebraic Theories (GATs)	75
3.1.5	Judgements and Typing Rules	77
3.2	Dependent Type Theory: Syntax and Models	78
3.2.1	Contexts and Substitutions	79
3.2.2	Structural Rules, Categories with Families (CwFs) and Natural Models	79
3.2.3	Universe Levels for Size Stratification	84
3.2.4	Type Formers	85
3.2.5	An Auxiliary Definition	95
3.3	The Curry-Howard Correspondence	95
3.3.1	Encoding Propositions as Types	96
3.3.2	Constructivity	97
4	Presheaf Models of Dependent Type Theory	99
4.1	Modelling Structural Rules: Presheaf Categories are CwFs . . .	99
4.1.1	Category of Contexts	100
4.1.2	Types Functor	100
4.1.3	Terms Functor	101
4.1.4	Empty Context and Context Extension	102
4.2	Universe Levels for Size Stratification	103
4.3	Type Formers	103
4.3.1	Π -types	103
4.3.2	Σ -types	105
4.3.3	Identity types	106
4.3.4	Universes	106
4.3.5	Other types	108

II	Contributions	109
5	Multimode Type Theory	111
5.1	Internalizing Transformations of Semantics	112
5.1.1	Internalizing Functors and CwF Morphisms	113
5.1.2	Internalizing Natural Transformations	115
5.1.3	Internalizing Adjunctions	119
5.2	Dependent Right Adjoints (DRAs)	122
5.2.1	Syntax and Semantics	123
5.2.2	Relation to Right Adjoints	124
5.3	Multimode Type Theory (MTT)	125
5.3.1	Introduction	127
5.3.2	The Syntax of MTT	130
5.3.3	Programming with Modalities	138
5.3.4	The Substitution Calculus of MTT	140
5.3.5	The Semantics of MTT	142
5.3.6	Applying MTT	145
5.3.7	Related Work	152
5.3.8	Conclusion	153
6	Presheaf Type Theory	155
6.1	Propositions: Syntax and Presheaf Semantics	157
6.1.1	Structural Rules	158
6.1.2	Universe of Propositions	159
6.1.3	Logical Operators	161
6.1.4	Equality Proposition	164
6.1.5	System Notation	165
6.1.6	Sketch of a Type Checking Algorithm	166
6.2	Types for Propositions	167
6.2.1	Proof Types	167
6.2.2	Extension Types	168
6.3	Presheaf Types: Syntax and Presheaf Semantics	169
6.3.1	Strictification	169

6.3.2	Glueing: The Final Slice Extension Operation	171
6.3.3	Pushout Type	172
6.3.4	Welding: The Initial Coslice Extension Operation	173
6.4	Presheaf Modalities	174
6.4.1	Right Adjoint Presheaf Functors	175
6.4.2	Central Liftings of Base Functors	176
6.5	Presheaf MTT	177
6.5.1	Extending the Type Checking Algorithm	178
7	Transpension: The Right Adjoint to the Π-type	181
7.1	Introduction and Related Work	182
7.1.1	The Power of Presheaves	182
7.1.2	Internalizing the Power of Presheaves	183
7.1.3	The Transpension Type	185
7.1.4	Contributions	186
7.2	A Naïve Transpension Type	188
7.3	A Snippet of MTT	191
7.4	A Mode Theory for Shape (Co)quantification	191
7.4.1	Shape Contexts	191
7.4.2	Mode Theory	192
7.4.3	Shapes and Multipliers	192
7.4.4	Modalities for Substitution	196
7.4.5	Modalities for (Co)quantification	197
7.5	An Informal Notation for Human Readers	198
7.6	Additional Typing Rules	203
7.7	Investigating the Transpension Type	203
7.8	Recovering Known Operators	206
7.9	Conclusion	211
8	Fibrancy	213
8.1	Introduction	213

8.1.1	Fibrant types	215
8.1.2	Stability of the Fibrant Replacement	217
8.1.3	Fibrancy of Π -types	221
8.2	Damped Factorization Systems	226
8.3	Stability	230
8.4	Robust NWFSs	231
8.4.1	Constructing Robust NWFSs	232
8.4.2	Robustness of Examples of NWFSs	233
8.4.3	Fibrancy of Π -types over a Robust NWFS	234
8.4.4	Stability of Robust NWFSs	235
8.5	Internalizing Stable NWFSs	237
8.5.1	Extensional Typing Rules for the Fibrant Replacement	237
8.5.2	Extensional Typing Rules for the Left Coreplacement	240
8.6	Internal Fibrancy via Eilenberg-Moore Algebras	244
8.6.1	Definition	245
8.6.2	Fibrant Replacement Elimination	245
8.6.3	Fibrancy of Type Formers	246
8.7	Internal Fibrancy, Directly	250
	Appendix 8.A Semantics of Extend after Multiplication	252
9	Degrees of Relatedness	257
9.1	Parametricity, from System F to DTT	258
9.1.1	System F	259
9.1.2	System $F\omega$	260
9.1.3	Dependent Type Theory	262
9.2	Parametric Quantifiers	265
9.2.1	Motivation	265
9.2.2	The Mode Theory and the Corresponding Instance of MTT	267
9.2.3	Extending the MTT instance to $\text{ParamDTT}_{\mathbf{a}}$	271
9.2.4	Wrapping up	273
9.3	Degrees of Relatedness	274

9.4	MTT as an Internal Language of the Model	278
9.5	Parametricity Features and their Requirements	279
10	Conclusion	283
10.1	Future Work	283
10.1.1	Towards Higher Directed Type Theory	283
10.1.2	Good Syntax	285
10.2	Implications in Practice	286
10.3	Is Information Presheavoidal?	287
	List of Publications	289
	List of Figures	291
	List of Mathematical Symbols	295
	Delimiters	295
	Arrows	296
	Judgements/Turnstiles	296
	Greek Letters	297
	Numbers	298
	Alphabetically	298
	Index	307
	Bibliography	319

Acknowledgements

This public acknowledgements section only contains acknowledgements that are primarily professional in nature.

Thanks to the members of the examination committee, as well as all well-intending anonymous reviewers for their time and effort in reading, questioning and thus improving my work.

Thanks to Atkey, Ghani, and Johann [AGJ14] and Bernardy, Coquand, and Moulin [BCM15] whom I do not really know, but I could dream of no more solid foundation to start my research from.

Thanks to my immediate superiors Dominique Devriese, Frank Piessens and Wouter Joosen and to DistriNet in general for a nonsense-free atmosphere of trust, respect, freedom, well-being, positivity, constructivity and confident ambition. Thanks to some key people who (used to) make DistriNet roll: Ghita, Annick, Katrien, Ann, Hilde.

Thanks also to the FWO and ultimately to the Belgian tax payers for the trust and the investment that made this work possible.

Many, many thanks to Dominique Devriese, who has been quite simply – en ik wik mijn woorden – the best supervisor I can imagine.

Many thanks also to Dan Licata. My stay in Middletown, right at the start of my PhD, and us talking directed type theory non-stop from breakfast till after dinner for several consecutive days really set things in motion for me.

Thanks to Jesper Cockx, my master thesis mentor and an exemplary scientist.

Thanks to Andrea Vezzosi, Daniel Gratzer, G.A. Kavvos and Lars Birkedal for the good international collaborations (Andrea also for the *many* discussions). Thanks to Koen Jacobs, Joris Ceulemans and Stelios Tsampas for the good collaborations at home.

Thanks to the people I met at conferences and research visits and with whom I had so many interesting discussions: Andreas Abel, Jean-Philippe Bernardy, Paolo Capriotti, Simon Huber, Nikolai Kraus, Magnus Baunsgaard Kristensen, Rasmus Ejlers Møgelberg, Max New, Gun Pinyo, Jason Reed, Christian Sattler, Sandro Stucki, Felix Wellen, Colin Zwanziger, and many others whom I disgracefully forget to mention, such as the people who explained to me in the Warsaw metro how simplicial sets work (or are they included in the list above?). Thanks to Amin Timany for similar discussions at home and for teaching me separation logic by repeated exposure.

Thanks to Raf, Lutgarde, Hilde, Dominique and Amin for the good teaching collaboration.

Thanks to Neline van Ginkel for mentoring this mathematician in the actual usage of a computer. This skill has proven invaluable.

I hold a PhD Fellowship from the Research Foundation - Flanders (FWO).

Abstract (EN)

In *modal* type theory, all functions and all variables are annotated with a *modality* describing the behaviour of the dependency. Applications include: modal logic (eponymously), variance of functors, parametricity and proof-irrelevance. The collection of modalities typically has the structure of an ordered monoid. Sometimes, the set of available modalities μ for functions $(\mu \vdash x : A) \rightarrow B$ depends on the types A and B . For example, when considering functions from \mathbb{N} to \mathbf{Bool} , we need not distinguish between parametric and non-parametric functions. Recently, Licata and Shulman have explained these phenomena by moving from an ordered monoid to a 2-category, whose objects are called *modes* and whose morphisms serve as modalities. In this case, we speak of *multimode* type theory. Here, one assigns a mode to every type, and the modality of a function must match the domain and codomain modes.

Presheaf models of dependent type theory have been successfully applied to model homotopy type theory (HoTT), parametricity, and directed and guarded type theory. There has been considerable interest in internalizing aspects of these presheaf models, either to make the resulting language more expressive (e.g. by providing internal parametricity operators for proving free theorems), or in order to carry out further reasoning internally, i.e. within type theory, allowing greater abstraction and sometimes automated verification.

This thesis makes a number of contributions in multimode and presheaf type theory, motivated by the preparation of the development of a higher-dimensional directed dependent type system with interacting modalities for functoriality and naturality. In such a type system, naturality of a construction could be asserted by construction, i.e. by non-violation, rather than by a tedious naturality proof.

In joint work with Gratzler, Kavvos and Birkedal, we developed a multimode type system MTT which is parametrized by an arbitrary external 2-category called the *mode theory*. We discuss semantics and demonstrate the good syntactic properties of this type system by being able to prove a canonicity result.

We give an overview of some base-category-agnostic presheaf operators including the *Glue*-type from cubical type theory, its dual *Weld* and the strictness axiom. We sketch a type-checking algorithm that is not specialized to some cubical system and that takes into account proposition variables.

While the constructions of presheaf models largely follow a common pattern, approaches towards internalization do not. Throughout the literature, various internal presheaf operators can be found and little is known about their relative expressivity. Moreover, some of these require that variables whose type is a shape (representable presheaf) be used affinely. We propose a novel type former, the transpension type, which is right adjoint to universal quantification over a shape. We give general typing rules and a presheaf semantics in terms of base category functors dubbed multipliers. We demonstrate how the transpension type and the strictness axiom can be combined to implement all and improve some of the internal presheaf operators that we are aware of.

Many presheaf models of type theory do not take into account all types of the general presheaf model, but only fibrant types. Notions of fibrancy types include: discrete types (for parametricity), Kan types (for HoTT), Segal and covariant types (for directed type theory) and clock-irrelevant types (for guarded type theory). For reasons already mentioned, it is worthwhile to try and internalize as much of the reasoning about fibrancy as possible. A central concept in internalizing fibrancy is the fibrant replacement monad acting on types. We introduce the *robustness* criterion for notions of fibrancy, which asserts that the fibrant replacement is stable under substitution, a necessity for internalization. As a bonus, it also asserts fibrancy of the Π -type if its codomain is fibrant. We demonstrate how Kan and Segal fibrancy fail to be robust, and how this can be remedied by considering *contextual* fibrancy. We introduce the notion of *damped* natural weak factorization systems as a categorical foundation for contextual fibrancy. We axiomatize an internal fibrant replacement operation and show how it can be used to define and reason about fibrancy internally. In a few examples, we also discuss how fibrancy can sometimes be defined more directly internally, sometimes using the transpension type.

Prior to most work in this thesis, we had contributed two type systems ParamDTT (with Vezzosi and Devriese) and RelDTT. The former features modalities for parametricity and ad hoc polymorphism, and the latter extends that to a more general mode theory that provides also irrelevance, shape-irrelevance and a novel *structural* modality. These type systems used *Glue* and *Weld* as internal parametricity operators. In this thesis, we give a high-level discussion of these systems and their semantics and explain how they are (almost) an instance of MTT. We also explain how the robustness criterion and the internalization of fibrancy apply to the notion of discreteness that we use to validate Reynolds' identity extension lemma.

Abstract (NL)

In *modale* typetheorie worden alle functies en alle variabelen geannoteerd met een *modaliteit* die het gedrag van de afhankelijkheid beschrijft. Enkele toepassingen zijn: modale logica (vandaar de naam), variantie van functoren, parametriciteit en bewijsirrelevantie. De verzameling van modaliteiten heeft doorgaans de structuur van een geordende monoïde. Soms hangt het af van de types A en B welke modaliteiten μ er ter beschikking zijn voor functies $(\mu \vdash x : A) \rightarrow B$. Voor functies van \mathbb{N} naar \mathbf{Bool} hoeven we bv. geen onderscheid te maken tussen parametrische en niet-parametrische functies. Recent hebben Licata en Shulman dit fenomeen verklaard door van een geordende monoïde over te gaan op een 2-categorie, waarvan men de objecten *modi* noemt en waarvan de morfismen dienen als modaliteiten. In dit geval spreken we van multimodustypetheorie. Daar kent men aan elk type een modus toe en moet de modaliteit van een functie overeenkomen met de modi van domein en codomein.

*Preschoof*modellen van dependent type theory worden met succes gebruikt om homotopietypetheorie (HoTT), parametriciteit en directed en guarded type theory te modelleren. Er gaat de laatste tijd veel aandacht naar het internaliseren van aspecten van deze preschoofmodellen, hetzij om de resulterende taal expressiever te maken (bv. door interne parametriciteitsoperatoren aan te bieden om free theorems te bewijzen), hetzij om verdere argumentatie intern in typetheorie uit te voeren, wat een grotere abstractie en soms automatische verificatie toelaat.

Deze thesis levert een aantal bijdragen aan multimodus- en preschooftypetheorie met het oog op de ontwikkeling van een hoger-dimensionaal directed dependent typesysteem met interagerende modaliteiten voor functorialiteit en natuurlijkheid. In een dergelijk typesysteem zou natuurlijkheid van een constructie kunnen blijken per constructie, uit het niet-schenden ervan, in plaats van uit langdradige natuurlijkheidsbewijzen.

In samenwerking met Gratzer, Kavvos en Birkedal hebben we een multimodus-typesysteem MTT ontwikkeld met als parameter een externe 2-categorie naar keuze, genaamd de *modustheorie*. We bespreken de semantiek en demonstreren de goede syntactische eigenschappen van dit systeem in de zin dat we een canoniciteitsresultaat kunnen bewijzen.

We geven een overzicht van enkele basiscategorie-agnostische preschoofoperatoren waaronder het Glue-type uit cubical type theory, zijn duale Weld en het strictness-axioma. We schetsen een typechecking-algoritme dat niet is toegespitst op één of ander cubical systeem en dat propositievariabelen toelaat.

Hoewel constructies van preschoofmodellen grotendeels éénzelfde patroon volgen, geldt dit niet voor methoden om ze te internaliseren. In de literatuur vinden we allerlei preschoofoperatoren en er is weinig gekend over hun relatieve expressiviteit. Bovendien vereisen sommige dat variabelen met als type een shape (een representeerbare preschoof) affien gebruikt worden. Wij stellen een nieuwe typeconstructor voor, het transpensiotype, dat rechts adjunct is aan universele kwantificatie over een shape. We geven algemene typeringsregels en een preschoofsemantiek in termen van functoren tussen basiscategorieën die we multiplicatoren noemen. We demonstreren hoe het transpensiotype en het strictness-axioma samen gebruikt kunnen worden om alle interne preschoofoperatoren waarvan we weet hebben, te implementeren, en enkele te verbeteren.

Veel preschoofmodellen van typetheorie beschouwen niet alle types in het algemene preschoofmodel, maar beperken zich tot fibrante types. Enkele voorbeelden zijn: discrete types (voor parametriciteit), Kan types (voor HoTT), Segal en covariante types (voor directed type theory) en klok-irrelevante types (voor guarded type theory). O.w.v. eerder vermelde redenen loont het de moeite om zoveel mogelijk van de argumentatie over fibrantie te internaliseren. Een centraal concept hierbij is de fibrante-vervangingsmonade op types. We introduceren het *robuustheids*criterium voor noties van fibrantie, dat impliceert dat de fibrante vervanging stabiel is onder substitutie, een vereiste om ze te internaliseren. Ditzelfde criterium impliceert bovendien dat het Π -type fibrant is van zodra het codomein fibrant is. We tonen waarom Kan- en Segal-fibrantie niet robuust zijn en hoe dit verholpen kan worden door in de plaats *contextuele* fibrantie te beschouwen. We axiomatiseren een interne fibrante-vervangingsoperatie en tonen hoe deze kan dienen om fibrantie intern te definiëren en te gebruiken. A.h.v. enkele voorbeelden bespreken we ook hoe fibrantie soms rechtstreeks intern kan gedefinieerd worden, al dan niet m.b.v. het transpensiotype.

Voorafgaand aan het meeste werk in deze thesis hebben we twee typesystemen ontwikkeld: ParamDTT (met Vezzosi en Devriese) en RelDTT. Het eerste biedt

modaliteiten voor parametriciteit en adhocpolymorfisme, en het tweede breidt dit uit tot een algemenere modustheorie die ook irrelevantie, shape-irrelevantie en het nieuwe concept *structuraliteit* omvat. Deze typesystemen gebruiken *Glue* en *Weld* als interne parametriciteitsoperatoren. In deze thesis bespreken we de grote lijnen van deze systemen en hun semantiek en leggen we uit hoe ze (bijna) een instantie van MTT zijn. We tonen ook hoe robuustheid en de internalisering van fibrantie van toepassing zijn op de notie van discreetheid die we gebruiken om Reynolds' identity extension lemma te doen gelden.

Chapter 1

Introduction

Parts of this introduction were taken from a non-public grant renewal application at the Research Foundation - Flanders (FWO), authored by myself in collaboration with Dominique Devriese.

The goal we originally set out for this PhD project was to establish higher-dimensional directed dependent type theories (higher DDTT) by formulating them, implementing them, proving their consistency and demonstrating their use. By a DDTT, we mean a dependent type theory which has not only a built-in notion of equality, but also of transformation. DDTTs would provide a powerful framework for computer-assisted reasoning about asymmetric phenomena such as subtyping, syntactic substitution and various kinds of non-invertible transformations. In particular, we expect to obtain functoriality-for-free, relieving programmers from the burden of explicitly implementing the ‘map’ operation of functors. Moreover, we aim to generalize the notion of parametricity in programming languages to one of naturality, interacting smoothly with the aforementioned functoriality. This would first of all extend the concept of naturality beyond natural transformations but also relieve mathematicians from the burden to prove that their operations are natural by giving them a method that asserts naturality by construction. On top of that, the intention was for naturality theorems to be provable *within* dependent type theory.

Unsurprisingly for those who tried, this turned out to be a project for more than a single PhD. None of the original goals has been fully achieved, but I believe that with this dissertation and its associated papers and technical reports, my co-authors and I make several important contributions in mapping out the road and freeing it from some important obstructions, and each of these contributions has collateral virtues.

In this introduction, I first provide some context (section 1.1) and provide an example to illustrate the use of the (intended) higher directed type theory (section 1.2). Next, I discuss the rationale behind this PhD project, the research activities that were undertaken and in what form they can be found in this thesis (sections 1.3 to 1.7). This is followed by an overview of the thesis, summarizing the contributions (section 1.8).

1.1 Context

Static type systems provide a way of guaranteeing safety and termination of computer programs, by preventing at compile-time that variables are assigned inappropriate values. They are also interesting from a logical perspective, as the Curry-Howard correspondence associates type operators to logical operators, so that propositions may be translated into types and vice versa [How80] (section 3.3). Proving a proposition then corresponds to constructing an element of the associated type.

Dependent type theories such as Martin-Löf type theory (MLTT) [Mar82; Mar98] include type theoretic counterparts for universal (\forall) and existential (\exists) quantification in logic. As such, all possible propositions can be translated into dependent types, and they can be proven formally by constructing a program of the corresponding type and having it type-checked by a computer. This crucial observation allows the use of dependently typed functional programming languages as proof assistants for proving either mathematical theorems or program correctness. Examples are Coq [Coq14], Agda [Nor09] and Idris [Bra13].

An important ingredient of logical reasoning is the concept of **equality**. This, too, has a type theoretic counterpart in MLTT: given values $a, b : A$, we have an ‘identity type’ $a \equiv_A b$ of proofs that a and b are equal. The precise definition of the identity type is subtle, but we have three operators at hand to ensure that equality is an equivalence relation on the elements of A : a reflexivity constructor ensuring that everything is equal to itself, a proof-composition operator $(x \equiv_A y) \rightarrow (y \equiv_A z) \rightarrow (x \equiv_A z)$ ensuring transitivity, and an inversion operator ensuring symmetry. As such, (classical) MLTT equips every type with a built-in equivalence relation called ‘(propositional) equality’.

Homotopy Type Theory (HoTT [Uni13]) starts from the observation that two objects can be identified in multiple ways. For example, a boolean is essentially the same thing as a bit: it takes one out of two values. But if I want to encode a boolean as a bit, do I encode `true` as 1 or as 0? I have to pick one of two ways in which a boolean is the same as a bit. While MLTT is usually

equipped with a ‘uniqueness of identity proofs’ axiom (UIP) [Uni13, §7.2], which states that all elements (proofs) of $a \equiv_A b$ are equal and hence that the only information encoded in an equality proof, is its existence; HoTT abandons this principle and interprets the type $a \equiv_A b$ as the type of all isomorphisms between a and b . This is formalized by Voevodsky’s **univalence axiom** [KLV12; Uni13], which essentially states that isomorphic types are propositionally equal.

If MLTT is a type theory with good support for notions of equality and HoTT is one with good support for notions of isomorphism, then **directed type theories** [LH11; Nuy15; RS17; Nor19; WL20] are aimed at supporting various asymmetric phenomena, such as a subtyping relation on types [Abe08], syntactic substitutions in programming language formalization [LH11], transformations between mathematical structures such as vector spaces or monads, functorial behaviour etc.

The key idea is to abandon the symmetry of the equality relation, leading to a type $a \leq_A b$ of inequality proofs. We can do this in the spirit of classical MLTT, with a ‘uniqueness of inequality proofs’ axiom, and obtain a theory that automatically equips any type with a kind of order relation. Or we can do this in the spirit of HoTT, interpreting $a \leq_A b$ as the type of transformations from a to b . In that case, we are equipping types with a much richer structure of transformations; what is mathematically called a (higher) category. An important aspect to consider in directed type theories is variance: dependencies can be increasing/covariant, decreasing/contravariant, or can disrespect inequality altogether. Modalities are annotations on function types and can be used to keep track of the variance of functions.

1.2 Higher Directed Type Theory in Practice

Although no type theory for higher¹ categories exists yet, we will demonstrate with an example what we hope to achieve with such a system.

1.2.1 Example Problem

In purely functional programming languages like Haskell, functions behave the way they do in mathematics: calling the same function with the same inputs

¹In the sense of (n, r) -categories. Type systems for $(\infty, 1)$ -categories do exist [RS17; WL20].

multiple times, will yield the same output, and no side effects occur in the process.²

If we do want to allow a function to cause side-effects, then we can give it a monadic return type [Mog89]. For example, if we want a function with output type A to be able to log messages of type W , then we give it output type $\text{Writer } W \ A := W \times A$. The type W should have the structure of a monoid, with $e : W$ and an associative binary operation $* : W \times W \rightarrow W$ for concatenating messages, and a unit element $e : W$ that serves as the empty message. Then the functor $\text{Writer } W$ is a monad (definition 2.2.46), whose unit (a.k.a. return) function $\eta : A \rightarrow \text{Writer } W \ A : a \mapsto (e, a)$ creates programs making no use of the logging functionality in the sense that they return the empty message, and whose bind operation (proposition 2.2.47) concatenates programs' messages:

$$\text{bind} : (A \rightarrow \text{Writer } W \ B) \rightarrow (\text{Writer } W \ A \rightarrow \text{Writer } W \ B)$$

$$\text{bind } f (w, a) = \text{let } ((w', b) = f a) \text{ in } (w * w', b).$$

If we want to *add* logging functionality to an existing monad M , we use the monad transformer $\text{WriterT } W$, where $\text{WriterT } W \ M \ A := M(W \times A)$. For example M could be the **Maybe** monad, where elements of **Maybe** A are either **nothing** or **just** a where $a : A$ (i.e. $\text{Maybe } A \cong A \uplus \text{Unit}$). A function of output type **Maybe** A is conceptually a function of output type A that has the option to fail. Then a function of output type $\text{WriterT } W \ \text{Maybe } A := \text{Maybe}(W \times A)$ is a function that has the option to fail and, if it doesn't, will yield an output of type A and may log messages of type W .

A monad morphism is nothing but a natural transformation that respects the unit and bind operations. In Moggi's framework of monadic side effects, a monad morphism $m : M_0 \rightarrow M_1$ can be thought of as a compiler that compiles the effectful operations available in M_0 to effects in M_1 .

Recall that the free monoid over S is given by $(\text{List } S, [], ++)$. Thus, an arbitrary function $s : S_0 \rightarrow S_1$ gives rise to a monoid morphism $\text{List } s : (\text{List } S_0, [], ++) \rightarrow (\text{List } S_1, [], ++)$, which in turn gives rise to a monad morphism $\text{WriterT } (\text{List } s) \ M : \text{WriterT } (\text{List } S_0) \ M \rightarrow \text{WriterT } (\text{List } S_1) \ M$ for any monad M . On the other hand, a monad morphism $m : M_0 \rightarrow M_1$ should give rise to a monad morphism $\text{WriterT } W \ m : \text{WriterT } W \ M_0 \rightarrow \text{WriterT } W \ M_1$ for any monoid W .

²Functions in Haskell, unlike functions as usually understood in mathematics, may be non-terminating. Moreover, functions in Haskell do have the side effect of consuming time, CPU and memory; this side effect is disregarded.

The challenge is now to construct the aforementioned monad morphisms and to prove commutativity of the following diagram:

$$\begin{array}{ccc}
 \text{WriterT (List } S_0) M_0 & \xrightarrow{\text{WriterT (List } S_0) m} & \text{WriterT (List } S_0) M_1 \\
 \text{WriterT (List } s) M_0 \downarrow & & \downarrow \text{WriterT (List } s) M_1 \\
 \text{WriterT (List } S_1) M_0 & \xrightarrow{\text{WriterT (List } S_1) m} & \text{WriterT (List } S_1) M_1
 \end{array}$$

If $s = \text{capitalize} : \text{String} \rightarrow \text{String}$ and $m = \text{just} : \text{Id} \rightarrow \text{Maybe}$, then what this says is that it doesn't matter whether we first capitalize all logged messages and then decide to allow but not use the option to fail, or the other way around.

Existence of the functions underlying the monad morphisms may be needed already when defining a program. The fact that the functions are monad morphisms and that the diagram commutes, may be necessary to prove correctness of a program. However, since all of these results are completely obvious and dull from a categorical viewpoint, we want to spend minimal time on implementing and proving them, and in particular we prefer not to bother with list induction. This is why we want native support for category theory in our programming language.

1.2.2 In Plain Dependent Type Theory

If we want face the challenge in plain dependent type theory in a somewhat principled manner, we would do the following:

- Show that `List` is a functor from types to monoids:
 - For every $f : A \rightarrow B$, show that there is a monoid morphism $\text{List } f : \text{List } A \rightarrow \text{List } B$:
 - * Define $\text{List } f$, by list recursion,
 - * Show that it respects the empty list (trivial) and list concatenation (by list induction).
 - Show that this operation respects identity and composition,³ by list induction.
- Show that `WriterT` is a functor from monoids to covariant monad transformers:
 - Show that $\text{WriterT } W$ is a covariant monad transformer for every monoid W :

³This is not actually needed for the challenge at hand, but is a matter of not doing half work.

- * Show that $\text{WriterT } W \ M$ is a monad for every monad M .
- * Show that, for any monad morphism $m : M_0 \rightarrow M_1$, we get a monad morphism $\text{WriterT } W \ m : \text{WriterT } W \ M_0 \rightarrow \text{WriterT } W \ M_1$:
 - Define $\text{WriterT } W \ m \ A : \text{WriterT } W \ M_0 \ A \rightarrow \text{WriterT } W \ M_1 \ A$ for any type A ,
 - Show that it is natural (remark 2.2.7) in A ,
 - Show that it respects the monad operations.
- * Show that this operation respects identity and composition.
- For any monoid morphism $w : W_0 \rightarrow W_1$, show that there is a morphism of covariant monad transformers $\text{WriterT } w : \text{WriterT } W_0 \rightarrow \text{WriterT } W_1$:
 - * Define $\text{WriterT } w \ M \ A : \text{WriterT } W_0 \ M \ A \rightarrow \text{WriterT } W_1 \ M \ A$ for any monad M and type A ,
 - * Show that it is natural in M ,
 - * Show that it is natural in A ,
 - * Show that it respects the monad transformer operation lift .⁴
- Show that this operation respects identity and composition.

1.2.3 In Homotopy Type Theory

Let us see how this simplifies in homotopy type theory (HoTT) [Uni13]. Of course, HoTT only has native support for isomorphisms, so we will assume that s and m are isomorphisms. We will also assume that Haskell types are sets (in the HoTT sense) and thus that kinds are 1-groupoids.

We then have to do the following:

- Show that `List` is a groupoid functor from types to monoids.
 - For every $f : A \cong B$, the univalence axiom provides an equality proof (a.k.a. path) $\text{ua } f : A \equiv B$. The function $\lambda X. (\text{List } X, [], ++, _)$ sending the type X to the free monoid⁵ over X respects equality, so we get a proof of $(\text{List } A, [], ++, _) \equiv_{\text{Monoid}} (\text{List } B, [], ++, _)$, which by the structure identity principle (SIP) [Uni13, §9.8] is the same as a monoid isomorphism.
 - This operation automatically respects identity and composition, because all HoTT functions respect identity and composition of paths.

⁴<https://hackage.haskell.org/package/transformers-0.5.6.2/docs/Control-Monad-Trans-Class.html>

⁵The underscore stands for the proofs of the monad laws.

So it turns out that we can prove this without knowing the implementation of `List` and with little knowledge of the definition of a monoid (we merely need to know that it is a ‘standard notion of structure’). We get this result essentially for free.

- Show that `WriterT` is a groupoid functor from monoids to groupoid-functorial monad transformers:
 - Show that `WriterT W` is a groupoid-functorial monad transformer for every monoid W :
 - * Show that `WriterT W M` is a monad for every monad M .
 - * Again, we get groupoid functoriality for free. Indeed, given a monad isomorphism $m : M_0 \cong M_1$, we get $M_0 \equiv_{\text{Monad}} M_1$ by the SIP, whence a proof of `WriterT W M0 \equiv_{Monad} WriterT W M1`, which by the SIP is the same as an isomorphism between the writer monads.
 - By similar reasoning, groupoid functoriality of `WriterT` is also for free.

If we use book HoTT [Uni13] to do the above, then we get functions that do not compute but instead block on the univalence axiom. However, since the arrival of cubical HoTT [Coh+17], the univalence axiom has computational content and we can actually apply `List s : List S0 \rightarrow List S1` to a concrete list and get an output.

1.2.4 In Higher Directed Type Theory

In the previous subsection, we saw that we could greatly shorten our todo list by moving to HoTT, and moreover that we are rid of all list inductions. The price we paid and which we seek to unpay by moving to higher DDTT, is that we had to assume that m and s are isomorphisms.

In higher DDTT, we expect that our todo list will look like this:

- Let the type-checker check that `List` is covariant (i.e. can be annotated with the covariance modality). In fact, let it check that the function $\lambda X.(\text{List } X, [], ++, _)$ sending the type X to the free monoid over X , is covariant. This requires that monoids depend on their structure by a special modality: a directed analogue of the structural modality which we introduced in Degrees of Relatedness [ND18a] (chapter 9).
- Show that `List` is a functor from types to monoids.
 - For every $f : A \rightarrow B$, the directed univalence axiom [WL20] provides an inequality proof (a.k.a. morphism or directed path) `dua f : A \leq`

B . By covariance, we get a proof of $(\text{List } A, [], ++, _)\leq_{\text{Monoid}}(\text{List } B, [], ++, _)$, which by an expected directed SIP is the same as a monoid morphism.

- This operation automatically respects identity and composition, because all covariant functions respect identity and composition of morphisms.

Again, we get this result essentially for free.

- Show that `WriterT` is a functor from monoids to covariant monad transformers:
 - Show that `WriterT W` is a covariant monad transformer for every monoid W :
 - * Show that `WriterT W M` is a monad for every monad M .
 - * Let the type-checker check that `WriterT W M` satisfies the covariance modality w.r.t. M .
 - * Again, we get the covariant action and laws for free. Indeed, given a monad isomorphism $m : M_0 \cong M_1$, we get $M_0 \leq_{\text{Monad}} M_1$ by the directed SIP, whence a proof of `WriterT W M_0` \leq_{Monad} `WriterT W M_1`, which by the directed SIP is the same as a morphism between the writer monads.
 - By similar reasoning, functoriality of `WriterT` is also for free.

Thus, we expect that higher DDTT can drastically simplify proofs of boring properties where HoTT can already do so for isomorphisms. This generalization is necessary because most transformations are not invertible. It is also complex, because while in HoTT all functions respect equality/isomorphism, in higher DDTT it is not reasonable to expect that all functions respect inequality/morphisms. Therefore, we need to keep track of the behaviour of functions in order to assert covariance, contravariance, naturality etc. of functions *by construction*, in a way that can (hopefully) be verified by a type-checker. This thesis is not concerned with the variance *checking* aspect, but with paving the road towards the design of a sound system in the first place.

1.3 ParamDDTT: Parametric Quantifiers

In my master thesis [Nuy15], I studied higher DDTT from a purely type theoretic point of view, trying to set up a system of typing rules that appeal to category-theoretic intuition and do not obviously introduce contradictions. At the start of my PhD, I wanted to underpin the work of my master thesis with a denotational semantics. The most natural setting to formulate these semantics seemed to be (higher) category theory, but attempts to model the style of directed type theory

from my master thesis in this setting kept failing.⁶ For the category theorist, the problem can be succinctly described by saying that the functor category functor $\text{Cat}^{\text{op}} \times \text{Cat} \rightarrow \text{Cat} : (\mathcal{C}, \mathcal{D}) \mapsto \mathcal{D}^{\mathcal{C}}$ does not preserve composition of profunctors⁷, failing the interpretation of the function type. For the type theorist, this problem is the reason that Reynolds' relationally parametric interpretation of System F [Rey83] features an identity extension lemma but no composition extension lemma as it would be violated by the function type, and that later models of parametricity [AM13; AGJ14] are formulated in reflexive graphs, which are exactly categories *without composition*.

The second most natural setting to work in, are presheaf categories. It is known [Hof97] (chapter 4) that every presheaf category is a model of dependent type theory. Presheaf categories appear naturally as models of parametricity in dependent type theory [AGJ14; BCM15; Mou16] but are also used as models of higher category theory (e.g. via the notion of quasi-categories [nLa20g]) and homotopy type theory [Coh+17; Hub16; BCH14; KLV12].

The idea arose in discussion with Andrea Vezzosi and Andreas Abel to model the undirected part of the intended DDTT, i.e. a type system with a naturality modality, but no modalities for functoriality. Naturality is then more typically called parametricity. This simplification of the ideas in my master thesis lead to a dependently typed system ParamDDT in which function types could be annotated as parametric or non-parametric, i.e. a system featuring a parametric \forall alongside the non-parametric Π [NVD17a] (section 9.2). The idea that functions of a given domain and codomain may or may not be parametric, and hence that we should keep track of whether they are if we want to rely on free theorems [Wad89], turned out to be an answer to an open question in the literature. Indeed, parametricity results about simpler type systems such as System F (the polymorphic λ -calculus) and System F ω had not been properly carried over to dependent type theory where large types are involved.

The original paper on ParamDDT [NVD17a] is not subsumed in this thesis, but in **section 9.2** of the chapter on Degrees of Relatedness, we give a **high-level discussion** of the system and its model, and explain how the system can be constructed more cleanly and efficiently with the tools that are available today.

The parametricity modality is modelled as a CwF morphism [Dyb96], which prompted a study of the **internalization of CwF morphisms** into type theory [Nuy17], which is found in **section 5.1** of the chapter on multimode type theory.

⁶Licata and Harper [LH11] do provide a model in category theory, but this work has a coupling of variance of types and terms (covariant terms live in covariant types) that we seek to relax.

⁷it does laxly, but this can be broken by exponentiating again (example 8.1.27)

Following Bernardy, Coquand, and Moulin [BCM15] and Moulin [Mou16] (henceforth: Moulin et al.), ParamDTT was modelled in cubical sets whose edges we annotated as expressing either equality (paths) or relatedness (bridges) [Nuy17]. Following Atkey, Ghani, and Johann [AGJ14], Reynolds’ identity extension lemma [Rey83] was modelled by restricting to *discrete* types. **Well-behavedness of discreteness** was originally proven ad hoc [Nuy17], but now follows more straightforwardly from the results on robust notions of fibrancy in **chapter 8** on fibrancy, see further.

We also wanted free parametricity theorems to be provable internally in ParamDTT. We could not rely on the **internal parametricity operators** by Moulin et al. [BCM15; Mou16], because these require an *affine* cubical model whereas discreteness of bridge and path types requires a *cartesian* cubical model. Instead, we used **the Glue type** from cubical HoTT [Coh+17] (stripped of its Kan fibrancy requirements) and introduced **a dual type Weld**. We showed that these types can be modelled in arbitrary presheaf models [Nuy17] and discuss them in **chapter 6** on presheaf type theory. We refer to the paper [NVD17a] for examples on how to apply these operators.

1.4 RelDTT: Degrees of Relatedness

In follow-up work, we abandon the idea that types and kinds should be the same thing, inspired via directed type theory by the fact that in category theory, the collection of n -categories is of course an n -category but, much more interestingly, is an $(n + 1)$ -category. This solves technical inconveniences in ParamDTT, such as the fact that small types contain unnecessary relational structure, whereas universes seemed to lack structure.

Following [LS16], we move from a modal type theory where function types are annotated by a modality (e.g. parametric or not) to a multimode type theory. In a multimode type theory, every judgement is annotated by a mode, which is of course just a syntactic feature but conceptually tells you in what category the judgement should be interpreted. Modalities then have a domain and a codomain and are modelled by CwF morphisms between the corresponding categories.

Concretely, we move to a setting where types are modelled as cubical sets whose edges are annotated with a degree of relatedness, generalizing the notion of bridges and paths from ParamDTT. The mode of a judgement then says how many degrees of relatedness are available in the type of that judgement. This way, universes can have more structure than the types they contain, whereas small types can be rid of meaningless relational structure.

We exhibit parametricity as just one out of many interesting and less interesting modalities, including ad hoc polymorphism, irrelevance (at type-checking type) [Pfe01; MS08; BB08; AS12], shape-irrelevance [AVW17], as well as a novel *structural* modality which explains how algebras (living in a kind) depend on their structure (a lower-level object living in a type).

The original paper on this type system RelDTT [ND18a] is not subsumed in this thesis, but again a high-level discussion that also relates it to today’s state of the art is given in a **dedicated chapter 9**. Other aspects of the system are handled the same way as for ParamDTT, see the previous section.

1.5 Transpension: The Right Adjoint to the Π -type

The observation by Dominique Devriese that it seemed impossible to prove parametricity of System F in ParamDTT, sparked an investigation of the comparative expressivity of internal parametricity operators [ND18b]. The crux, it turned out, is that the operators by Moulin et al. [BCM15; Mou16] do something that *Glue* and *Weld* do not: promote cells of a cubical set to a higher dimension. Of course abstraction over a dimension does the opposite, e.g. a square in $\Pi(i : \mathbb{I}).A$ where i ranges over the interval (i.e. a line), is a cube in A . In **chapter 7** and its associated technical report [Nuy20], we introduce the *transpension type former* $\check{\times} i.A$ which is right adjoint to the function type, and we explain how this operation can be used to derive Moulin et al.’s internal parametricity operators, as well as other internal presheaf operators such as the amazing right adjoint \surd [Lic+18].

The semantics of the transpension type and its associated operators are parametrized by an almost arbitrary functor which we call a multiplier and which interprets context extension with a *shape variable* $u : \mathbb{U}$. We introduce a series of criteria (including ‘affine’ and ‘cartesian’) for classifying multipliers and deduce internal properties depending on those criteria.

1.6 Robust Notions of Fibrancy

Many type systems modelled in presheaf categories interpret the type judgement not in the standard way (section 4.1.2), but have to restrict to a subset of all presheaf types. As mentioned, in order to validate Reynolds’ identity extension lemma, in models of parametricity we need to restrict to *discrete* types [AGJ14; NVD17a; ND18a; CH19]. In models of HoTT [e.g. KLV12; Coh+17], one restricts to *Kan fibrant* types, which are types equipped with

appropriate composition operations for (higher) paths. In presheaf models of directed type theory [RS17; WL20], one is interested in *Segal fibrant* types (with composition operations for (higher) morphisms), covariant types (essentially Haskell’s functors), and other notions.⁸ In models of guarded type theory [BM18], one restricts to *clock-irrelevant* types.

All of these conditions are notions of *fibrancy*, which means that the types satisfying the condition are those types $\Gamma \vdash T$ type such that the weakening substitution $(\Gamma, x : T) \rightarrow \Gamma$ is a *fibration*, i.e. belongs to the right class of a *factorization system* on the presheaf CwF.

Because the most obvious interpretation for the parametric quantifier \exists in ParamDTT does not automatically preserve discreteness, we instead have to use its ‘discrete replacement’ to actively force it to be discrete. Hence, we want this discrete replacement operation to be stable under substitution. The technical report on ParamDTT [Nuy17] contains a proof that this is the case, which was an unpleasant experience to write and is an unpleasant experience to read. When moving to RelDTT, I was less than excited about the prospect of having to redo it, especially with the long-term goal of higher DDTT in mind.

So I developed a more principled approach and formulated the *robustness* criterion [Nuy18b] (section 8.4). A notion of fibrancy that is generated in a robust way, as is the case for discreteness (example 8.4.7), automatically comes with a fibrant replacement operation that is stable under substitution.

Excitingly, robust notions of fibrancy also have the property that the Π -type is fibrant as soon as its codomain is. Although Segal-fibrancy (example 8.1.8) is not robust, we can follow Boulier and Tabareau [BT17] in moving to contextual fibrancy, in which case we *can* satisfy the robustness criterion and thus model the directed Π -type under restricted circumstances (proposition 8.6.2) for *some* notion of Segal fibrancy.

The observation that notions of fibrancy such as discreteness and clock-irrelevance [BM18] which are robust can be *defined* internally (section 8.7) whereas non-robust notions of fibrancy such as Kan fibrancy and Segal fibrancy⁹ cannot [Lic+18], prompted the question whether robustness was somehow related to being definable internally.

This question is partially answered in section 8.6: robust notions of fibrancy come with a fibrant replacement monad that can at least be axiomatized

⁸Restricting to those types is in general not feasible, as they are not closed under important type formers.

⁹Riehl and Shulman’s notion of Segal-fibrancy [RS17] is more restrictive than ours in the sense that it only allows for composition of *homogeneous* arrows. In Boulier and Tabareau’s terminology [BT17] we would call this *degenerate* Segal fibrancy.

internally. Then we can internally define the type of algebras for this monad, which are the fibrant types. I proved no results on the internal *definability* of the fibrant replacement monad, but in the case of discreteness it would clearly have to be a quotient inductive type (QIT) [Uni13].

The essential theory of factorization systems is presented in **section 2.4**. This presentation is succinct, but does contain quite a few examples. My own contributions regarding factorization systems are all in **chapter 8**.

1.7 MTT: Well-Behaved Multimode Type Theory

Following Pfenning [Pfe01] and Abel [Abe06; Abe08], we had formulated ParamDTT and RelDTT with a left division operation on contexts: whenever the type-checker moves into a modal subterm, its left Galois connection (left adjoint) is applied to all modality annotations in the context.

While trying to implement a proof-assistant for these type systems [Nuy19; ND19b], I noticed that computation of the left division could be postponed until usage of a variable subject to the division. Moreover, to type-check such usage, it is checked that the variable’s modality is less than the identity modality. But by the Galois connection, the ‘denominator’ modality could again be brought to the right, so that the division in fact *never* needs to be computed. As such, I was able to turn the division from a context operation into a context constructor, inadvertently creating a hybrid with the Fitch-style approach of modal type theory [BGM17; Bir+20; GSB19a].

This hybrid multimode type system subsequently underwent the scrutiny of my co-authors Daniel Gratzer, G. A. Kavvos and Lars Birkedal – who praised it for having a cleaner substitution calculus than other modal type systems, be they Fitch-style or based on left division – which resulted in a paper [Gra+20b] subsumed in section 5.3 of this thesis, and an extensive technical report [Gra+20a].

1.8 Overview

There are two main ways to structure a mathematical text. One way is to present the mathematics as it was developed; this is more or less how this chapter has been structured. The advantage is that the motivation always comes before the problem, and the problem before the solution. A disadvantage

is that we are constantly bumping into obstacles and having to redo earlier work. This makes a text longer and more cluttered on later reference.

For this reason, I wilfully apply the anti-didactic inversion, putting definitions and lemmas before the theorems that need them, so that we only ever have to refer upwards.

Prerequisites

Chapter 2 contains the mathematical prerequisites of this thesis: category theory, presheaves and factorization systems. It is not meant to be read front-to-back. We do recommend readers to have a glance to learn about some peculiar notations.

Chapter 3 on formal systems and dependent type theory (DTT) introduces the notion of generalized algebraic theories (GATs) which we can use as a formal definition of what a type theory is. Next, it introduces DTT and the Curry-Howard correspondence.

Chapter 4 summarizes the general presheaf model of DTT [Hof97; HS97].

Contributions

Chapter 5 is about multimode type theory. It starts with the study of the internalization of CwF morphisms underpinning the semantics of ParamDTT and RelDTT (section 5.1), continues by discussing Birkedal et al.'s independently developed notion of dependent right adjoints (DRAs) [Bir+20] for comparison and completeness (section 5.2), and finally subsumes our MTT paper (section 5.3) [Gra+20b].

Chapter 6 on presheaf type theory is a mix of existing work and contributions, but none that I am especially excited about. It contains a number of extensions to DTT that are meaningful in every presheaf category. We also sketch a type-checking algorithm for the sort of propositions that *Glue*- and *Weld*-types are annotated with. The main reason for this content is to give *some* relatively precise description of these operators when used in an arbitrary presheaf category. The chapter finishes with a brief discussion of interactions with modal type theory.

Chapter 7 presents a type system featuring the transpension type. The type system is obtained by instantiating MTT on a specific mode theory that provides modes such as ‘fresh for i ’, ‘for all i ’ and ‘transpend over i ’ where i is a shape

variable external to MTT. We investigate the structure of the transpension type and explain how to recover known presheaf operators.

Chapter 8 on fibrancy starts by extending factorization systems to *damped* factorization systems in order to reason about *contextual* fibrancy. Next, we define how to generate robust notions of fibrancy and we prove the good properties that it entails: stability of the fibrant replacement, and fibrancy of the Π -type inherited from the codomain alone. We give typing rules to internalize a stable fibrant replacement monad and then use these to internally define fibrant types as the algebras for that monad. Finally, we briefly discuss by example how even without an internal fibrant replacement, we can define some notions of fibrancy internally, if necessary using the transpension type.

Chapter 9 on Degrees of Relatedness, of which all but the last section come from the technical report on MTT [Gra+20a], explains how Reynolds' relationally parametric model of System F [Rey83] evolved into a parametric model of DTT, and how we further extended it to the systems ParamDTT and RelDTT. We explain how improved versions of these systems can be obtained by extending certain instantiations of MTT, and we say a word on how much of the construction of the semantics of RelDTT could be internalized into a *different* instance of MTT. In the final section 9.5, we discuss when one does and does not need a parametric modality in DTT.

Chapter 10 concludes the thesis.

Part I

Prerequisites

Chapter 2

Mathematical Prerequisites

In this chapter, we recall some well-known mathematical concepts, with the purpose of being self-contained and establishing our notations and conventions.

In section 2.1, we establish our default metatheory. In section 2.2, we review important categorical concepts. In section 2.3, we review important concepts related to presheaves and establish our fairly unusual presheaf notations. In section 2.4, we discuss a small fragment of the theory of factorization systems – a categorical notion underpinning the concept of fibrancy in type theory – which we will make use of in chapter 8.

This chapter is mostly intended for reference, rather than for a front-to-back lecture, with the exception of section 2.4 on factorization systems. We recommend readers to check section 2.1 as well as our unusual but in my personal opinion extremely enlightening presheaf notation 2.3.2 (as well as the notations used for working with presheaf models of DTT in chapter 4).

Nearly all concepts in this chapter are standard. Further information about them can be found in the many reference works on category theory. Since there is little use in my referring the reader to a reference work that I am barely acquainted with, I shall refer instead to my favorite one, which is the nLab [nLab]. The nLab contains basic information about *many* concepts, and when the information one finds there is insufficient or incomprehensible, then there are often valuable references to other works. The few non-standard exceptions are:

- A definition of ends and co-ends via the twisted arrow category (section 2.2.6). This is unusual but equivalent to the standard definition.

- Dependent ends and co-ends (section 2.2.7), with associated dependent (co)-Yoneda lemmas (section 2.3.4). These are only used on a few occasions, although they are omnipresent in the technical report associated to chapter 7 [Nuy20].
- In the same spirit, a definition of the category of elements (definition 2.2.37) that is more general than usual.
- Base pullbacks (section 2.3.7), but these are entirely shrugworthy.
- A distinction between NWFSs and Grandis-Tholen NWFSs, although we prove that these notions are equivalent (section 2.4.5).

Notation 2.0.1. The language Haskell contains a feature called *newtypes*. These are types A having a single constructor $a : B \rightarrow A$ and are internally represented as B . The advantage is that if both A and B are endowed with a different structure of the same kind (e.g. both are monoids but with a different operation), then the use of a and a^{-1} tells us whether we are dealing with elements of A or B and hence, what monoid operation we should use.

We find it useful to adopt a similar convention in mathematical texts. We will occasionally define a set $A := \{a(b) \mid b \in B\}$. This essentially means that we define $A := B$ but will strive for maximal notational hygiene in the use of the **label** a and its inverse, which are really just id_B under the hood. In order to signal that something is a label, we will typeset it in gray.

2.1 Set Theory

Unless otherwise mentioned, all metatheory in this thesis takes place in Zermelo-Fraenkel set theory with the axiom of choice (ZFC) [Kun13]. We intend to be explicit about invocations of the axiom of choice. We also assume Grothendieck's axiom of universes [nLa20c]:

Definition 2.1.1. A **Grothendieck universe** is a set in ZFC that is itself a model of ZFC.

Axiom 2.1.2 (Axiom of universes). Every set is an element of a Grothendieck universe.

Definition 2.1.3. For any ordinal number α , let G_α be the smallest Grothendieck universe such that $\mathsf{G}_\beta \in \mathsf{G}_\alpha$ for all $\beta < \alpha$. The size of an object x is the least α such that $x \in \mathsf{G}_\alpha$. An object is small if it has size 0, and large if it is (potentially) not small.

2.2 Category Theory

2.2.1 Categories, Functors and Natural Transformations

Definition 2.2.1. A category \mathcal{C} consists of:

- A set of **objects** $\text{Obj}(\mathcal{C})$,
- For every two objects $x, y \in \text{Obj}(\mathcal{C})$, a set of **morphisms** $\text{Hom}(x, y)$,
- At every object x , an **identity** morphism $\text{id}_x \in \text{Hom}(x, x)$,
- For every three objects $x, y, z \in \text{Obj}(\mathcal{C})$, a **composition** operation $\circ : \text{Hom}(y, z) \times \text{Hom}(x, y) \rightarrow \text{Hom}(x, z)$, such that

$$\varphi \circ \text{id} = \varphi, \quad \text{id} \circ \varphi = \varphi, \quad (\varphi \circ \chi) \circ \psi = \varphi \circ (\chi \circ \psi). \quad (2.1)$$

Notation 2.2.2. We will also write \mathcal{C} for $\text{Obj}(\mathcal{C})$, $\varphi : x \rightarrow y$ for $\varphi \in \text{Hom}(x, y)$ and id for id_x .

Example 2.2.3. Examples of categories are:

- The category **Set** of small sets and functions,
- The category **Grp** of groups and group homomorphisms,
- The category **Vect_K** of vector spaces and linear maps over the field K ,
- The category **Cat** of small categories and functors.
- The category $\mathcal{D}^{\mathcal{C}}$ of functors $\mathcal{C} \rightarrow \mathcal{D}$ and natural transformations.

Definition 2.2.4. A (covariant) **functor** $F : \mathcal{C} \rightarrow \mathcal{D}$ consists of:

- an action on objects $F : \text{Obj}(\mathcal{C}) \rightarrow \text{Obj}(\mathcal{D})$,
- for every $x, y \in \text{Obj}(\mathcal{C})$, an action on morphisms $F : \text{Hom}_{\mathcal{C}}(x, y) \rightarrow \text{Hom}_{\mathcal{D}}(Fx, Fy)$, such that

$$F \text{id}_x = \text{id}_{Fx}, \quad F(\chi \circ \varphi) = F\chi \circ F\varphi. \quad (2.2)$$

Definition 2.2.5. The **opposite** \mathcal{C}^{op} of a category \mathcal{C} is defined by $\text{Obj}(\mathcal{C}^{\text{op}}) = \text{Obj}(\mathcal{C})$ and $\text{Hom}_{\mathcal{C}^{\text{op}}}(x, y) = \text{Hom}_{\mathcal{C}}(y, x)$, with obvious identity and composition. The opposite $F^{\text{op}} : \mathcal{C}^{\text{op}} \rightarrow \mathcal{D}^{\text{op}}$ of a functor $F : \mathcal{C} \rightarrow \mathcal{D}$ is defined the obvious way. A **contravariant functor** from \mathcal{C} to \mathcal{D} is a functor $\mathcal{C}^{\text{op}} \rightarrow \mathcal{D}$.

Every statement about a category \mathcal{C} can be expressed in terms of \mathcal{C}^{op} , and often this saves proving work. This phenomenon is referred to as duality.

Definition 2.2.6. A **natural transformation** $\alpha : F \rightarrow G : \mathcal{C} \rightarrow \mathcal{D}$ consists of a morphism $\alpha_x : Fx \rightarrow Gx$ for every $x \in \mathcal{C}$ such that for any $\varphi : x \rightarrow y$, we have $\alpha_y \circ F\varphi = G\varphi \circ \alpha_x =: \alpha \star \varphi : Fx \rightarrow Gy$. A **natural isomorphism** is a natural transformation that has an inverse. The **composite** $\beta \circ \alpha : F \rightarrow H$ of $\alpha : F \rightarrow G$ and $\beta : G \rightarrow H$ is defined by objectwise composition. The **whiskering** $\alpha' \star \alpha : F'F \rightarrow G'G$ of $\alpha : F \rightarrow G : \mathcal{C} \rightarrow \mathcal{D}$ and $\alpha' : F' \rightarrow G' : \mathcal{D} \rightarrow \mathcal{E}$ is defined by $(\alpha' \star \alpha)_x := \alpha' \star \alpha_x$. We also write $\alpha'F$ for $\alpha' \star \text{id}_F$ and $F'\alpha$ for $\text{id}_{F'} \star \alpha$.

Remark 2.2.7. More generally and more informally, an operation on categories is called **natural** w.r.t. an object argument if it satisfies all equations that can be cooked up from a morphism between objects inserted at that argument.

Definition 2.2.8. The **functor category** $\mathcal{D}^{\mathcal{C}}$ has as objects the functors $\mathcal{C} \rightarrow \mathcal{D}$ and as morphisms natural transformations.

2.2.2 Mono- and Epimorphisms

Definition 2.2.9. A morphism $\varphi \in \text{Hom}(x, y)$ is called:

- a **monomorphism** or just **mono** if $\varphi \circ \sqsubset : \text{Hom}(w, x) \rightarrow \text{Hom}(w, y)$ is injective,
- an **epimorphism** or just **epi** if $\sqsupset \circ \varphi : \text{Hom}(y, z) \rightarrow \text{Hom}(x, z)$ is injective,
- **split mono** if $\sqsupset \circ \varphi : \text{Hom}(y, z) \rightarrow \text{Hom}(x, z)$ is surjective,
- **split epi** if $\varphi \circ \sqsubset : \text{Hom}(w, x) \rightarrow \text{Hom}(w, y)$ is surjective.

Proposition 2.2.10. A split monomorphism is a morphism $\varphi : x \rightarrow y$ that has a **retraction** $\rho : y \rightarrow x$ such that $\rho \circ \varphi = \text{id}_x$. Dually, a split epimorphism is a morphism $\varphi : x \rightarrow y$ that has a **section** $\sigma : y \rightarrow x$ such that $\varphi \circ \sigma = \text{id}_y$.

Proof. We only prove the latter statement. Let $\varphi : x \rightarrow y$ be split epi. Then, by split surjectivity, $\text{id}_y : y \rightarrow y$ is of the form $\varphi \circ \sigma$ for some $\sigma : y \rightarrow x$. Conversely, let $\varphi : x \rightarrow y$ have a section σ , and pick $\chi : w \rightarrow y$. Then $\chi = \varphi \circ (\sigma \circ \chi)$. \square

Corollary 2.2.11. Split monomorphisms are mono. Dually, split epimorphisms are epi. \square

2.2.3 Full and Faithful

Definition 2.2.12. A functor $F : \mathcal{C} \rightarrow \mathcal{D}$ is:

- **full** if it is surjective on Hom-sets,

- **faithful** if it is injective on Hom-sets,
- **fully faithful** if it is bijective on Hom-sets.

Definition 2.2.13. A subcategory $\mathcal{D} \subseteq \mathcal{C}$ of a category \mathcal{C} is a category \mathcal{D} such that $\text{Obj}(\mathcal{D}) \subseteq \text{Obj}(\mathcal{C})$ and, for all $x, y \in \text{Obj}(\mathcal{D})$, we have $\text{Hom}_{\mathcal{D}}(x, y) \subseteq \text{Hom}_{\mathcal{C}}(x, y)$. This means the inclusion functor $\mathcal{D} \rightarrow \mathcal{C}$ is necessarily faithful.

A full subcategory is a subcategory $\mathcal{D} \subseteq \mathcal{C}$ such that for all $x, y \in \text{Obj}(\mathcal{D})$, we have $\text{Hom}_{\mathcal{D}}(x, y) = \text{Hom}_{\mathcal{C}}(x, y)$. This is the case if and only if the inclusion functor is fully faithful.

2.2.4 Arrow Categories

Definition 2.2.14. The **walking arrow** \uparrow is the category with two objects 0 and 1 and a single non-identity morphism $0 \rightarrow 1$.

Definition 2.2.15. The **arrow category** of \mathcal{C} is defined to be the functor category \mathcal{C}^{\uparrow} . It has as objects triples (x, y, φ) , also denoted $(x \xrightarrow{\varphi} y)$ where $x, y \in \mathcal{C}$ and $\varphi : x \rightarrow y$, and as morphisms $(x \xrightarrow{\varphi} y) \rightarrow (x' \xrightarrow{\varphi'} y')$ commutative squares (ξ, ψ) :

$$\begin{array}{ccc} x & \xrightarrow{\xi} & x' \\ \varphi \downarrow & & \downarrow \varphi' \\ y & \xrightarrow{\psi} & y'. \end{array}$$

It is equipped with obvious functors $\text{Dom} : \mathcal{C}^{\uparrow} \rightarrow \mathcal{C}$ and $\text{Cod} : \mathcal{C}^{\uparrow} \rightarrow \mathcal{C}$ and an obvious morphism $\text{mor} : \text{Dom} \rightarrow \text{Cod}$.

Definition 2.2.16. The **twisted arrow category** $\text{Tw}(\mathcal{C})$ of \mathcal{C} has the same objects as the arrow category, and as morphisms $(x \xrightarrow{\varphi} y) \rightarrow (x' \xrightarrow{\varphi'} y')$ commutative squares (ξ, ψ) :

$$\begin{array}{ccc} x & \xleftarrow{\xi} & x' \\ \varphi \downarrow & & \downarrow \varphi' \\ y & \xrightarrow{\psi} & y'. \end{array}$$

It is equipped with obvious functors $\text{Dom} : \text{Tw}(\mathcal{C}) \rightarrow \mathcal{C}^{\text{op}}$ and $\text{Cod} : \text{Tw}(\mathcal{C}) \rightarrow \mathcal{C}$.

Notation 2.2.17. We will sometimes write $\vec{x} = (x_0 \xrightarrow{x_1} x_1)$ for elements of the (twisted) arrow category, and $\vec{\varphi} = (\varphi_0, \varphi_1)$ for morphisms.

2.2.5 Limits and Colimits

Definition 2.2.18. A **diagram** in a category \mathcal{C} is a functor $F : \mathcal{I} \rightarrow \mathcal{C}$ from some other category \mathcal{I} .

The idea is that \mathcal{I} is the shape of the diagram, determining the number of nodes (as the cardinality of $\text{Obj}(\mathcal{I})$), where the arrows go, and which arrows should compose to which other arrows. Meanwhile, the functor F sends nodes to their label (an object in \mathcal{C}), arrows to their label (a morphism in \mathcal{C}) and functoriality requires that the required composition laws hold.

Definition 2.2.19. A **limit** of a diagram $F : \mathcal{I} \rightarrow \mathcal{C}$ is an object $\lim F = \lim_i F_i \in \mathcal{C}$ such that there is a 1-1 correspondence, natural in x , between morphisms $x \rightarrow \lim F$ and collections, natural in i , of morphisms $x \rightarrow F_i$.

Dually, a **colimit** of a diagram $F : \mathcal{I} \rightarrow \mathcal{C}$ is an object $\text{colim } F = \text{colim}_i F_i \in \mathcal{C}$ such that there is a 1-1 correspondence, natural in y , between morphisms $\text{colim } F \rightarrow y$ and collections, natural in i , of morphisms $F_i \rightarrow y$.

Limits and colimits do not always exist, but are always unique up to isomorphism.

In Set_α , the category of sets of size α , we can construct (co)limits over diagrams of size α explicitly:

Proposition 2.2.20. For \mathcal{I} of size α , a limit of $F : \mathcal{I} \rightarrow \text{Set}_\alpha$ is given by the set of dependent functions $f : (i \in \mathcal{I}) \rightarrow F_i$ such that, for any morphism $\varphi : i \rightarrow j$, we have $(F\varphi)(f_i) = f_j$. \square

Proposition 2.2.21. For \mathcal{I} of size α , a colimit of $F : \mathcal{I} \rightarrow \text{Set}_\alpha$ is given by the set of dependent pairs $(i \in \mathcal{I}) \times F_i$ divided by the least equivalence relation that relates (i, x) and $(j, (F\varphi)(x))$ for every $\varphi : i \rightarrow j$. \square

Remark 2.2.22. Using limits in set, the (co)limit of $F : \mathcal{I} \rightarrow \mathcal{C}$ may be defined more succinctly using natural isomorphisms:

$$\text{Hom}_{\mathcal{C}}(\sqcup, \lim_i F_i) \cong \lim_i \text{Hom}_{\mathcal{C}}(\sqcup, F_i) : \mathcal{C} \rightarrow \text{Set},$$

$$\text{Hom}_{\mathcal{C}}(\text{colim}_i F_i, \sqcup) \cong \lim_i \text{Hom}_{\mathcal{C}}(F_i, \sqcup) : \mathcal{C} \rightarrow \text{Set}.$$

Example 2.2.23. Some limits and colimits get special names and/or notations:

- A **binary (cartesian) product** $A \times B$ is a limit of the discrete diagram $A \quad B$ (which is a diagram whose shape is the category with two objects and only identity morphisms). We have $\text{Hom}(X, A \times B) \cong \text{Hom}(X, A) \times \text{Hom}(X, B)$.

- A **binary coproduct** $A \uplus B$ is a colimit of the discrete diagram $A \quad B$. We have $\text{Hom}(A \uplus B, Y) \cong \text{Hom}(A, Y) \times \text{Hom}(B, Y)$.
- For any set I , a **(cartesian) product** $\prod_{i \in I} A_i$ is a limit of the discrete diagram given by the functor $i \mapsto A_i$ over the discrete category (with only identity arrows) on I . We have $\text{Hom}(X, \prod_{i \in I} A_i) \cong \prod_{i \in I} \text{Hom}(X, A_i)$.
- For any set I , a **coproduct** $\coprod_{i \in I} A_i$ is a colimit of the discrete diagram $i \mapsto A_i$ over the discrete category on I . We have $\text{Hom}(\coprod_{i \in I} A_i, Y) \cong \prod_{i \in I} \text{Hom}(A_i, Y)$.
- A **pullback** $A \times_B C$ is a limit of a diagram $A \rightarrow B \leftarrow C$.
- A **pushout** $A \uplus_B C$ is a colimit of a diagram $A \leftarrow B \rightarrow C$.
- A **final** or **terminal object** \top is a limit of the empty diagram. We have $\text{Hom}(X, \top) \cong \top$, a singleton.
- An **initial object** \perp is a colimit of the empty diagram. We have $\text{Hom}(\perp, Y) \cong \top$, a singleton.
- An **equalizer** E of $\varphi, \chi : A \rightarrow B$ is a limit of the diagram $A \rightrightarrows_{\varphi, \chi} B$. The canonical morphism $E \rightarrow A$ is also referred to as the equalizer. In **Set**, we have $E \cong \{x \in A \mid \varphi(x) = \chi(x)\}$.
- A **coequalizer** E of $\varphi, \chi : A \rightarrow B$ is a colimit of the diagram $A \rightrightarrows_{\varphi, \chi} B$. The canonical morphism $B \rightarrow E$ is also referred to as the coequalizer. In **Set**, E is B divided by the least equivalence relation that relates $\varphi(x)$ with $\chi(x)$ for all $x \in A$.

Definition 2.2.24. A natural transformation $\nu : F \rightarrow G$ is called **cartesian** (or **equifibred**) if for every $\varphi : x \rightarrow y$, the diagram

$$\begin{array}{ccc}
 Fx & \xrightarrow{\nu_x} & Gx \\
 F\varphi \downarrow \lrcorner & & \downarrow F\varphi \\
 Fy & \xrightarrow{\nu_y} & Gy
 \end{array}$$

is a pullback square, i.e. Fx is the pullback of the rest of the square.

2.2.6 Ends and Co-ends

Proposition 2.2.25. We have a covariant invertible functor $\text{Tw}(\mathcal{C}) \cong \text{Tw}(\mathcal{C}^{\text{op}}) : (x \xrightarrow{\varphi} y) \mapsto (y \xrightarrow{\varphi} x)$.

Definition 2.2.26. The **end** $\forall i.F(i, i)$ of a functor $F : \mathcal{I}^{\text{op}} \times \mathcal{I} \rightarrow \mathcal{C}$ is the limit of $F \circ (\text{Dom}, \text{Cod}) : \text{Tw}(\mathcal{I}) \rightarrow \mathcal{C}$.

Dually, the **co-end** $\exists i.F(i, i)$ of a functor $F : \mathcal{I}^{\text{op}} \times \mathcal{I} \rightarrow \mathcal{C}$ is the colimit of $F \circ (\text{Cod}^{\text{op}}, \text{Dom}^{\text{op}}) : \text{Tw}(\mathcal{I})^{\text{op}} \rightarrow \mathcal{C}$.

The general construction of (co)limits in Set_α would have us consider dependent functions/pairs with a component from $\text{Tw}(\mathcal{I})$. However, we can simplify:

Proposition 2.2.27. For \mathcal{I} of size α , an end of $F : \mathcal{I}^{\text{op}} \times \mathcal{I} \rightarrow \text{Set}_\alpha$ is given by the set of dependent functions $f : (i \in \mathcal{I}) \rightarrow F(i, i)$ such that, for any morphism $\varphi : i \rightarrow j$, we have $F(\text{id}, \varphi)(f i) = F(\varphi, \text{id})(f j)$. \square

Proposition 2.2.28. For \mathcal{I} of size α , a co-end of $F : \mathcal{I}^{\text{op}} \times \mathcal{I} \rightarrow \text{Set}_\alpha$ is given by the set of dependent pairs $(i \in \mathcal{I}) \times F(i, i)$ divided by the least equivalence relation that relates $(i, F(\varphi, \text{id})(z))$ and $(j, F(\text{id}, \varphi)(z))$ for every $\varphi : i \rightarrow j$ and $z \in F(j, i)$. \square

Unless otherwise mentioned, we will assume ends and co-ends of sets to be constructed as above.

Lemma 2.2.29. Given two functors $F : \mathcal{I}^{\text{op}} \rightarrow \text{Set}$ and $G : \mathcal{I} \rightarrow \text{Set}$, construct the co-end $\exists i.Fi \times Gi$ as in proposition 2.2.28. Then if two elements (i, x, y) and (i', x', y') are equal, there exists a zigzag ζ from i to i' , i.e. a diagram

$$i \rightarrow i_1 \leftarrow i_2 \rightarrow \dots \leftarrow i_{t-1} \rightarrow i_t \leftarrow i' \quad (2.3)$$

abbreviated to

$$i \overset{\zeta}{\rightsquigarrow} i' \quad (2.4)$$

as well as elements $x_r \in Fi_r$ and $y_r \in Gi_r$ for every node i_r of the zigzag such that for every arrow $\varphi : i_r \rightarrow i_s$ in the zigzag, we have $x_r = F\varphi(x_s)$ and $G\varphi(y_r) = y_s$. \square

Proposition 2.2.30. For any functor $F : \mathcal{I} \rightarrow \mathcal{C}$, we have $\lim_i Fi \cong \forall i.Fi$ and $\text{colim}_i Fi \cong \exists i.Fi$.

Proof. Note that our notation implies we are precomposing F with $\mathcal{I}^{\text{op}} \times \mathcal{I} \rightarrow \mathcal{I}$. By remark 2.2.22, it is sufficient to prove $\lim_i Fi \cong \forall i.Fi$ for $F : \mathcal{I} \rightarrow \text{Set}$, and that follows immediately from propositions 2.2.20 and 2.2.27. \square

2.2.7 Dependent Ends and Co-ends

The definitions in this section are non-standard. In the previous section, we defined ends and co-ends of functors from $\mathcal{I}^{\text{op}} \times \mathcal{I}$ by restricting them to $\text{Tw}(\mathcal{I})$ or $\text{Tw}(\mathcal{I})^{\text{op}}$. In this section, we allow the functor to actually use not only domain and codomain of the twisted arrow, but also the arrow itself.

Definition 2.2.31. A **dependent end** of a functor $F : \text{Tw}(\mathcal{I}) \rightarrow \mathcal{C}$, somewhat ambiguously denoted $\forall i.F(i \xrightarrow{\text{id}} i)$, is a limit of F .

Definition 2.2.32. A **dependent co-end** of a functor $F : \text{Tw}(\mathcal{I})^{\text{op}} \rightarrow \mathcal{C}$, somewhat ambiguously denoted $\exists i.F(i \xrightarrow{\text{id}} i)$, is a colimit of F .

Proposition 2.2.33. For \mathcal{I} of size α , a dependent end of $F : \text{Tw}(\mathcal{I}) \rightarrow \text{Set}_\alpha$ is given by the set of dependent functions $f : (i \in \mathcal{I}) \rightarrow F(i \xrightarrow{\text{id}} i)$ such that, for any morphism $\varphi : i \rightarrow j$, we have $F(\text{id}, \varphi)(f i) = F(\varphi, \text{id})(f j)$. \square

Proposition 2.2.34. For \mathcal{I} of size α , a dependent co-end of $F : \text{Tw}(\mathcal{I})^{\text{op}} \rightarrow \text{Set}_\alpha$ is given by the set of dependent pairs $(i \in \mathcal{I}) \times F(i \xrightarrow{\text{id}} i)$ divided by the least equivalence relation that relates $(i, F(\varphi, \text{id})(z))$ and $(j, F(\text{id}, \varphi)(z))$ for every $\varphi : i \rightarrow j$ and $z \in F(i \xrightarrow{\varphi} j)$. \square

Unless otherwise mentioned, we will assume dependent ends and co-ends of sets to be constructed as above.

Example 2.2.35. Assume a functor $G : \mathcal{C} \rightarrow \mathcal{D}$. One way to denote the set of natural transformations $\text{Id}_\mathcal{C} \rightarrow \text{Id}_\mathcal{C}$ which map to the identity under G , is as

$$A := \forall (c \in \mathcal{C}). \{ \chi : c \rightarrow c \mid G\chi = \text{id}_{Gc} \}.$$

In order to read the above as a dependent end, we must find a functor $G : \text{Tw}(\mathcal{C}) \rightarrow \text{Set}$ such that $G(c \xrightarrow{\text{id}} c) = \{ \chi : c \rightarrow c \mid G\chi = \text{id}_{Gc} \}$. Clearly every covariant occurrence of c refers to the codomain of $(c \xrightarrow{\text{id}} c)$, whereas every contravariant occurrence refers to the domain. So when we apply G to a general object $(x \xrightarrow{\varphi} y)$ of $\text{Tw}(\mathcal{C})$, we should substitute x for every contravariant c and y for every covariant c . We can then throw in φ wherever this is necessary to keep things well-typed, as φ disappears anyway when $(x \xrightarrow{\varphi} y) = (c \xrightarrow{\text{id}} c)$. Thus, we get

$$G(x \xrightarrow{\varphi} y) = \{ \chi : x \rightarrow y \mid G\chi = G\varphi \}.$$

So we see that using a *dependent* end was necessary in order to mention id_c , as this generalizes to $\varphi : x \rightarrow y$ to which we do not have access in a non-dependent end.

An element of $\nu \in A$ is then a function

$$\nu : (c \in \mathcal{C}) \rightarrow \{ \chi : c \rightarrow c \mid G\chi = \text{id}_{Gc} \}$$

such that, whenever $\varphi : x \rightarrow y$, we have $\varphi \circ \nu_x = \nu_y \circ \varphi$.

2.2.8 Slices and Elements

Definition 2.2.36. Given $U \in \mathcal{W}$, the **slice category**¹ \mathcal{W}/U has objects (W, ψ) with $\psi : W \rightarrow U$ and as morphisms $(V, \varphi) \rightarrow (W, \psi)$ all morphisms $\chi : V \rightarrow W$ such that $\psi = \varphi \circ \chi$. It is equipped with a functor $\Sigma_U : \mathcal{W}/U \rightarrow \mathcal{W} : (W, \psi) \mapsto W$, and for every morphism $v : U \rightarrow U'$, we get a functor $\Sigma^v : \mathcal{W}/U \rightarrow \mathcal{W}/U' : (W, \psi) \rightarrow (W, v \circ \psi)$.

The **coslice category**² U/\mathcal{W} is defined dually.

Definition 2.2.37. Given a functor $F : \text{Tw}(\mathcal{C}) \rightarrow \text{Set}$, the **category of elements**

$$\int_{c \in \mathcal{C}} F(c \xrightarrow{\text{id}_c} c)$$

has objects (c, f) where $c \in \text{Obj}(\mathcal{C})$ and $f \in F(c \xrightarrow{\text{id}_c} c)$ and morphisms $\varphi : (c, f) \rightarrow (d, g)$ where $\varphi : c \rightarrow d$ and $F(\text{id}_c, \varphi) f = F(\varphi, \text{id}_d) g \in F(c \xrightarrow{\varphi} d)$.

The category of elements of $G \circ \text{Dom}$ (or $H \circ \text{Cod}$), where $G : \mathcal{C}^{\text{op}} \rightarrow \text{Set}$ (or $H : \mathcal{C} \rightarrow \text{Set}$) is also called the category of elements of G (or H) and denoted as $\int_{c \in \mathcal{C}} Gc$ (or $\int_{c \in \mathcal{C}} Hc$) which is in fact not an exception to the above notation.

Example 2.2.38 (Algebras for an endofunctor). Assume an endofunctor $G : \mathcal{C} \rightarrow \mathcal{C}$. The category of G -algebras is defined as

$$\int_{c \in \mathcal{C}} \text{Hom}(Gc, c).$$

Inspecting the variance of the ‘integrand’, reveals that we are taking the category of elements of the functor $F : \text{Tw}(\mathcal{C}) \rightarrow \text{Set} : (x \xrightarrow{\varphi} y) \mapsto \text{Hom}(Gx, y)$. Thus, an object is a pair (c, χ) where $\chi : Gc \rightarrow c$ and a morphism $\varphi : (c, \chi) \rightarrow (d, \delta)$ is a morphism $\varphi : c \rightarrow d$ such that $\varphi \circ \chi = \delta \circ G\varphi$.

Example 2.2.39 (Algebras for a pointed endofunctor). Assume a pointed endofunctor $G : \mathcal{C} \rightarrow \mathcal{C}$ with pointing $\eta : \text{Id} \rightarrow G$. The category of (G, η) -algebras is defined as

$$\int_{c \in \mathcal{C}} \{\chi : Gc \rightarrow c \mid \chi \circ \eta_c = \text{id}_c\}.$$

Clearly, we are then taking the category of elements of the functor

$$F : \text{Tw}(\mathcal{C}) \rightarrow \text{Set} : (x \xrightarrow{\varphi} y) \mapsto \{\chi : Gx \rightarrow y \mid \chi \circ \eta_x = \varphi\}.$$

Thus, an object is a pair (c, χ) where $\chi : Gc \rightarrow c$ such that $\chi \circ \eta_c = \text{id}_c$, and a morphism $\varphi : (c, \chi) \rightarrow (d, \delta)$ is a morphism $\varphi : c \rightarrow d$ such that $\varphi \circ \chi = \delta \circ G\varphi$.

¹also called ‘over category’

²also called ‘under category’

Example 2.2.40 (Eilenberg-Moore algebras for a monad). Assume a monad (M, η, μ) (definition 2.2.46). The following attempt at defining the category of Eilenberg-Moore algebras for M (definition 2.2.49) is ill-formed:

$$\int_{c \in \mathcal{C}} \{\chi : Mc \rightarrow c \mid \chi \circ \eta_c = \text{id} \text{ and } \chi \circ \mu_c = \chi \circ M\chi\}.$$

Indeed, the ‘integrand’ is not an instance of a functor

$$F : \text{Tw}(\mathcal{C}) \rightarrow \text{Set} :$$

$$(x \xrightarrow{\varphi} y) \mapsto \{\chi : Mx \rightarrow y \mid \chi \circ \eta_x = \varphi \text{ and } \chi \circ \mu_x = \chi \circ \sqcup \circ M\chi\}.$$

since there is no morphism $My \rightarrow Mx$ available which we can insert in the gap.

2.2.9 Adjunctions

Definition 2.2.41. Two functors $L : \mathcal{D} \rightarrow \mathcal{C}$ and $R : \mathcal{C} \rightarrow \mathcal{D}$ are **adjoint**, denoted $L \dashv R$ with L being the left adjoint and R the right adjoint, if they are equipped with either of the following equally informative structures:

- a natural transformation $\mathbf{A} : \text{Hom}_{\mathcal{C}}(L_, _) \cong \text{Hom}_{\mathcal{D}}(_, R_) : \mathcal{D}^{\text{op}} \times \mathcal{C} \rightarrow \text{Set}$ (called transposition),
- natural transformations $\eta : \text{Id}_{\mathcal{D}} \rightarrow RL$ (the unit) and $\varepsilon : LR \rightarrow \text{Id}_{\mathcal{C}}$ (the co-unit) such that $\varepsilon L \circ L\eta = \text{id}_L$ and $R\varepsilon \circ \eta R = \text{id}_R$ (the adjunction laws).

Lemma 2.2.42. The above structures are indeed equally informative.

Proof. Given \mathbf{A} , we define $\eta_d := \mathbf{A}(\text{id}_{Ld})$ and $\varepsilon_c := \mathbf{A}^{-1}(\text{id}_{Rc})$. These transformations are natural:

$$RL\varphi \circ \eta = RL\varphi \circ \mathbf{A}(\text{id}) = \mathbf{A}(L\varphi) = \mathbf{A}(\text{id}) \circ \varphi = \eta \circ \varphi,$$

$$\varepsilon \circ LR\psi = \mathbf{A}^{-1}(\text{id}) \circ LR\psi = \mathbf{A}^{-1}(R\psi) = \psi \circ \mathbf{A}^{-1}(\text{id}) = \psi \circ \varepsilon.$$

The adjunction laws hold:

$$\varepsilon L \circ L\eta = \mathbf{A}^{-1}(\text{id}_{RLd}) \circ L\mathbf{A}(\text{id}_{Ld}) = \mathbf{A}^{-1}(\text{id}_{RLd} \circ \mathbf{A}(\text{id}_{Ld})) = \text{id}_{Ld},$$

$$R\varepsilon \circ \eta R = R\mathbf{A}^{-1}(\text{id}_{Rc}) \circ \mathbf{A}(\text{id}_{LRc}) = \mathbf{A}(\mathbf{A}^{-1}(\text{id}_{Rc}) \circ \text{id}_{LRc}) = \text{id}_{Rc}.$$

Conversely, given ε and η we define $\mathbf{A}(\psi) = R\psi \circ \eta$ and $\mathbf{A}^{-1}(\varphi) = \varepsilon \circ L\varphi$, which are mutually inverse by the adjunction laws and naturality of the unit/co-unit.

It is straightforward to verify that going to and fro yields the original \mathbf{A} or unit/co-unit. \square

Lemma 2.2.43. Let $L \dashv R$.

- Natural transformations $LF \rightarrow G$ correspond to natural transformations $F \rightarrow RG$, naturally in F and G .
- Natural transformations $FR \rightarrow G$ correspond to natural transformations $F \rightarrow GL$, naturally in F and G .

Proof. The first statement is trivial.

To see the second statement, we send $\zeta : FR \rightarrow G$ to $\zeta L \circ F\eta : F \rightarrow GL$, and conversely $\theta : F \rightarrow GL$ to $G\varepsilon \circ \theta R : FR \rightarrow G$. Naturality is clear. Mapping ζ to and fro, we get

$$G\varepsilon \circ \zeta LR \circ F\eta R = \zeta \circ FR\varepsilon \circ F\eta R = \zeta. \quad (2.5)$$

Mapping θ to and fro, we get

$$G\varepsilon L \circ \theta RL \circ F\eta = G\varepsilon L \circ GL\eta \circ \theta = \theta. \quad \square$$

Proposition 2.2.44. If a functor has two left/right adjoints, then these are naturally isomorphic in a manner compatible with the unit, co-unit and transposition.

Proof. We prove this for left adjoints. Assume $L_1, L_2 \dashv R$ where $L_1, L_2 : \mathcal{D} \rightarrow \mathcal{C}$ and $R : \mathcal{C} \rightarrow \mathcal{D}$. Then by lemma 2.2.43 we have, naturally in $F : \mathcal{D} \rightarrow \mathcal{C}$:

$$(L_1 \rightarrow F) \cong_{\mathbf{A}_1} (\text{Id}_{\mathcal{D}} \rightarrow RF) \cong_{\mathbf{A}_2^{-1}} (L_2 \rightarrow F).$$

Entering id_{L_1} on the left yields $\lambda_{2,1} : L_2 \rightarrow L_1$ on the right.

Entering id_{L_2} on the right yields $\lambda_{1,2} : L_1 \rightarrow L_2$ on the left.

Entering $\lambda_{1,2}$ on the left should again yield id_{L_2} , but it also yields $(\mathbf{A}_2^{-1} \circ \mathbf{A}_1)(\lambda_{1,2} \circ \text{id}_{L_1}) = \lambda_{1,2} \circ (\mathbf{A}_2^{-1} \circ \mathbf{A}_1)(\text{id}_{L_1}) = \lambda_{1,2} \circ \lambda_{2,1}$. Adding a symmetric observation, we find that $\lambda_{2,1} = \lambda_{1,2}^{-1}$. We shall now write $\lambda = \lambda_{1,2} : L_1 \cong L_2$.

To show compatibility:

- A** We should show for any $\nu : L_2 \rightarrow F$, that $\mathbf{A}_1(\nu \circ \lambda) = \mathbf{A}_2(\nu) : \text{Id} \rightarrow RF$, and we have

$$\mathbf{A}_1(\nu \circ \lambda) = R\nu \circ \mathbf{A}_1(\lambda), \quad \mathbf{A}_2(\nu \circ \text{id}) = R\nu \circ \mathbf{A}_2(\text{id}),$$

and $\mathbf{A}_1(\lambda) = \mathbf{A}_2(\text{id})$ by definition of λ .

η We should show that $R\lambda \circ \eta_1 = \eta_2 : \text{Id} \rightarrow RL_2$. We have

$$R\lambda \circ \eta_1 = R\lambda \circ \mathbf{A}_1(\text{id}_{L_1}) = \mathbf{A}_1(\lambda) = \mathbf{A}_2(\text{id}_{L_2}) = \eta_2.$$

ε We should show that $\varepsilon_1 = \varepsilon_2 \circ \lambda R : L_1 R \rightarrow \text{Id}$. We have

$$\mathbf{A}_1(\varepsilon_2 \circ \lambda R) = \mathbf{A}_2(\varepsilon_2) = \text{id}_R = \mathbf{A}_1(\varepsilon_1),$$

so applying \mathbf{A}_1^{-1} completes the proof. \square

2.2.10 Algebras and Coalgebras

Definition 2.2.45. An **algebra** for an endofunctor F is an object a equipped with a morphism $\alpha : Fa \rightarrow a$. An **algebra morphism** $\varphi : (a, \alpha) \rightarrow (b, \beta)$ is a morphism $\varphi : a \rightarrow b$ such that $\varphi \circ \alpha = \beta \circ F\varphi$.

Dually, a **coalgebra** for an endofunctor F is an object a equipped with a morphism $\alpha : a \rightarrow Fa$. A **coalgebra morphism** $\varphi : (a, \alpha) \rightarrow (b, \beta)$ is a morphism $\varphi : a \rightarrow b$ such that $\beta \circ \varphi = F\varphi \circ \alpha$.

2.2.11 Monads and Comonads

The content of this section is entirely standard; Voutas has an exquisite brief note that goes a bit more in depth [Vou12].

Definition 2.2.46. A **monad** is an endofunctor $M : \mathcal{C} \rightarrow \mathcal{C}$ equipped with natural transformations $\eta : \text{Id} \rightarrow M$ (the **unit**) and $\mu : MM \rightarrow M$ (the monadic **multiplication**) such that $\mu \circ \eta M = \mu \circ M\eta = \text{id} : M \rightarrow M$ (unit laws) and $\mu \circ \mu M = \mu \circ M\mu : M^3 \rightarrow M$ (associativity).

A **comonad** is an endofunctor $K : \mathcal{C} \rightarrow \mathcal{C}$ equipped with natural transformations $\varepsilon : K \rightarrow \text{Id}$ (the **co-unit**) and $\delta : K \rightarrow KK$ (the comonadic **duplication**) such that $\varepsilon K \circ \delta = K\varepsilon \circ \delta = \text{id} : K \rightarrow K$ and $\varepsilon K \circ \varepsilon = K\varepsilon \circ \varepsilon : K \rightarrow K^3$.

Endofunctors equipped with just a unit η (co-unit ε) but without multiplication μ (duplication δ) are also called **pointed (copointed) endofunctors**.

Bind and Extend

Proposition 2.2.47. A monad is equivalently characterized as a functor $M : \mathcal{C} \rightarrow \mathcal{C}$ equipped with natural transformations $\eta : \text{Id} \rightarrow M$ (the unit) and $\text{bind} :$

$\forall x, y. \text{Hom}(x, My) \rightarrow \text{Hom}(Mx, My)$ such that $\text{bind}(\eta_x) = \text{id}_{Mx}$, $\text{bind}(\varphi) \circ \eta = \varphi$ (unit laws) and $\text{bind}(\chi) \circ \text{bind}(\varphi) = \text{bind}(\text{bind}(\chi) \circ \varphi)$ (associativity).

A comonad is equivalently characterized as a functor $K : \mathcal{C} \rightarrow \mathcal{C}$ equipped with natural transformations $\varepsilon : K \rightarrow \text{Id}$ (the co-unit) and $\text{extend} : \forall x, y. \text{Hom}(Kx, y) \rightarrow \text{Hom}(Kx, Ky)$ such that $\text{extend}(\varepsilon_x) = \text{id}_{Kx}$, $\varepsilon \circ \text{extend}(\varphi) = \varphi$ (co-unit laws) and $\text{extend}(\varphi) \circ \text{extend}(\chi) = \text{extend}(\varphi \circ \text{extend}(\chi))$.

Proof. We prove this for monads.

We can define $\mu_x = \text{bind}(\text{id}_{Mx}) : MMx \rightarrow Mx$. Then we have

$$\begin{aligned} \mu \circ \eta &= \text{bind}(\text{id}) \circ \eta = \text{id}, \\ \mu \circ M\eta &= \text{bind}(\text{id}) \circ M\eta = \text{bind}(\text{id} \circ \eta) = \text{bind}(\eta) = \text{id}, \\ \mu \circ M\mu &= \text{bind}(\text{id}) \circ M\mu = \text{bind}(\text{id} \circ \mu) \\ &= \text{bind}(\text{bind}(\text{id}) \circ \text{id}) = \text{bind}(\text{id}) \circ \text{bind}(\text{id}) = \mu \circ \mu M. \end{aligned}$$

Conversely, we can define $\text{bind}(\varphi) = \mu \circ M\varphi$. Then we have

$$\begin{aligned} \text{bind}(\eta) &= \mu \circ M\eta = \text{id}, \\ \text{bind}(\varphi) \circ \eta &= \mu \circ M\varphi \circ \eta = \mu \circ \eta M \circ \varphi = \varphi, \\ \text{bind}(\text{bind}(\chi) \circ \varphi) &= \mu \circ M(\mu \circ M\chi) \circ M\varphi \\ &= (\mu \circ M\mu) \circ MM\chi \circ M\varphi = (\mu \circ \mu M) \circ MM\chi \circ M\varphi \\ &= \mu \circ M\chi \circ \mu \circ M\varphi = \text{bind}(\chi) \circ \text{bind}(\varphi). \end{aligned}$$

Finally, these definitions are mutually inverse:

$$\mu \circ M\text{id}_{Mx} = \mu, \quad \text{bind}(\text{id}) \circ M\varphi = \text{bind}(\varphi). \quad \square$$

Adjunctions and (Co)algebras

Proposition 2.2.48. Any two adjoint functors $L \dashv R$ give rise to a monad RL with unit $\eta : \text{Id} \rightarrow RL$ and multiplication $R\varepsilon L : RLRL \rightarrow RL$, and to a comonad LR with unit $L\eta R : LR \rightarrow LRLR$ and co-unit $\varepsilon : LR \rightarrow \text{Id}$. \square

Definition 2.2.49. An **Eilenberg-Moore algebra** or **monad algebra** for a monad M is an object a equipped with a morphism $\alpha : Ma \rightarrow a$ such that

$\alpha \circ \eta_a = \text{id}_a : a \rightarrow a$ and $\alpha \circ M\alpha = \alpha \circ \mu_a : MMa \rightarrow a$. A morphism of Eilenberg-Moore algebras $\varphi : (a, \alpha) \rightarrow (b, \beta)$ is a morphism $\varphi : a \rightarrow b$ such that $\varphi \circ \alpha = \beta \circ M\varphi$.

Dually, one defines an **Eilenberg-Moore coalgebra** or **comonad coalgebra** for a comonad K , and a morphism of such coalgebras.

We write $\text{EM}(M)$ or $\text{EM}(K)$ for the **Eilenberg-Moore category** of a (co)monad, which is the category of Eilenberg-Moore (co)algebras for that (co)monad.

One similarly defines (co)algebras for (co)pointed endofunctors which satisfy only the law $\alpha \circ \eta = \text{id}$ or $\varepsilon \circ \alpha = \text{id}$.

Proposition 2.2.50. Every monad $M : \mathcal{C} \rightarrow \mathcal{C}$ gives rise to functors $F_M : \mathcal{C} \rightarrow \text{EM}(M) : x \mapsto (Mx, \mu_x)$ and $U_M : \text{EM}(M) \rightarrow \mathcal{C} : (a, \alpha) \mapsto a$ and we have $F_M \dashv U_M$ giving rise again to the monad $M = U_M F_M$.

Dually, every comonad K gives rise to $U_K \dashv C_K$ over $\text{EM}(K)$.

Proof. Functoriality is clear. To see adjointness, we have unit $\eta : \text{Id} \rightarrow M$ and co-unit $\varepsilon_{(a, \alpha)} = \alpha : (Ma, \mu) \rightarrow (a, \alpha)$, which is a morphism thanks to one algebra law. We have

$$U_M \varepsilon_{(a, \alpha)} \circ \eta_{U_M(a, \alpha)} = \alpha \circ \eta_a = \text{id}_a : a \rightarrow a$$

$$\varepsilon_{F_M x} \circ F_M \eta_x = \mu_x \circ \eta_x = \text{id}_x : (Mx, \mu) \rightarrow (Mx, \mu).$$

These functors clearly compose to M with the same unit. For the multiplication, we find $U_M \varepsilon_{F_m x} = \mu_x$. □

Proposition 2.2.51. For any monad M , the factorization $M = U_M F_M$ with co-unit ε_M over the Eilenberg-Moore category $\text{EM}(M)$ is the final adjoint factorization: any other adjoint factorization $M = UF$ with co-unit ε over a category \mathcal{D} yields a unique commutative diagram:

$$\begin{array}{ccc}
 \mathcal{D} & \xrightarrow{H} & \text{EM}(M) \\
 \uparrow F & \nearrow U & \downarrow U_M \\
 F_M \uparrow & & \\
 \mathcal{C} & \xrightarrow{M} & \mathcal{C}
 \end{array}$$

such that $H\varepsilon = \varepsilon_M H : F_M U \rightarrow H$.

Proof. We first show existence. Let $Hd = (Ud, U\varepsilon_d)$. This is an Eilenberg-Moore algebra, because $U\varepsilon_d \circ \eta_{Ud} = \text{id}$ and

$$U\varepsilon_d \circ MU\varepsilon_d = U\varepsilon_d \circ UFU\varepsilon_d = U\varepsilon_d \circ U\varepsilon_{FUd} = U\varepsilon_d \circ \mu_{Ud},$$

where in the second step we swapped the operands using naturality. Then $U_M Hd = Ud$ and $HFd = (UFd, U\varepsilon_{Fd}) = (Md, \mu_d) = F_M d$ as required. We have $(\varepsilon_M)_{Hd} = (\varepsilon_M)_{(Ud, U\varepsilon_d)} = U\varepsilon_d = H\varepsilon_d$, so the co-unit law is also satisfied.

To see uniqueness, note that the carrier of Hd is predetermined since $U_M H = U$, and that the structure of Hd can be obtained by taking $U_M(\varepsilon_M)_{Hd} = U_M H\varepsilon_d = U\varepsilon_d$. \square

Example 2.2.52. The functor $\text{List} : \text{Set} \rightarrow \text{Set}$ which maps a set X to the set $\text{List } X$ of finite lists of elements of X , is a monad with unit $\eta : X \rightarrow \text{List } X : x \mapsto [x]$ and concatenation as multiplication. The category $\mathbf{EM}(\text{List})$ is exactly the category of monoids, since on an Eilenberg-Moore algebra (X, ξ) we can define

$$e := \xi([\]), \quad x * y := \xi([x, y]).$$

Then we have

$$\begin{aligned} e * x &= \xi([e, x]) = (\xi \circ \text{List } \xi)(([\], [x])) = (\xi \circ \mu)(([\], [x])) = \xi([x]) = x, \\ x * (y * z) &= \xi([x, y * z]) = (\xi \circ \text{List } \xi)(([x], [y, z])) = (\xi \circ \mu)(([x], [y, z])) \\ &= \xi([x, y, z]) = \dots = (x * y) * z. \end{aligned}$$

The former proves one unit law for the monoid (the other is proven similarly) and the latter proves associativity.

The functor F_{List} sends a set X to the free monoid $(\text{List } X, \mu_X)$ over X .

Free Monads

Proposition 2.2.53. [nLa20h] Assume the category \mathcal{C} has small colimits. Let (F, η) be a pointed endofunctor on \mathcal{C} . Then there exists a free monad $(F^\infty, \eta^\infty, \mu^\infty)$ over (F, η) where $F^\infty x$ is obtained as the colimit of the following commutative diagram:

$$\begin{array}{ccccccc} x & \xrightarrow{\eta_1} & F^{(1)}x & \xrightarrow{\eta_2} & F^{(2)}x & \xrightarrow{\eta_3} & F^{(3)}x & \xrightarrow{\eta_4} & \dots \\ \eta \downarrow & \nearrow \tau_1 & \eta \downarrow & \nearrow \tau_2 & \eta \downarrow & \nearrow \tau_3 & \eta \downarrow & \nearrow \tau_4 & \\ Fx & \xrightarrow{F\eta_1} & FF^{(1)}x & \xrightarrow{F\eta_2} & FF^{(2)}x & \xrightarrow{F\eta_3} & FF^{(3)}x & \xrightarrow{F\eta_4} & \dots \end{array}$$

where $\eta_i = \tau_i \circ \eta$ and τ_1 is the identity and each further τ_{i+1} is defined as the equalizer of $F\eta_i$ and $\tau_i \circ \eta$.

The idea is that $F^{(i)}$ is a quotient of F^i which identifies $\eta F^i, F\eta F^{i-1}, \dots, F^i \eta$ into a single morphism $\eta_{i+1} : F^{(i)} \rightarrow F^{(i+1)}$.

Proof. Write $\alpha_i : F^{(i)} \rightarrow F^\infty$ and $\beta_i : FF^{(i)} \rightarrow F^\infty$. Clearly, the canonical natural transformation $F \rightarrow F^\infty$ will be given by β_0 and the unit is $\eta^\infty = \alpha_0$. It is not hard to see that there will be a general morphism $F^{(i)}F^{(j)} \rightarrow F^{(i+j)}$ and this way, we can define μ^∞ by unfolding the double colimit, coercing the obtained diagram into the one above and refolding. The effect is that $\mu^\infty \circ (\alpha_1 \star \alpha_1) = \alpha_2$.

Let (M, η', μ') be a monad and $\nu : F \rightarrow M$ a morphism of pointed functors, i.e. a natural transformation such that $\eta' \circ \nu = \nu \circ \eta$. We need to prove that there is a unique monad morphism $\nu^\infty : F^\infty \rightarrow M$ such that $\nu^\infty \circ \beta_0 = \nu$.

By the monad laws, there is a canonical natural transformation $\mu'_i : M^i \rightarrow M$. If ν^∞ is to be a monad morphism, we require

$$\nu^\infty \circ \alpha_i = \nu^\infty \circ \mu_i^\infty \circ \alpha_1^{\star i} = \mu' \circ (\nu^\infty \circ \alpha_1)^{\star i} = \mu' \circ \nu^{\star i},$$

which uniquely determines the transformation ν^∞ by the universal property of the colimit. □

Idempotent (Co)monads

Definition 2.2.54. A monad is called idempotent if μ is invertible. A comonad is called idempotent if δ is invertible.

Proposition 2.2.55. The following conditions are equivalent: M is idempotent, μ is invertible, ηM is invertible, $M\eta$ is invertible. For an idempotent monad, $M\eta = \eta M$. The dual also holds.

Proof. The left/right inverse to an isomorphism is automatically the unique inverse. So $\mu \circ \eta M = \mu \circ M\eta = \text{id}$ imply that all or none are inverse and if all are inverse, then $M\eta = \eta M = \mu^{-1}$. □

Proposition 2.2.56. An Eilenberg-Moore algebra for an idempotent monad M is an object x so that $\eta : x \rightarrow Mx$ is invertible. An algebra morphism is any morphism between algebras. The dual also holds.

Proof. If x is an Eilenberg-Moore algebra, then we have $\xi : Mx \rightarrow x$ which satisfies $\xi \circ \eta = \text{id}$. Conversely, we have $\eta \circ \xi = M\xi \circ \eta M = M\xi \circ M\eta = \text{id}$, so $\xi = \eta^{-1}$. Then naturality of η implies that any morphism is an algebra morphism.

If $\eta : x \rightarrow Mx$ is invertible, then (x, η^{-1}) satisfies the monad laws. \square

2.2.12 Subobject Classifier

Definition 2.2.57. A **subobject** of an object $c \in \mathcal{C}$ is an object b paired with a monomorphism $b \hookrightarrow c$.

A subobject classifier of a category \mathcal{C} is an object $\mathbf{Prop} \in \mathcal{C}$ (often denoted Ω) for which there exists a correspondence between morphisms $c \rightarrow \mathbf{Prop}$ and subobjects of c . Thinking of objects as some sort of collections, the idea is that \mathbf{Prop} is the collection of truth values, and the morphism $c \rightarrow \mathbf{Prop}$ sends those elements to ‘true’ that are in the image of the corresponding subobject $b \hookrightarrow c$.

Definition 2.2.58. A **subobject classifier** [Law70] of a category \mathcal{C} with finite limits, is an object $\mathbf{Prop} \in \mathcal{C}$ equipped with a morphism $\top : \top \rightarrow \mathbf{Prop}$ from the terminal object (and therefore mono), such that any subobject $\iota : b \hookrightarrow c$ of any $c \in \mathcal{C}$ is a pullback of \top along a unique morphism $(\in b) : c \rightarrow \mathbf{Prop}$.³ Clearly, b can be retrieved from $(\in b)$ up to isomorphism by actually taking the pullback.

$$\begin{array}{ccc}
 b & \xhookrightarrow{\iota} & c \\
 \downarrow & \lrcorner & \downarrow (\in b) \\
 \top & \xhookrightarrow[\top]{} & \mathbf{Prop}.
 \end{array}$$

2.2.13 Higher Category Theory

As this thesis contains barely any formal higher categorical content, we also keep the concepts informal.

Definition 2.2.59. A **2-category** is a category in which the Hom-sets are not sets but categories, and where composition is a functor. The objects of the Hom-categories are called **1-arrows**, **1-cells**, **1-morphisms** or just **morphisms**,

³We are assuming that ι is obvious from the context.

and the morphisms of the Hom-categories are called **2-arrows**, **2-cells** or **2-morphisms**. A **strict** 2-category satisfies unit and composition laws on the nose, whereas a **weak** 2-category satisfies them up to coherent isomorphism.

Example 2.2.60. An example is the strict 2-category \mathbf{Cat} whose objects are categories, and whose Hom-categories are functor categories. Hence, its 1-cells are functors and its 2-cells are natural transformations.

Example 2.2.61. A **profunctor** $\mathcal{P} : \mathcal{C} \nrightarrow \mathcal{D}$ is a functor $\mathcal{P} : \mathcal{C}^{\text{op}} \times \mathcal{D} \rightarrow \mathbf{Set}$. A morphism of profunctors is just a natural transformation.

We have a weak 2-category \mathbf{Prof} whose objects are categories, whose 1-cells are profunctors, and whose 2-cells are morphisms of profunctors. The identity is given by $\text{Hom} : \mathcal{C} \nrightarrow \mathcal{C}$, and composition is given by a co-end: $(\mathcal{Q} \circ \mathcal{P})(x, z) = \exists y. \mathcal{P}(x, y) \times \mathcal{Q}(y, z)$. The co-Yoneda lemma asserts that unit laws are respected up to isomorphism, whereas associativity is straightforwardly proven. Coherence of the isomorphisms involved is daunting to even state, let alone prove.

Definition 2.2.62. A **2-functor** $F : \mathcal{C} \rightarrow \mathcal{D}$ between 2-categories is like a functor but acts functorially on Hom-categories. A **strict** 2-functor preserves identity and composition on the nose, whereas a **weak** 2-functor (also called **pseudofunctor**) preserves them up to coherent isomorphism.

Definition 2.2.63. A **pseudonatural transformation** is a natural transformation between 2-functors that satisfies naturality up to coherent isomorphism.

Definition 2.2.64. The 2-category \mathcal{C}^{op} is the 2-category \mathcal{C} with 1-cells reversed: $\text{Hom}_{\mathcal{C}^{\text{op}}}(x, y) = \text{Hom}_{\mathcal{C}}(y, x)$.

The 2-category \mathcal{C}^{co} is the 2-category \mathcal{C} with 2-cells reversed: $\text{Hom}_{\mathcal{C}^{\text{co}}}(x, y) = \text{Hom}_{\mathcal{C}}(x, y)^{\text{op}}$.

2.3 Presheaves

2.3.1 Definition and Yoneda-embedding

Definition 2.3.1. A **presheaf** with base category \mathcal{W} is a functor $\mathcal{W}^{\text{op}} \rightarrow \mathbf{Set}$. The category of presheaves $\mathbf{Psh}(\mathcal{W})$ is the functor category $\mathbf{Set}^{\mathcal{W}^{\text{op}}}$.

Notation 2.3.2. We use the presheaf notations from earlier work [Nuy18a], concretely:

- The application of a presheaf $\Gamma \in \mathbf{Psh}(\mathcal{W})$ to an object $W \in \mathcal{W}$ is denoted $W \Rightarrow \Gamma$. An element $\gamma : W \Rightarrow \Gamma$ is called a **cell** of Γ of **shape** W .

- The **restriction** of $\gamma : W \Rightarrow \Gamma$ by $\varphi : V \rightarrow W$, i.e. the action of Γ on φ applied to γ , is denoted $\gamma \circ \varphi$ or $\gamma\varphi : V \Rightarrow \Gamma$.
- The application of a presheaf morphism $\sigma : \Gamma \rightarrow \Delta$ to $\gamma : W \Rightarrow \Gamma$ is denoted $\sigma \circ \gamma$ or $\sigma\gamma$. By naturality of σ , we have $\sigma \circ (\gamma \circ \varphi) = (\sigma \circ \gamma) \circ \varphi$.

If \mathcal{W} has a terminal object, then we call a cell $\top \Rightarrow \Gamma$ a **point**.

Definition 2.3.3. The **Yoneda-embedding** $\mathbf{y} : \mathcal{W} \rightarrow \text{Psh}(\mathcal{W})$ is defined by $(V \Rightarrow \mathbf{y}W) = (V \rightarrow W)$, i.e. $\mathbf{y}W = \text{Hom}(\sqcup, W)$. A presheaf is called **representable** if it is, up to isomorphism, in the image of the Yoneda-embedding.

Notation 2.3.4. Given a presheaf $\Gamma \in \text{Psh}(\mathcal{W})$, we denote the category of elements as $\mathcal{W}/\Gamma := \int_{W \in \mathcal{W}} (W \Rightarrow \Gamma)$.

Proposition 2.3.5. The slice category \mathcal{W}/W is the category of elements $\mathcal{W}/\mathbf{y}W$. \square

Proposition 2.3.6. All limits and colimits of presheaves of size α over diagrams of size α exist and can be taken componentwise.

Proof. For limits, let Γ be the componentwise limit of Γ_i , i.e. $(W \Rightarrow \Gamma) := \lim_{i \in \mathcal{I}} (W \Rightarrow \Gamma_i)$. Then we have:

$$\begin{aligned} (\Delta \rightarrow \Gamma) &= \forall W. (W \Rightarrow \Delta) \rightarrow (W \Rightarrow \Gamma) \\ &\cong \forall W. (W \Rightarrow \Delta) \rightarrow \lim_i (W \Rightarrow \Gamma_i) \\ &\cong \lim_i \forall W. (W \Rightarrow \Delta) \rightarrow (W \Rightarrow \Gamma_i) \\ &= \lim_i (\Delta \rightarrow \Gamma_i), \end{aligned}$$

so that $\Gamma = \lim_i \Gamma_i$ by remark 2.2.22. The move of \lim_i is justified by the explicit construction in **Set** (proposition 2.2.20).

For colimits, let Γ be the componentwise colimit of Γ_i , i.e. $(W \Rightarrow \Gamma) := \text{colim}_{i \in \mathcal{I}} (W \Rightarrow \Gamma_i)$. Then we have:

$$\begin{aligned} (\Gamma \rightarrow \Delta) &= \forall W. (W \Rightarrow \Gamma) \rightarrow (W \Rightarrow \Delta) \\ &\cong \forall W. (\text{colim}_i W \Rightarrow \Gamma_i) \rightarrow \lim_i (W \Rightarrow \Delta) \\ &\cong \lim_i \forall W. (W \Rightarrow \Gamma_i) \rightarrow (W \Rightarrow \Delta) \\ &= \lim_i (\Gamma_i \rightarrow \Delta), \end{aligned}$$

so that $\Gamma = \text{colim}_i \Gamma_i$ by remark 2.2.22. \square

2.3.2 Examples of Base and Presheaf Categories

Example 2.3.7 (Sets). Let \mathbf{Point} be the category with a single object $*$ and only the identity morphism. Then $\mathbf{Psh}(\mathbf{Point}) \cong \mathbf{Set}$, and \mathbf{y}^* is the singleton.

Reflexive Graphs

Example 2.3.8 (Reflexive graphs). A reflexive graph Γ consists of a set $\Gamma_{\mathbb{N}}$ of nodes and a set $\Gamma_{\mathbb{I}}$ of edges, with two functions $\mathbf{s}, \mathbf{t} : \Gamma_{\mathbb{I}} \rightarrow \Gamma_{\mathbb{N}}$ assigning to each edge a source and target node (we say that e is an edge from $\mathbf{s}(e)$ to $\mathbf{t}(e)$), and a function $\mathbf{r} : \Gamma_{\mathbb{N}} \rightarrow \Gamma_{\mathbb{I}}$ such that $\mathbf{s}(\mathbf{r}(x)) = \mathbf{t}(\mathbf{r}(x)) = x$ assigning to each node x a reflexive edge from x to x .

Reflexive graphs can be organized as presheaves over the category \mathbf{RG} generated by the following diagram and equations:

$$\begin{array}{ccc} & \xrightarrow{\mathbf{s}} & \\ \mathbb{N} & \xleftarrow{\mathbf{r}} & \mathbb{I} \\ & \xrightarrow{\mathbf{t}} & \end{array} \quad \begin{array}{l} \mathbf{r} \circ \mathbf{s} = 1_{\mathbb{N}}, \\ \mathbf{r} \circ \mathbf{t} = 1_{\mathbb{N}}. \end{array}$$

The idea is that $\mathbb{N} \Rightarrow \Gamma$ is the set of nodes, $\mathbb{I} \Rightarrow \Gamma$ is the set of edges, and that $(\sqsubset \circ \mathbf{s}), (\sqsubset \circ \mathbf{t}) : (\mathbb{I} \Rightarrow \Gamma) \rightarrow (\mathbb{N} \Rightarrow \Gamma)$ extract the source and target of an edge, whereas $(\sqsubset \circ \mathbf{r}) : (\mathbb{N} \Rightarrow \Gamma) \rightarrow (\mathbb{I} \Rightarrow \Gamma)$ produces the reflexive edge on a node. The equations assert that the edge $\gamma \circ \mathbf{r}$ really goes from γ to γ .

A reflexive graph morphism $\sigma : \Gamma \rightarrow \Delta$ is then a natural transformation, sending nodes $\gamma : \mathbb{N} \Rightarrow \Gamma$ to nodes $\sigma \circ \gamma : \mathbb{N} \Rightarrow \Delta$ and edges $\gamma : \mathbb{I} \Rightarrow \Gamma$ to edges $\sigma \circ \gamma : \mathbb{I} \Rightarrow \Delta$ such that the image of the source $\sigma \circ (\gamma \circ \mathbf{s})$ is the source of the image $(\sigma \circ \gamma) \circ \mathbf{s}$, and similar for target and reflexivity.

The graph $\mathbf{y}^{\mathbb{N}}$ contains a single node $\text{id}_{\mathbb{N}} : \mathbb{N} \Rightarrow \mathbf{y}^{\mathbb{N}}$ and a single reflexive edge $\mathbf{r} : \mathbb{I} \Rightarrow \mathbf{y}^{\mathbb{N}}$. The graph $\mathbf{y}^{\mathbb{I}}$ contains a single non-trivial edge $\text{id}_{\mathbb{I}}$, its source \mathbf{s} , its target \mathbf{t} and the reflexive edges $\mathbf{s} \circ \mathbf{r}$ and $\mathbf{t} \circ \mathbf{r}$:

$$\begin{array}{c} \mathbf{r} \\ \curvearrowright \\ \text{id}_{\mathbb{N}} \end{array} \quad \mathbf{s} \circ \mathbf{r} \quad \mathbf{t} \circ \mathbf{r}$$

A straightforward generalization of \mathbf{RG} , is the category ${}^a\mathbf{RG}$, which has a different arrows $\mathbb{N} \rightarrow \mathbb{I}$, so that ${}^2\mathbf{RG} \cong \mathbf{RG}$. Presheaves over ${}^a\mathbf{RG}$ are a -ary reflexive graphs, where every edge has not 2 but a endpoints.

Another possible modification is the addition of an involution $\neg : \mathbb{I} \rightarrow \mathbb{I}$ such that $\mathbf{s} \circ \neg = \mathbf{t}$ and (hence) $\mathbf{t} \circ \neg = \mathbf{s}$, yielding \mathbf{RG}_{\neg} . Presheaves over this category are undirected reflexive graphs.

Atkey, Ghani, and Johann [AGJ14] use reflexive graphs to model dependently typed parametricity.

Simplicial Sets

Example 2.3.9 (Simplicial sets). A simplicial set Γ is like a reflexive graph, containing nodes, edges with a source and a target, and a reflexive edge on every node. However, on top of that, simplicial sets contain higher dimensional structure. This begins with triangles, which have 3 vertices and three sides (and the vertices are the sources/targets of the sides), and conversely there are 2 flat triangles on every edge. Next, we have tetrahedra, which have 4 faces (these are triangles), 6 sides, and 4 nodes. In general, a simplicial set Γ comes with a set Γ_n of n -simplices (these are n -dimensional triangles/tetrahedra) for every n , as well as operations for extracting lower-dimensional faces of higher-dimensional simplices (these operations are called **face maps**), as well as for extracting flat higher-dimensional simplices from lower-dimensional ones (called **degeneracy maps**).

Simplicial sets can be organized as presheaves over the **simplex category** $\mathbf{Simplex}$, which is essentially the skeleton of the category of non-empty finite linear orders. Concretely:

- Its objects are natural numbers (including 0), we write $\text{Obj}(\mathbf{Simplex}) = \{[n] \mid n \in \mathbb{N}\}$,
- Morphisms $\varphi : \text{Hom}([m], [n])$ are non-decreasing functions

$$\varphi : \{0 \leq 1 \leq \dots \leq m\} \rightarrow \{0 \leq 1 \leq \dots \leq n\}.$$

The idea is that $[n] \Rightarrow \Gamma$ is the set of n -dimensional simplices in Γ . In particular, $[0] \Rightarrow \Gamma$ is the set of nodes and $[1] \Rightarrow \Gamma$ is the set of edges. Restriction by a map $0 \mapsto i : [0] \rightarrow [n]$ extracts the i th node of an n -simplex. Restriction by a map $\{0 \mapsto i, 1 \mapsto j\} : [1] \rightarrow [n]$ extracts the edge from the i th node to the j th node of an n -simplex. Similarly, we extract the faces of an n -simplex with maps $[2] \rightarrow [n]$. In general, injective maps are face maps.

On the other hand, surjective maps provide flat simplices. For example, there are two flat triangles on every edge, given by the maps

$$\{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 1\} : [2] \rightarrow [1], \quad \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\} : [2] \rightarrow [1].$$

We remark that there is a fully faithful functor $\mathbf{RG} \rightarrow \mathbf{Simplex}$, i.e. a reflexive graph is really the 1-dimensional truncation of a simplicial set.

The presheaf $\mathbf{y}[n]$ contains a non-trivial n -simplex $\text{id}_{[n]}$ as well as all of its faces and all degenerate simplices on all those faces.

Simplicial sets are used as a model for (higher dimensional) categories; see any reference on the subject. Kapulkin, Lumsdaine, and Voevodsky [KLV12] use simplicial sets to model homotopy type theory (HoTT) [Uni13].

Cubical Sets

Example 2.3.10 (Cartesian cubical sets). Cubical sets are another generalization of reflexive graphs. A cubical set Γ comes with, for every n , a set Γ_n , not of n -dimensional simplices but of n -dimensional cubes. There is a great deal of different notions of cubical sets, depending on the available restriction maps. For example, we may or may not allow extracting diagonals, or obtaining squares from edges by folding them open like a fan (i.e. connections).

In this first example, we consider cartesian cubical sets, which have diagonals but no connections. These can be organized as presheaves over the category of cartesian cubes \mathbf{Cube} , which is the free cartesian monoidal category with same terminal object over \mathbf{RG} . Concretely:

- Its objects take the form $(i_1 : \mathbb{I}, \dots, i_n : \mathbb{I})$ (the names are desugared to de Bruijn indices, i.e. the objects are really just natural numbers),
- Its morphisms $(i_1 : \mathbb{I}, \dots, i_n : \mathbb{I}) \rightarrow (j_1 : \mathbb{I}, \dots, j_m : \mathbb{I})$ are arbitrary functions $\varphi : \{j_1, \dots, j_m\} \rightarrow \{i_1, \dots, i_n\} \cup \{0, 1\} : j \mapsto j\langle\varphi\rangle$. We also write $\varphi = (j_1\langle\varphi\rangle/j_1, \dots, j_m\langle\varphi\rangle/j_m)$. If a variable i is not used, we may write i/\emptyset to emphasize this.

Again there is a fully faithful functor $F : \mathbf{RG} \rightarrow \mathbf{Cube}$:

$$F\mathbb{N} = (), \quad F\mathbb{I} = (i : \mathbb{I}), \quad F\mathbf{s} = (0/i), \quad F\mathbf{t} = (1/i), \quad F\mathbf{r} = (i/\emptyset).$$

In general, substitutions $0/i$ and $1/i$ extract faces of cubes, weakening i/\emptyset extracts flat $(n + 1)$ -cubes from n -cubes and contraction $(i/j, i/k) : (i : \mathbb{I}) \rightarrow (j : \mathbb{I}, k : \mathbb{I})$ extracts diagonals.

A straightforward generalization is the category of a -ary cartesian cubes ${}^a\mathbf{Cube}$, where morphisms φ send a variable j to $j\langle\varphi\rangle \in \{i_1, \dots, i_n\} \cup \{0, \dots, a - 1\}$.

In the binary case, we may additionally throw in an involution $\neg : \mathbb{I} \rightarrow \mathbb{I}$ satisfying $\neg 0 = 1$ and $\neg 1 = 0$, yielding the category \mathbf{Cube}_- .

Example 2.3.11 (Affine cubical sets). If we rule out diagonals, then we obtain affine cubical sets, which are presheaves over the category of a -ary affine cubes ${}^a\mathbf{Cube}_\square$, which is the free monoidal category with (same) terminal unit over ${}^a\mathbf{RG}$. Concretely:

- Objects are as in ${}^a\mathbf{Cube}$,

- Morphisms are as in ${}^a\text{Cube}$ such that if $j\langle\varphi\rangle = k\langle\varphi\rangle \notin \{0, \dots, a-1\}$, then $j = k$. This rules out diagonal maps.

Bezem, Coquand, and Huber [BCH14] use binary affine cubical sets to model cubical HoTT. Bernardy, Coquand, and Moulin [BCM15] and Moulin [Mou16] use unary ones to model internal unary parametricity.

Again, in the binary case, we can throw in an involution, yielding $\text{Cube}_{\square, \neg}$.

Example 2.3.12 (CCHM cubical sets). Cohen, Coquand, Huber, and Mörtberg [Coh+17] define the category CCHM of CCHM cubes, whose objects are as in Cube , and where morphisms $(i_1 : \mathbb{I}, \dots, i_n : \mathbb{I}) \rightarrow (j_1 : \mathbb{I}, \dots, j_m : \mathbb{I})$ are functions from $\{j_1, \dots, j_m\}$ to the free de Morgan algebra over $\{i_1, \dots, i_n\}$. This means in particular that there is an involution $\neg : \mathbb{I} \rightarrow \mathbb{I}$ as well as two operations $\wedge, \vee : \mathbb{I}^2 \rightarrow \mathbb{I}$, called **connections**, such that

$$\begin{aligned} 0 \vee i &= i \vee 0 = i, & 0 \wedge i &= i \wedge 0 = 0, \\ 1 \vee i &= i \vee 1 = 1, & 1 \wedge i &= i \wedge 1 = i. \end{aligned}$$

Restriction by a connection folds open an edge $e : (i : \mathbb{I}) \Rightarrow \Gamma$ from $x = e(0/i)$ to $y = e(1/i)$ like a fan, yielding a square:

$$\begin{array}{ccc} & y \xrightarrow{y(k/\circ)} y & x \xrightarrow{e(k/i)} y \\ \begin{array}{c} \uparrow \\ j \\ \downarrow \end{array} & e(j/i) \left| \begin{array}{c} e(j \vee k/i) \\ y(j/\circ) \end{array} \right| & x(j/\circ) \left| \begin{array}{c} e(j \wedge k/i) \\ e(j/i) \end{array} \right| \\ \xrightarrow{k} & x \xrightarrow{e(k/i)} y & x \xrightarrow{x(k/\circ)} x \end{array}$$

Example 2.3.13 (Depth d cubical sets). Let DCube_d with $d \geq -1$ be the category of depth d cubes, used as a base category in degrees of relatedness [ND18a; Nuy18a] (chapter 9). This is a generalization of the category of binary cartesian cubes Cube , where instead of typing every dimension with *the* interval \mathbb{I} , we type them with the k -interval $\langle k \rangle$, where $k \in \{0, \dots, d\}$ is called the degree of relatedness of the edge. Its objects take the form $(i_1 : \langle k_1 \rangle, \dots, i_n : \langle k_n \rangle)$. Conceptually, we have a map $\langle k \rangle \rightarrow \langle \ell \rangle$ if $k \geq \ell$. Thus, morphisms $\varphi : (i_1 : \langle k_1 \rangle, \dots, i_n : \langle k_n \rangle) \rightarrow (j_1 : \langle \ell_1 \rangle, \dots, j_m : \langle \ell_m \rangle)$ send every variable $j : \langle \ell \rangle$ of the codomain to $j\langle\varphi\rangle$, which is either 0, 1 or a variable $i : \langle k \rangle$ of the domain such that $k \geq \ell$.

- If $d = -1$, then there is only one object $()$ and only the identity morphism, i.e. we have the point category.
- If $d = 0$, we just get Cube .

- If $d = 1$, we obtain the category of bridge/path cubes $\text{BPCube} := \text{DCube}_1$. We write \mathbb{P} for $\langle 0 \rangle$ (the path interval) and \mathbb{B} for $\langle 1 \rangle$ (the bridge interval). Bridge/path cubical sets are used as a model for parametric quantifiers [NVD17a; Nuy17] (chapter 9).

Indices, Trees and Clocks

Example 2.3.14 (Topos of trees). The ordinal number ω can be seen as a category:

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \dots$$

When $i \leq j$, we write \downarrow_i^j for the unique morphism $\downarrow_i^j : i \rightarrow j$. The category $\text{Psh}(\omega)$ is called the **topos of trees** and is used, among other things, as a model of guarded type theory [Bir+12]. The name ‘topos of trees’ is derived from a visual representation of a presheaf $\Gamma \in \text{Psh}(\omega)$: if one lists on the i th row of a sheet of paper, counting bottom-up, the elements of $i \Rightarrow \Gamma$, and then connects every $\gamma : i + 1 \Rightarrow \Gamma$ to its unique restriction in $i \Rightarrow \Gamma$, zero or more trees form.

The presheaf $\mathbf{y}k$ contains a single i -cell $i \Rightarrow \mathbf{y}k$ if $i \leq k$, and no i -cells otherwise.

Example 2.3.15 (Sierpiński topos). Of course we can also consider presheaf categories over other ordinals. In particular, the **Sierpiński topos** is the presheaf category $\text{Psh}(2)$ over $2 = \{0 \rightarrow 1\}$. It is, essentially, the arrow category of Set .

Example 2.3.16 (Clocks). Let Clock be the category of clocks, used as a base category in guarded type theory [BM18]. It is the free cartesian category over ω . Concretely:

- Its objects take the form $(i_1 : \oplus_{k_1}, \dots, i_n : \oplus_{k_n})$ where all $k_j \geq 0$. We can think of a variable of type \oplus_k as representing a clock (i.e. a time dimension) paired up with a certificate that we do not care what happens after the time on this clock exceeds k .
- Correspondingly, we should have a map $\oplus_k \rightarrow \oplus_\ell$ if $k \leq \ell$. So the morphisms $\varphi : (i_1 : \oplus_{k_1}, \dots, i_n : \oplus_{k_n}) \rightarrow (j_1 : \oplus_{\ell_1}, \dots, j_m : \oplus_{\ell_m})$ are functions that send (de Bruijn) variables $j : \oplus_\ell$ of the codomain to a variable $j\langle\varphi\rangle : \oplus_k$ of the domain such that $k \leq \ell$.

2.3.3 The Yoneda and Co-Yoneda Lemmas

Theorem 2.3.17 (Yoneda lemma). For any presheaf $\Gamma \in \text{Psh}(\mathcal{W})$, we have

$$\forall W. (\text{Hom}_{\mathcal{W}}(W, W_0) \rightarrow (W \Rightarrow \Gamma)) \cong (W_0 \Rightarrow \Gamma),$$

naturally in W_0 . We remark that the end on the left (constructed as in proposition 2.2.27) is exactly the set of presheaf morphisms $\mathbf{y}W_0 \rightarrow \Gamma$. Hence $(\mathbf{y}W_0 \rightarrow \Gamma) \cong (W_0 \rightrightarrows \Gamma)$.

Notation 2.3.18. We will omit applications of this isomorphism $(\mathbf{y}W \rightarrow \Gamma) \cong (W \rightrightarrows \Gamma)$, i.e. if $\gamma : W \rightrightarrows \Gamma$, then we also use $\gamma : \mathbf{y}W \rightarrow \Gamma$.

Proof. Pick $\sigma : \mathbf{y}W_0 \rightarrow \Gamma$. We claim that σ is entirely determined by $\sigma \circ \text{id}_{W_0} : W_0 \rightrightarrows \Gamma$. Indeed:

$$\sigma \circ \psi = \sigma \circ (\text{id}_{W_0} \circ \psi) = (\sigma \circ \text{id}_{W_0}) \circ \psi,$$

i.e. $\sigma \circ \psi$ (the application of σ to ψ) is found by restricting $\sigma \circ \text{id}_{W_0}$ (the application of σ to id_{W_0}) by ψ . Hence, the assignment $\sigma \mapsto \sigma \circ \text{id}_{W_0}$ is invertible.

It is also natural in W_0 . Indeed, pick $\chi : W_0 \rightarrow W_1$. Then $\sigma \circ \mathbf{y}\chi$ is sent to

$$(\sigma \circ \mathbf{y}\chi) \circ \text{id}_{W_1} = \sigma \circ (\chi \circ \text{id}_{W_1}) = \sigma \circ (\text{id}_{W_0} \circ \chi) = (\sigma \circ \text{id}_{W_0}) \circ \chi. \quad \square$$

Theorem 2.3.19 (Co-Yoneda lemma). For any presheaf $\Gamma \in \mathbf{Psh}(\mathcal{W})$, we have

$$\exists W. (\text{Hom}_{\mathcal{I}}(W_0, W) \times (W \rightrightarrows \Gamma)) \cong (W_0 \rightrightarrows \Gamma),$$

naturally in W_0 .

Proof. Construct the co-end as in proposition 2.2.28. In one direction, we send (W, ψ, γ) to $\gamma \circ \psi$ (which is clearly natural in W_0); in the other, we send γ_0 to $(W_0, \text{id}, \gamma_0)$. One side of the isomorphism is trivial. For the other, we have to show that $(W, \psi, \gamma) = (W_0, \text{id}, \gamma \circ \psi)$, but this follows from the equality relation on the co-end. \square

2.3.4 Dependent Yoneda and Co-Yoneda Lemmas

The following theorems are neither standard nor very surprising.

Theorem 2.3.20 (Dependent Yoneda lemma). For any presheaves $\Gamma \in \mathbf{Psh}(\mathcal{W})$ and $T \in \mathbf{Psh}(\mathcal{W}/\Gamma)$, we have

$$\forall W. ((\psi : W \rightarrow W_0) \rightarrow ((W, \gamma_0 \circ \psi) \rightrightarrows T)) \cong ((W_0, \gamma_0) \rightrightarrows T),$$

naturally in $(W_0, \gamma_0) \in \mathcal{W}/\Gamma$.

Proof. We have

$$\begin{aligned}
& \forall W.((\psi : W \rightarrow W_0) \rightarrow ((W, \gamma_0 \circ \psi) \Rightarrow T)) \\
& \cong \forall (W, \gamma).(\psi : \text{Hom}_{\mathcal{W}/\Gamma}((W, \gamma), (W_0, \gamma_0)) \rightarrow ((W, \gamma) \Rightarrow T)) \\
& \cong \forall W.(\gamma : W \Rightarrow \Gamma) \rightarrow (\psi : \text{Hom}_{\mathcal{W}/\Gamma}((W, \gamma), (W_0, \gamma_0)) \rightarrow ((W, \gamma) \Rightarrow T)) \\
& \cong ((W_0, \gamma_0) \Rightarrow T).
\end{aligned}$$

In the first step, we add an additional argument γ but constrain it to be $\gamma = \gamma_0 \circ \psi$ by asking that ψ be a morphism of slices. The second step is a matter of reorganizing the naturality condition. The third step is the Yoneda lemma. \square

Notation 2.3.21. For any logical formula P , when we write P as a set, we mean the subsingleton $\{\star \mid P\}$.

Theorem 2.3.22 (Dependent co-Yoneda lemma). For any presheaves $\Gamma \in \text{Psh}(\mathcal{W})$ and $T \in \text{Psh}(\mathcal{W}/\Gamma)$, we have

$$\begin{aligned}
& \exists W.(\psi : W_0 \rightarrow W) \times (\gamma : W \Rightarrow \Gamma) \times ((W, \gamma) \Rightarrow T) \times (\gamma \circ \psi = \gamma_0) \\
& \cong ((W_0, \gamma_0) \Rightarrow T)
\end{aligned}$$

naturally in $(W_0, \gamma_0) \in \mathcal{W}/\Gamma$.

Proof. We have

$$\begin{aligned}
& \exists W.(\psi : W_0 \rightarrow W) \times (\gamma : W \Rightarrow \Gamma) \times ((W, \gamma) \Rightarrow T) \times (\gamma \circ \psi = \gamma_0) \\
& \cong \exists (W, \gamma).(\psi : W_0 \rightarrow W) \times ((W, \gamma) \Rightarrow T) \times (\gamma \circ \psi = \gamma_0) \\
& \cong \exists (W, \gamma).(\psi : (W_0, \gamma_0) \rightarrow (W, \gamma)) \times ((W, \gamma) \Rightarrow T) \\
& \cong ((W_0, \gamma_0) \Rightarrow T).
\end{aligned}$$

The first step is a matter of comparing equivalence relations. In the second step, we use the equality proof to reframe ψ as a morphism of slices. The last step is the co-Yoneda lemma. \square

2.3.5 Injective and Surjective Morphisms

Proposition 2.3.23. A morphism of presheaves $\sigma : \Gamma \rightarrow \Delta$ is:

- mono if and only if every component is injective, in which case we call σ **injective**,
- epi if and only if every component is surjective, in which case we call σ **surjective**.

Proof. For mono-/epimorphisms $\sigma : \Gamma \rightarrow \Delta$, the following function is injective/surjective:

$$(W \rightrightarrows \Gamma) \cong (\mathbf{y}W \rightarrow \Gamma) \rightarrow (\mathbf{y}W \rightarrow \Delta) \cong (W \rightrightarrows \Delta),$$

which shows that σ is injective/surjective.

Next, we show that if σ is surjective, then it is epi. Pick $\tau_1, \tau_2 : \Delta \rightarrow \Theta$ such that $\tau_1 \circ \sigma = \tau_2 \circ \sigma$. We show that $\tau_1 = \tau_2$. Pick $\delta : W \rightrightarrows \Delta$. By surjectivity, there is some $\gamma : W \rightrightarrows \Gamma$ such that $\delta = \sigma \circ \gamma$. Then $\tau_1 \circ \delta = \tau_2 \circ \delta$.

Finally, we show that if σ is injective, then it is mono. Pick $\rho_1, \rho_2 : \Theta \rightarrow \Gamma$ such that $\sigma \circ \rho_1 = \sigma \circ \rho_2$. We show that $\rho_1 = \rho_2$. Pick $\theta : W \rightrightarrows \Theta$. Then $\sigma \circ \rho_1 \circ \theta = \sigma \circ \rho_2 \circ \theta$, so by injectivity $\rho_1 \circ \theta = \rho_2 \circ \theta$. \square

Corollary 2.3.24. Given $\varphi : V \rightarrow W$, the morphism $\mathbf{y}\varphi : \mathbf{y}V \rightarrow \mathbf{y}W$ is

- mono if and only if φ is mono,
- epi if and only if φ is split epi. \square

2.3.6 Subobject Classifier

Proposition 2.3.25. All presheaf categories have a subobject classifier (definition 2.2.58).

Proof. In order to define $\mathbf{Prop} \in \mathbf{Psh}(\mathcal{W})$, we need to define $W \rightrightarrows \mathbf{Prop}$ for all $W \in \mathcal{W}$. But by the Yoneda-lemma, this is isomorphic to $(\mathbf{y}W \rightarrow \mathbf{Prop})$, which should be the set of isomorphism classes of subobjects of $\mathbf{y}W$. Now, since on presheaves we actually have a notion of subpresheaf $\Delta \subseteq \Gamma$ (meaning that $(W \rightrightarrows \Delta) \subseteq (W \rightrightarrows \Gamma)$ naturally in W), we can use these subpresheaves as canonical representants of their isomorphism class, and we can just define (notation 2.0.1)

$$(W \rightrightarrows \mathbf{Prop}) := \{(\in \Upsilon) \mid \Upsilon \subseteq \mathbf{y}W\}. \quad (2.6)$$

Note that Υ is obtained from $(\in \Upsilon)$ by taking the pullback of $\top : \top \rightarrow \mathbf{Prop}$ along $(\in \Upsilon)$.

Restriction by $\varphi : V \rightarrow W$ yields a morphism $(\in \Upsilon) \circ \varphi : \mathbf{y}V \rightarrow \mathbf{Prop}$ which classifies a similar pullback, which is also the pullback (preimage) $(\mathbf{y}\varphi)^{-1}(\Upsilon)$ of Υ along $\mathbf{y}\varphi$:

$$\begin{array}{ccc}
 (\mathbf{y}\varphi)^{-1}(\Upsilon) & \longrightarrow & \Upsilon \\
 \downarrow \subseteq & & \downarrow \subseteq \\
 \mathbf{y}V & \xrightarrow{\mathbf{y}\varphi} & \mathbf{y}W.
 \end{array}$$

Hence, we *must* define restriction by $(\in \Upsilon) \circ \varphi := (\in (\mathbf{y}\varphi)^{-1}(\Upsilon))$.

The morphism $\top : \top \rightarrow \mathbf{Prop}$ sends any $() : W \Rightarrow \top$ to $\top \circ () := \mathbf{y}W \subseteq \mathbf{y}W$.

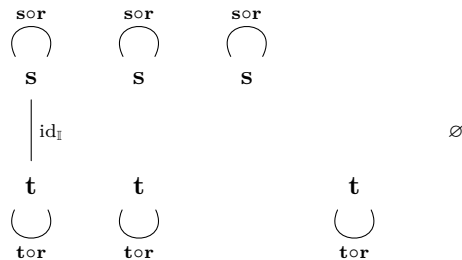
Next, given a subobject (without loss of generality, a subpresheaf) $\Gamma' \subseteq \Gamma$, we need to define the corresponding morphism $(\in \Gamma') : \Gamma \rightarrow \mathbf{Prop}$, i.e. given $\gamma : W \Rightarrow \Gamma$, we need to define $(\in \Gamma') \circ \gamma : W \Rightarrow \mathbf{Prop}$. This is again a map $(\in \Gamma') \circ \gamma : \mathbf{y}W \rightarrow \mathbf{Prop}$, and it classifies the pullback of \top along $(\in \Gamma') \circ \gamma$, which is the pullback of Γ' along $\gamma : \mathbf{y}W \rightarrow \Gamma$, i.e. the preimage $\gamma^{-1}(\Gamma')$. Thus, we *must* define $(\in \Gamma') \circ \gamma := (\in \gamma^{-1}(\Gamma'))$. This clearly respects restriction.

The pullback $\Gamma'' \subseteq \Gamma$ of $\top : \top \rightarrow \mathbf{Prop}$ along $(\in \Gamma') : \Gamma \rightarrow \mathbf{Prop}$ is now the subpresheaf of cells $\gamma : W \Rightarrow \Gamma$ such that $(\in \Gamma') \circ \gamma = (\in \gamma^{-1}(\Gamma')) : W \Rightarrow \mathbf{Prop}$ is the total presheaf $(\in \mathbf{y}W)$, i.e. such that the image of $\gamma : \mathbf{y}W \rightarrow \Gamma$ is entirely in Γ' , i.e. such that γ is a cell of Γ' . In summary, $\Gamma'' = \Gamma'$.

Most authors instead define $W \Rightarrow \mathbf{Prop}$ to be the set of *sieves* on W , but a sieve could be defined to be any full subcategory of $\mathcal{W}/W \cong \mathcal{W}/\mathbf{y}W$ that is the category of elements of a subpresheaf of $\mathbf{y}W$. □

Example 2.3.26 (Sets). Continuing example 2.3.7, the subobject classifier in the category of sets $\mathbf{Psh}(\mathbf{Point})$ is the set of subsets of $\mathbf{y}*$, the singleton. Hence, it contains two elements: $(\in \mathbf{y}*)$ (truth) and the empty presheaf (falsehood).

Example 2.3.27 (Reflexive graphs). Continuing example 2.3.8, the subobject classifier of $\mathbf{Psh}(\mathbf{RG})$ has as its nodes $\mathbb{N} \Rightarrow \mathbf{Prop}$ the subgraphs of $\mathbf{y}\mathbb{N}$, namely $(\in \mathbf{y}\mathbb{N})$ (truth \top) and the empty presheaf (falsehood \perp). The edges $\mathbb{I} \Rightarrow \mathbf{Prop}$ are all 5 subpresheaves of $\mathbf{y}\mathbb{I}$:



The first is truth \top , the reflexive edge from \top to \top ; the last is falsehood \perp , the reflexive edge from \perp to \perp . The other three are non-reflexive edges from \top to \top , \top to \perp and \perp to \top (resp.).

Example 2.3.28 (Topos of trees). Continuing example 2.3.14, the subobject classifier of the topos of trees $\mathbf{Psh}(\omega)$ has as its k -cells $k \Rightarrow \mathbf{Prop}$ all subpresheaves of $\mathbf{y}k$, which are the empty presheaf and all $(\in \mathbf{y}i)$ for $i \leq k$. The restriction of $(\in \mathbf{y}i) : k \Rightarrow \mathbf{Prop}$ to $j \Rightarrow \mathbf{Prop}$ is $(\in \mathbf{y} \min(i, j))$.

2.3.7 Base Pullbacks and Representable Morphisms

The following notion is neither standard nor very remarkable:

Definition 2.3.29. Given presheaves $\sigma : \Delta \rightarrow \Gamma : \mathbf{Psh}(\mathcal{W})$ and $\gamma : W \Rightarrow \Gamma$, a **base pullback** $\Delta \times_{\Gamma} W$ is a final object making the following diagram commute:

$$\begin{array}{ccc} \Delta \times_{\Gamma} W & \dashrightarrow & W \\ \downarrow \lrcorner & & \Downarrow \gamma \\ \Delta & \xrightarrow{\sigma} & \Gamma. \end{array}$$

In other words, we have a natural isomorphism of sets

$$(V \rightarrow \Delta \times_{\Gamma} W) \cong (V \Rightarrow \Delta) \times_{(V \Rightarrow \Gamma)} (V \rightarrow W). \quad (2.7)$$

Proposition 2.3.30. The Yoneda-embedding of a base pullback is a pullback: $\mathbf{y}(\Delta \times_{\Gamma} W) \cong \Delta \times_{\Gamma} \mathbf{y}W$.

Proof. We have

$$\begin{aligned} (V \Rightarrow \mathbf{y}(\Delta \times_{\Gamma} W)) &\cong (V \Rightarrow \Delta) \times_{(V \Rightarrow \Gamma)} \times (V \Rightarrow \mathbf{y}W) \\ &= (V \Rightarrow \Delta) \times_{(V \Rightarrow \Gamma)} \mathbf{Hom}(V, W) \\ &\cong (V \rightarrow \Delta \times_{\Gamma} W). \quad \square \end{aligned}$$

Definition 2.3.31. A morphism of presheaves $\sigma : \Delta \rightarrow \Gamma$ is **representable** if all base pullbacks along σ exist. [Sta23]

2.3.8 Lifting Functors

Theorem 2.3.32. Any functor $F : \mathcal{V} \rightarrow \mathcal{W}$ gives rise to functors $F_! \dashv F^* \dashv F_*$, with a natural isomorphism $F_! \circ \mathbf{y} \cong \mathbf{y} \circ F : \mathcal{V} \rightarrow \widehat{\mathcal{W}}$. We will call $F_! : \widehat{\mathcal{V}} \rightarrow \widehat{\mathcal{W}}$ the **left lifting** of F to presheaves, $F^* : \widehat{\mathcal{W}} \rightarrow \widehat{\mathcal{V}}$ the **central** and $F_* : \widehat{\mathcal{V}} \rightarrow \widehat{\mathcal{W}}$ the **right lifting**.⁴ [StaVC]

The operation $\sqsubset^* : \mathbf{Cat}^{\text{coop}} \rightarrow \mathbf{Cat}$ sending $F : \mathcal{V} \rightarrow \mathcal{W}$ to $F^* : \mathbf{Psh}(\mathcal{W}) \rightarrow \mathbf{Psh}(\mathcal{V})$ is a strict 2-functor. Hence $\sqsubset_!, \sqsubset_* : \mathbf{Cat} \rightarrow \mathbf{Cat}$ are pseudofunctors.

Proof. Using quantifier symbols for ends and co-ends, we can define:

$$\begin{aligned} W \Rightarrow F_! \Gamma &:= \exists V. (W \rightarrow FV) \times (V \Rightarrow \Gamma), \\ V \Rightarrow F^* \Delta &:= FV \Rightarrow \Delta \\ W \Rightarrow F_* \Gamma &:= \forall V. (FV \rightarrow W) \rightarrow (V \Rightarrow \Gamma) = (F^* \mathbf{y}W \rightarrow \Gamma). \end{aligned}$$

By the co-Yoneda lemma, we have, naturally in W :

$$\begin{aligned} W \Rightarrow F_! \mathbf{y}V &= \exists V'. (W \rightarrow FV') \times (V' \rightarrow V) \\ &\cong (W \rightarrow FV) = (W \Rightarrow \mathbf{y}FV), \end{aligned}$$

i.e. $F_! \mathbf{y}V \cong \mathbf{y}FV$.

Adjointness also follows from applications of the Yoneda and co-Yoneda lemmas. [StaVC]

It is evident from the definition of \sqsubset^* that it preserves identity and composition on the nose, and easy to check that it turns around natural transformations. By uniqueness of the left/right adjoint (proposition 2.2.44), $\sqsubset_!$ and \sqsubset_* are then pseudofunctors. \square

Notation 2.3.33. • We denote the cell $(V, \varphi, \gamma) : W \Rightarrow F_! \Gamma$ as $F_! \gamma \circ \varphi$. If we rename $F_!$, then we will also do so in this notation. We will further abbreviate $F_! \gamma \circ \text{id} = F_! \gamma$ and, if $\Gamma = \mathbf{y}V$, also $F_! \text{id} \circ \varphi = \varphi$.

- We denote the transpositions $\mathbf{A}_F : F_! \dashv F^*$ and $\mathbf{B}_F : F^* \dashv F_*$.
- If $\delta : FV \Rightarrow \Delta$, then we write $\mathbf{A}_F(\delta) : V \Rightarrow F^* \Delta$.
- If $\gamma : F^* \mathbf{y}W \Rightarrow \Gamma$, then we write $\mathbf{B}_F(\gamma) : W \Rightarrow F_* \Gamma$.

⁴The central and right liftings are also sometimes called the inverse image and direct image of F , but these are actually more general concepts and as such could perhaps cause confusion or unwanted connotations in some circumstances. The left-central-right terminology is very no-nonsense.

Remark 2.3.34. We have noted that $\sqcup_!$, \sqcup^* and \sqcup_* are at least pseudofunctors on \mathbf{Cat} . Hence, they have an action on natural transformations. The actions of the images of a natural transformation $\nu : F \rightarrow G : \mathcal{V} \rightarrow \mathcal{W}$ are given by

$$\begin{aligned} \nu_! \circ F_!(\gamma^{V \Rightarrow \Gamma}) \circ \psi^{W \rightarrow FV} &= G_! \gamma \circ \nu \circ \psi && : W \Rightarrow G_! \Gamma, \\ \nu^* \circ A_G(\delta^{GV \Rightarrow \Delta}) &= A_F(\delta \circ \nu) && : V \Rightarrow F^* \Delta, \\ \nu_* \circ B_F(\sigma^{F^* \mathbf{y}W \rightarrow \Gamma}) &= B_G(\sigma \circ \nu^*) && : W \Rightarrow G_* \Gamma. \end{aligned}$$

Remark 2.3.35. An adjunction $L \dashv R$ between base categories, leads to a chain

$$L_! \dashv R_! \cong L^* \dashv R^* \cong L_* \dashv R_*.$$

The three adjunctions have units

$$\eta_! : \text{Id} \rightarrow R_! L_!, \quad \varepsilon^* : \text{Id} \rightarrow R^* L^*, \quad \eta_* : \text{Id} \rightarrow R_* L_*,$$

and co-units

$$\varepsilon_! : L_! R_! \rightarrow \text{Id}, \quad \eta^* : L^* R^* \rightarrow \text{Id}, \quad \varepsilon_* : L_* R_* \rightarrow \text{Id}.$$

This involves a small abuse of notation for the left and right liftings in the sense that e.g. $\eta_! : \text{Id}_! \rightarrow (RL)_!$ has a (co)domain that is isomorphic but generally not equal to the ones given above.

The isomorphisms follow from uniqueness of adjoints.

Example 2.3.36 (Sets and reflexive graphs). Continuing examples 2.3.7 and 2.3.8, there is a pair of adjoint functors $\sqcap \dashv \Delta$:

$$\begin{aligned} \sqcap : \mathbf{RG} &\rightarrow \mathbf{Point} : _ \mapsto *, \\ \Delta : \mathbf{Point} &\rightarrow \mathbf{RG} : * \mapsto \mathbb{N}. \end{aligned}$$

We have $\sqcap \Delta = \text{Id}_{\mathbf{Point}}$.

Lifting these yields an adjoint quadruple $\sqcap \dashv \Delta \dashv \sqcup \dashv \nabla$ of functors between $\mathbf{Psh}(\mathbf{Point})$ and $\mathbf{Psh}(\mathbf{RG})$:

- $\sqcap := \sqcap_!$ maps a graph to its set of connected components,
- $\Delta := \Delta^* \cong \Delta_!$ maps a set to a discrete graph with only identity edges,
- $\sqcup := \sqcup_* \cong \Delta^*$ maps a graph to its set of nodes,
- $\nabla := \Delta_*$ maps a set to a codiscrete graph which has a unique edge between any two nodes.

Moreover, we have $\prod \Delta \cong \sqcup \Delta \cong \sqcup \nabla \cong \text{Id} : \text{Psh}(\text{Point}) \rightarrow \text{Psh}(\text{Point})$, so that we obtain idempotent (co)monads $\int \dashv \flat \dashv \ddagger$ on the category of reflexive graphs.

Similar adjoint quadruples feature in cohesive type theory [see e.g. LS16] and in the model of ParamDTT (section 9.2) [Nuy17].

2.4 Factorization Systems

In this section, we study factorization systems, which is necessary background for chapter 8 on fibrancy. This section uses a lecture note by Riehl [Rie08] and work by Grandis and Tholen [GT06] as primary sources, but we only extract the bare necessities and some definitions and proofs are expanded and reorganized with the intention of increasing accessibility while still keeping it brief.

How to read Factorization systems are the categorical foundation of notions of fibrancy in type theory, of which many examples can be found here. We recommend the reader to review section 1.6 in order to understand the motivation for studying fibrancy, and perhaps to skim the introductory section 8.1 before, after or in parallel with the current section in order to see some examples of fibrancy in action in type theory. It should be absolutely feasible to omit all proofs in the current section.

2.4.1 Introduction

A factorization system on a category \mathcal{C} provides two classes of morphisms called left maps and right maps, and a factorization of any morphism φ as the composite $\varphi = \rho \circ \lambda$ of a right map ρ and a left map λ . A type T will be called fibrant if the weakening substitution $(\Gamma, x : T) \rightarrow \Gamma$ is a right map.

In section 2.4.2, we introduce orthogonal factorization systems (OFSs), which are factorization systems where the factorization of a morphism is unique up to isomorphism. A consequence of this is that the operations \mathbf{L} and \mathbf{R} sending φ to its left/right factor, are closure operators, sending left/right morphisms to themselves. Another consequence is that any square from a left morphism to a right morphism (called a *lifting problem*) has a unique diagonal (the *solution* or *lifting*).

In section 2.4.3, we move to weak factorization systems (WFS), which drop the condition that the factorization must be unique. Then the existence of solutions to lifting problems can no longer be proven, so it is required in the definition of

a WFS, again non-uniquely. This means that \mathbf{L} and \mathbf{R} are not really operators at all: a WFS merely asserts the existence of *some* factorization, but doesn't pick one. If we have choice, we can pick one ourselves, but this operation will of course be completely arbitrary.

Functorial WFSs (FWFSs) (section 2.4.4) address this issue by asking a functorial factorization operation. However, FWFSs are still ill-behaved as the solutions to lifting problems are not an operation.

Natural WFSs (NWFSs) (section 2.4.5) are finally the appropriate algebraization of WFSs. An NWFS is an FWFS in which lifting problems are required to be in some sense unique. As is not unusual in category theory when we change a uniqueness condition into an algebraic operation, the operators \mathbf{L} and \mathbf{R} that send a morphism φ to its left/right factors λ and φ are now no longer closure operators, but a comonad and a monad. The right (left) morphisms are then the (co)algebras for this (co)monad.

In section 2.4.6, we discuss a procedure for creating NWFSs (including OFSs). We can pick a class of *generating left maps*, define the right maps to be those that lift the generating left maps, and the left maps to be those that lift the right maps. This turns out to be an NWFS. All notions of fibrancy of interest in this thesis can be generated in this way⁵ and the robustness condition on NWFSs in chapter 8 is a condition about the way the NWFS is generated from the left.

2.4.2 Orthogonal Factorization Systems (OFSs)

We remind the reader of notation 2.2.17.

Definition 2.4.1. Two morphisms $\ell_{\uparrow} : \ell_0 \rightarrow \ell_1$ and $r_{\uparrow} : r_0 \rightarrow r_1$ are **orthogonal** (denoted $\vec{\ell} \perp \vec{r}$) if, for every $\vec{\varphi} : \text{Hom}_{\mathcal{C}_{\uparrow}}(\vec{\ell}, \vec{r})$ (called a **lifting problem**), there exists a *unique* lifting (a.k.a. filler or solution) $\sigma : \ell_1 \rightarrow r_0$:

$$\begin{array}{ccc}
 \ell_0 & \xrightarrow{\varphi_0} & r_0 \\
 \ell_{\uparrow} \downarrow & \nearrow \sigma & \downarrow r_{\uparrow} \\
 \ell_1 & \xrightarrow{\varphi_1} & r_1.
 \end{array} \tag{2.8}$$

⁵In fact, that statement is vacuous, as any NWFS is generated in this way if in hindsight we let the generating left maps be *all* left maps. So of course we mean *non-trivially* generated.

Definition 2.4.2. [nLa20f] An **orthogonal factorization system (OFS)** $(\mathcal{L}, \mathcal{R})$ on a category \mathcal{C} consists of two classes of morphisms \mathcal{L} and \mathcal{R} (called **left** and **right maps** resp.)

- which both contain all isomorphisms and are both closed under composition,
- such that every morphism φ factors as $\varphi = r_{\uparrow} \circ \ell_{\uparrow}$ with $\vec{\ell} \in \mathcal{L}$ and $\vec{r} \in \mathcal{R}$ in a *unique* way up to *unique* isomorphism.

Example 2.4.3 (Injective presheaf morphisms). In presheaf categories, we can take surjective morphisms as left maps and injective morphisms as right maps. The factorization is over the image of the morphism.

Example 2.4.4 (Fully faithful functors). [nLa20f] In \mathbf{Cat} , we can take functors bijective on objects as left maps, and fully faithful functors as right maps. The factorization of $F : \mathcal{C} \rightarrow \mathcal{D}$ is over the category \mathcal{E} whose objects are those of \mathcal{C} but where $\text{Hom}_{\mathcal{E}}(x, y) = \text{Hom}_{\mathcal{D}}(Fx, Fy)$.

Example 2.4.5 (Codiscrete graphs). Similarly, in the category of reflexive graphs $\mathbf{Psh}(\mathbf{RG})$, we call a morphism $\sigma : \Gamma \rightarrow \Delta$ a **codiscrete fibration** if, for any two nodes $\gamma_0, \gamma_1 : \mathbb{N} \Rightarrow \Gamma$ and every edge $\delta : \mathbb{I} \Rightarrow \Delta$ from $\sigma \circ \gamma_0$ to $\sigma \circ \gamma_1$ (i.e. $\delta \circ \mathbf{s} = \sigma \circ \gamma_0$ and $\delta \circ \mathbf{t} = \sigma \circ \gamma_1$), there is a unique edge $\gamma : \mathbb{I} \Rightarrow \Gamma$ from γ_0 to γ_1 (i.e. $\gamma \circ \mathbf{s} = \gamma_0$ and $\gamma \circ \mathbf{t} = \gamma_1$) such that $\sigma \circ \gamma = \delta$. This is the analogue of a fully faithful functor for reflexive graphs. We call a graph Γ codiscrete if $\Gamma \rightarrow \top$ is a codiscrete fibration. This means that there is a unique edge between any two nodes.

Then the codiscrete fibrations are the right class of an OFS whose left class consists of graph morphisms that are bijective on nodes. The factorization of $\sigma : \Gamma \rightarrow \Delta$ is over the graph Θ whose nodes are those of Γ and whose edges are triples $(\gamma_0, \gamma_1, \delta)$ where γ_0 and γ_1 are nodes of Γ and δ is an edge of Δ from $\sigma \circ \gamma_0$ to $\sigma \circ \gamma_1$.

Example 2.4.6 (Discrete graphs). In the category of reflexive graphs $\mathbf{Psh}(\mathbf{RG})$, we call a morphism $\sigma : \Gamma \rightarrow \Delta$ a **discrete fibration** if every edge in Γ that is sent to a reflexive edge in Δ , is itself reflexive. Here, we call an edge $\delta : \mathbb{I} \Rightarrow \Delta$ reflexive if it is of the form $\delta_0 \circ \mathbf{r}$ for some node $\delta_0 : \mathbb{N} \Rightarrow \Delta$. Of course, this node is then $\delta_0 = \delta \circ \mathbf{s}$, since $\mathbf{r} \circ \mathbf{s} = \text{id}_{\mathbb{N}}$. So an edge δ is reflexive if and only if $\delta = \delta \circ \mathbf{s} \circ \mathbf{r}$, i.e. if it is the reflexive edge at its own source. So we can express the discreteness condition as:

$$\forall(\gamma : \mathbb{I} \Rightarrow \Gamma).(\sigma \circ \gamma = \sigma \circ \gamma \circ \mathbf{s} \circ \mathbf{r}) \Rightarrow (\gamma = \gamma \circ \mathbf{s} \circ \mathbf{r}).$$

We call a graph Γ discrete if $\Gamma \rightarrow \top$ is a discrete fibration. This means that there are only reflexive edges.

Given $\sigma : \Gamma \rightarrow \Delta$, a discrete fibration $\text{Fac } \sigma \rightarrow \Delta$ is obtained by simply identifying the necessary cells in Γ . Then we obtain an OFS where the right maps are the discrete fibrations, and where the left maps are the maps $\sigma : \Gamma \rightarrow \Delta$ for which $\text{Fac } \sigma \rightarrow \Delta$ is an isomorphism.

Example 2.4.7 (0-discrete depth cubical sets). In the models of ParamDTT and RelDTT (chapter 9) [NVD17a; ND18a; Nuy18a], one is interested in 0-discrete fibrations (discrete fibrations for short), which are morphisms $\sigma : \Gamma \rightarrow \Delta$ in $\text{Psh}(\text{DCube}_d)$ such that every cube in Γ is reflexive in every $\{0\}$ -dimension in which it becomes reflexive in Δ . This condition can be expressed as:

$$\forall(\gamma : (W, i : \{0\}) \Rightarrow \Gamma).(\sigma \circ \gamma = \sigma \circ \gamma \circ (0/i, i/\circ)) \Rightarrow (\gamma = \gamma \circ (0/i, i/\circ)).$$

This gives rise to an OFS, whose factorization is obtained in a manner similar to the previous example. An object Γ is called 0-discrete if $\Gamma \rightarrow \top$ is a 0-discrete fibration.

In the category DCube_{-1} , the concept of a $\{0\}$ -edge is unavailable. There, we interpret the trivially satisfied relation as the $\{0\}$ -edge relation. The requirement that reflexivity is reflected, then becomes the requirement that equality is reflected. Note that $\text{DCube}_{-1} \cong \text{Point}$, so depth -1 cubical sets are just sets, and discreteness then means injectivity.

Proposition 2.4.8. In an OFS, the left maps are precisely those orthogonal to all right maps, and vice versa.

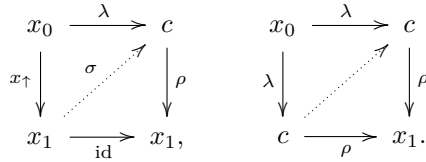
Proof. We only prove the former statement, the latter follows by duality.

First, let $\vec{\ell} \in \mathcal{L}$ and $\vec{r} \in \mathcal{R}$. We show that $\vec{\ell} \perp \vec{r}$. Pick a lifting problem $\vec{\varphi} : \vec{\ell} \rightarrow \vec{r}$. Then we can factor $\varphi_0 = \rho_0 \circ \lambda_0$ and $\varphi_1 = \rho_1 \circ \lambda_1$ and by uniqueness of the factorization of $r_{\uparrow} \circ \varphi_0 = \varphi_1 \circ \ell_{\uparrow}$, we get a unique isomorphism:

$$\begin{array}{ccccc} & & \varphi_0 & & \\ & & \curvearrowright & & \\ \ell_0 & \xrightarrow{\lambda_0} & c_0 & \xrightarrow{\rho_0} & r_0 \\ & & \cong \downarrow c_{\uparrow} & & \downarrow r_{\uparrow} \\ \ell_1 & \xrightarrow{\lambda_1} & c_1 & \xrightarrow{\rho_1} & r_1 \\ & & \varphi_1 & & \end{array}$$

Thus, we get a diagonal $\rho_0 \circ c_{\uparrow}^{-1} \circ \lambda_1 : \ell_1 \rightarrow r_0$. This diagonal is unique, because every other such diagonal σ must factor as $\sigma = \rho_{\sigma} \circ \lambda_{\sigma}$, and ρ_{σ} is essentially ρ_0 by uniqueness of the factorization of $\varphi_0 = \sigma \circ \ell_{\uparrow}$, and λ_{σ} is essentially λ_1 by uniqueness of the factorization of $\varphi_1 = r_{\uparrow} \circ \sigma$. Thus, $\vec{\ell} \perp \vec{r}$.

Now assume that \vec{x} is orthogonal to all right maps. We factor it as $x_{\uparrow} = \rho \circ \lambda$ and get unique solutions to the following lifting problems:



However, the second diagram has two solutions: id_c and $\sigma \circ \rho : c \rightarrow c$. Thus, these must be equal, so that ρ is an isomorphism and therefore a left map and then $x_{\uparrow} = \rho \circ \lambda$ is also a left map. \square

2.4.3 Weak Factorization Systems (WFSs)

While there are quite some interesting examples of orthogonal factorization systems, it is sometimes not reasonable to ask that the factorization and the solution to the lifting problem be unique. If it is not, then there are multiple ways to generalize the definition:

- We can merely ask that there exist *some* solution. The appropriate generalization is called a **weak factorization system (WFS)**.
- We can ask for a factorization *operation* that selects a designated factorization for every morphism. It is then typically desirable that this operation be functorial: it should also factor commuting squares. This is called a **functorial weak factorization system (FWFS)**.⁶
- It turns out that the structure of an FWFS can be organized so that it reveals two functors: a pointed one and a copointed one. We saw in example 2.2.52 that if we want algebraic structures which satisfy some interesting laws expressed as general equations, then we really need the multiplication operation. A generalization of an FWFSs is a **natural weak factorization system (NWFS)** [GT06], in which both functors are upgraded to monads.

Definition 2.4.9. Two morphisms $\vec{\ell}$ and \vec{r} have the **left/right lifting property** w.r.t. each other (denoted $\vec{\ell} \pitchfork \vec{r}$) if, for every lifting problem $\vec{\varphi} : \vec{\ell} \rightarrow \vec{r}$, there exists *some* solution $\sigma : \ell_1 \rightarrow r_0$.

Definition 2.4.10. A **weak factorization system (WFS)** $(\mathcal{L}, \mathcal{R})$ on a category \mathcal{C} consists of two classes of morphisms \mathcal{L} and \mathcal{R} (called **left** and **right maps** resp.)

⁶Some authors use the term *algebraic* weak factorization system (AWFS) but there seems to be disagreement as to the precise meaning.

- such that the left maps are precisely those that have the left lifting property w.r.t. all right maps and vice versa,
- such that every morphism φ factors as $\varphi = r_{\uparrow} \circ \ell_{\uparrow}$ with $\vec{\ell} \in \mathcal{L}$ and $\vec{r} \in \mathcal{R}$.

Note that for OFSs we proved proposition 2.4.8 from uniqueness of the factorization. Here, we do not have that property, so we put its implication in the definition. Then it is immediately clear that both classes contain all isomorphisms and are closed under composition, which we stated in definition 2.4.2 but can omit now.

Example 2.4.11 (Surjective functions). [Rie08] In \mathbf{Set} , we can take as the *left* class the injective functions and as the *right* class the surjective functions. So we have swapped classes when comparing to example 2.4.3. Lifting problems are solved using the axiom of choice. A factorization of $f : A \rightarrow B$ is possible over $A \uplus B$ or $A \times B$. This is a WFS but not an OFS.

Remark 2.4.12. A **model structure** on a category \mathcal{C} consists of three classes of morphisms $(\mathcal{L}, \mathcal{W}, \mathcal{R})$, called the classes of **cofibrations**, **weak equivalences** and **fibrations**, such that $(\mathcal{L}, \mathcal{W} \cap \mathcal{R})$ and $(\mathcal{W} \cap \mathcal{L}, \mathcal{R})$ are WFSs and a few other properties are satisfied. Elements of $\mathcal{W} \cap \mathcal{L}$ or $\mathcal{W} \cap \mathcal{R}$ are called **trivial/acyclic (co)fibrations**. [nLa20e]

Every WFS gives rise to a model structure by taking \mathcal{W} to be the class of *all* maps [Rie08]. In this sense, it is justified to speak of fibrations instead of right maps, and of (acyclic) cofibrations instead of left maps. We will refrain from this in the current chapter, but will speak of fibrations in chapter 8 so that we can also speak of fibrant types. We will refrain from using the word cofibration altogether, because we are never actually using an interesting model structure.

2.4.4 Functorial Weak Factorization Systems (FWFSs)

Definition 2.4.13. A **functorial weak factorization system (FWFS)** is a WFS equipped with:

- A functor $\mathbf{Fac} : \mathcal{C}^{\uparrow} \rightarrow \mathcal{C}$,
- Natural transformations $\ell : \mathbf{Dom} \rightarrow \mathbf{Fac}$ (landing in \mathcal{L}) and $\mathbf{r} : \mathbf{Fac} \rightarrow \mathbf{Cod}$ (landing in \mathcal{R}) such that $\mathbf{r}_{\vec{x}} \circ \ell_{\vec{x}} = x_{\uparrow} : x_0 \rightarrow x_1$.

These give rise to:

- A copointed **left coreplacement** endofunctor $\mathbf{L} : \mathcal{C}^{\uparrow} \rightarrow \mathcal{C}^{\uparrow} : \vec{x} \mapsto (x_0 \xrightarrow{\ell_{\vec{x}}} \mathbf{Fac} \vec{x})$ with co-unit $\vec{\varepsilon}_{\vec{x}}^{\mathbf{L}} = (\mathrm{id}_{x_0}, \mathbf{r}_{\vec{x}})$,

- A pointed **right replacement** endofunctor $\mathbf{R} : \mathcal{C}^\uparrow \rightarrow \mathcal{C}^\uparrow : \vec{x} \mapsto (\mathbf{Fac} \vec{x} \xrightarrow{\mathbf{r}_{\vec{x}}} x_1)$ with unit $\vec{\eta}_{\vec{x}}^{\mathbf{R}} = (\ell_{\vec{x}}, \text{id}_{x_1})$.

The following theorem shows that \mathbf{Fac} , ℓ and \mathbf{r} fully determine the FWFS, *provided* that it is known they are part of some FWFS:

Theorem 2.4.14. We have:

$$\mathcal{L} = \{ \vec{x} \in \mathcal{C}^\uparrow \mid \vec{x} \text{ admits a coalgebra structure for } (\mathbf{L}, \vec{\varepsilon}^{\mathbf{L}}) \},$$

$$\mathcal{R} = \{ \vec{x} \in \mathcal{C}^\uparrow \mid \vec{x} \text{ admits an algebra structure for } (\mathbf{R}, \vec{\eta}^{\mathbf{R}}) \}.$$

Proof. We only prove the second statement, the first one is dual.

Take $\vec{r} \in \mathcal{R}$. We get a solution χ_0 for the following lifting problem:

$$\begin{array}{ccc} r_0 & \xrightarrow{\text{id}} & r_0 \\ \ell_{\vec{r}} \downarrow & \nearrow \chi_0 & \downarrow r_{\uparrow} \\ \mathbf{Fac} \vec{r} & \xrightarrow{\mathbf{r}_{\vec{r}}} & r_1 \end{array}$$

Then we have $\vec{\chi} := (\chi_0, \text{id}_{r_1}) : \mathbf{R} \vec{r} \rightarrow \vec{r}$ and $\vec{\chi} \circ \vec{\eta}_{\vec{r}}^{\mathbf{R}} = \text{id}_{\vec{r}}$, so $\vec{\chi}$ is an $(\mathbf{R}, \vec{\eta}^{\mathbf{R}})$ -algebra structure for \vec{r} .

Conversely, let $(\vec{x}, \vec{\chi})$ be an $(\mathbf{R}, \vec{\eta}^{\mathbf{R}})$ -algebra. Note that, necessarily, $\chi_1 = \text{id}$, i.e. $\vec{\chi}$ is of the form (χ_0, id) . We show that \vec{x} is a right map by showing $\vec{\ell} \pitchfork \vec{x}$ for all left maps $\vec{\ell}$. We solve the lifting problem $\vec{\varphi} : \vec{\ell} \rightarrow \vec{x}$ by solving $\mathbf{R} \vec{\varphi} \circ \vec{\eta}_{\vec{\ell}}^{\mathbf{R}} = (\mathbf{Fac} \vec{\varphi} \circ \ell_{\vec{\ell}}, \varphi_1 \circ \text{id}) : \vec{\ell} \rightarrow \mathbf{R} \vec{x}$ and then postcomposing with $\vec{\chi} : \mathbf{R} \vec{x} \rightarrow \vec{x}$:

$$\begin{array}{ccccccc} \ell_0 & \xrightarrow{\text{id}} & \ell_0 & \xrightarrow{\varphi_0} & x_0 & \xrightarrow{\text{id}} & x_0 \\ \downarrow \ell_{\uparrow} & & \downarrow \ell_{\vec{\ell}} & & \downarrow \ell_{\vec{x}} & \nearrow \chi_0 & \downarrow x_{\uparrow} \\ \mathbf{Fac} \vec{\ell} & \xrightarrow{\mathbf{Fac} \vec{\varphi}} & \mathbf{Fac} \vec{x} & & \mathbf{Fac} \vec{x} & & \mathbf{Fac} \vec{x} \\ \nearrow \mathbf{r}_{\vec{\ell}} & & \nearrow \sigma & & \searrow \mathbf{r}_{\vec{x}} & & \downarrow x_{\uparrow} \\ \ell_1 & \xrightarrow{\varphi_1} & \ell_1 & & x_1 & & x_1 \end{array}$$

This completes the proof. \square

Example 2.4.15 (Surjective functions). Continuing example 2.4.11, the factorization of $f : A \rightarrow B$ over $A \uplus B$ is functorial. Then we have

$$\mathbf{L}(A \xrightarrow{f} B) = (A \xrightarrow{\text{inl}} A \uplus B), \quad \mathbf{R}(A \xrightarrow{f} B) = (A \uplus B \xrightarrow{[f, \text{id}]} B).$$

According to theorem 2.4.14, a function $(A \xrightarrow{f} B)$ is surjective if and only if it admits an $(\mathbf{R}, \bar{\eta}^{\mathbf{R}})$ -algebra structure. This is easily seen to mean that f has a section, which can indeed be obtained using the axiom of choice.

Dually, according to theorem 2.4.14, a function $(A \xrightarrow{f} B)$ is injective if and only if it admits an $(\mathbf{L}, \bar{\varepsilon}^{\mathbf{L}})$ -algebra structure. Such an algebra structure is essentially a map $g : B \rightarrow A \uplus B$ which sends $f(a)$ to $\text{inl } a$ (which can only be done if f is injective) and $b \notin f(A)$ to $g(b) = \text{inr}(b)$.

Dual reasoning applies to the factorization of $f : A \rightarrow B$ over $A \times B$.

2.4.5 Natural Weak Factorization Systems (NWFSs)

Above theorem 2.4.14, we emphasized that the functorial factorization $(\text{Fac}, \ell, \mathbf{r})$ only determines the underlying WFS *provided* that there is an underlying WFS. The reason is that otherwise, we cannot prove that ℓ and \mathbf{r} produce (co)algebras.

Indeed, if we want \mathbf{r} to produce for any $\vec{x} \in \mathcal{C}^\uparrow$ an algebra $(\mathbf{R} \vec{x}, \bar{\mu}_{\vec{x}}^{\mathbf{R}})$, then we need to postulate for any \vec{x} a morphism $\bar{\mu}_{\vec{x}}^{\mathbf{R}} : \mathbf{R} \mathbf{R} \vec{x} \rightarrow \mathbf{R} \vec{x}$, such that $\bar{\mu}_{\vec{x}}^{\mathbf{R}} \circ \bar{\eta}_{\mathbf{R} \vec{x}}^{\mathbf{R}} = \text{id}_{\mathbf{R} \vec{x}}$. If we want to do this naturally, then we need a natural transformation $\bar{\mu}^{\mathbf{R}} : \mathbf{R} \mathbf{R} \rightarrow \mathbf{R}$ such that $\bar{\mu}^{\mathbf{R}} \circ \bar{\eta}^{\mathbf{R}} = \text{id}_{\mathbf{R}} : \mathbf{R} \rightarrow \mathbf{R}$. This starts to look a lot like a monadic multiplication for \mathbf{R} .

And there is more. In FWFSs, we have guaranteed that the factorization is functorial, but the solution to the lifting problem is still only asserted by an existential quantifier. We would like to move to an algebraic operation. And then one would expect that this is operation is natural. Concretely, write $\vec{\ell} \pitchfork \vec{r}$ for the set of operators proving that $\vec{\ell} \in \mathcal{C}^\uparrow$ and $\vec{r} \in \mathcal{C}^\uparrow$ satisfy the lifting property. We can spell it out as

$$\vec{\ell} \pitchfork \vec{r} := (\vec{\varphi} : \vec{\ell} \rightarrow \vec{r}) \rightarrow \{\sigma : l_1 \rightarrow r_0 \mid \sigma \circ \ell_\uparrow = \varphi_0 \text{ and } r_\uparrow \circ \sigma = \varphi_1\}.$$

Then we would, naïvely, want a natural solution for all left and right maps, i.e. an element of the dependent end

$$\forall(\vec{\ell} \in \mathcal{L}). \forall(\vec{r} \in \mathcal{R}). (\vec{\ell} \pitchfork \vec{r}).$$

Originally, \mathcal{L} and \mathcal{R} were defined as sets, but in order to take ends, we can regard them as full subcategories of \mathcal{C}^\uparrow .

However, in the light of the results of section 2.4.4, we should know better. Indeed, there we have seen that \mathcal{L} and \mathcal{R} are the arrows that admit a (co)algebra structure, and we have exploited this (co)algebra structure to reconstruct the liftings. So if we want algebraic liftings, then we need algebraic structures, i.e. we need the right maps to be not just maps that admit an algebra structure, but maps *equipped* with an algebra structure. Thus, we shall generalize from subsets $\mathcal{L}, \mathcal{R} \subseteq \text{Obj}(\mathcal{C}^\uparrow)$ to functors $U_{\mathbf{L}} : \mathcal{L} \rightarrow \mathcal{C}^\uparrow$ and $U_{\mathbf{R}} : \mathcal{R} \rightarrow \mathcal{C}^\uparrow$, and the two sides of the factorization will factor over these as $C_{\mathbf{L}} : \mathcal{C}^\uparrow \rightarrow \mathcal{L}$ and $F_{\mathbf{R}} : \mathcal{C}^\uparrow \rightarrow \mathcal{R}$. The (co)pointings will be typed $\varepsilon^{\mathbf{L}} : \mathbf{L} = U_{\mathbf{L}}C_{\mathbf{L}} \rightarrow \text{Id}$ and $\tilde{\eta}^{\mathbf{R}} : \text{Id} \rightarrow \mathbf{R} = U_{\mathbf{R}}F_{\mathbf{R}}$.

In section 2.4.4, we derived the (co)algebra structure from the lifts and vice versa. Now that both will be algebraic operations, we have to show moreover that these derivations are mutually inverse. This will require that if \tilde{r} is a right map, then the $\tilde{\chi}$ obtained in section 2.4.4 is a morphism of right maps, i.e. in the image of $U_{\mathbf{R}}$. Thus, for any right map $\hat{r} \in \mathcal{R}$, which manifests as $U_{\mathbf{R}}\hat{r} \in \mathcal{C}^\uparrow$, we require a morphism of right maps $F_{\mathbf{R}}U_{\mathbf{R}}\hat{r} \rightarrow \hat{r}$. So we want a natural transformation $\varepsilon^{\mathbf{R}} : F_{\mathbf{R}}U_{\mathbf{R}} \rightarrow \text{Id}_{\mathcal{R}}$. In short, we require that $U_{\mathbf{R}} \dashv F_{\mathbf{R}}$ and dually $C_{\mathbf{L}} \dashv U_{\mathbf{L}}$, turning \mathbf{L} into a comonad and \mathbf{R} into a monad.

This yields Grandis and Tholen's notion of a natural weak factorization system (NWFS) [GT06]. We give a different definition that better suits the current narrative, but then prove that it is essentially the same thing.

Definition 2.4.16. A **natural weak factorization system (NWFS)** on a category \mathcal{C} consists of the following data:

- A functor $\text{Fac} : \mathcal{C}^\uparrow \rightarrow \mathcal{C}$.
- Categories \mathcal{L} and \mathcal{R} of left/right maps.
- Two adjoint functors $C_{\mathbf{L}} \dashv U_{\mathbf{L}}$ where $U_{\mathbf{L}} : \mathcal{L} \rightarrow \mathcal{C}^\uparrow$.

We write $\mathbf{L} = U_{\mathbf{L}}C_{\mathbf{L}}$ for the composite **left coreplacement** comonad, $\eta^{\mathbf{L}} : \text{Id}_{\mathcal{L}} \rightarrow C_{\mathbf{L}}U_{\mathbf{L}}$ for the unit, and $\varepsilon^{\mathbf{L}} : \mathbf{L} = U_{\mathbf{L}}C_{\mathbf{L}} \rightarrow \text{Id}_{\mathcal{C}^\uparrow}$ for the co-unit.

We require

$$\text{Dom } \mathbf{L} = \text{Dom}, \quad \text{Cod } \mathbf{L} = \text{Fac}, \quad \text{Dom } \varepsilon^{\mathbf{L}} = \text{id}, \quad \text{Dom } U_{\mathbf{L}}\eta^{\mathbf{L}} = \text{id}.$$

If $\acute{\ell} \in \mathcal{L}$, then we write $\vec{\ell} = (\ell_0 \xrightarrow{\acute{\ell}} \ell_1) := U_{\mathbf{L}}\acute{\ell}$.

- Two adjoint functors $U_{\mathbf{R}} \dashv F_{\mathbf{R}}$ where $U_{\mathbf{R}} : \mathcal{R} \rightarrow \mathcal{C}^\uparrow$.

We write $\mathbf{R} = U_{\mathbf{R}}F_{\mathbf{R}}$ for the composite **right replacement** monad, $\tilde{\eta}^{\mathbf{R}} : \text{Id}_{\mathcal{C}^\uparrow} \rightarrow U_{\mathbf{R}}F_{\mathbf{R}} = \mathbf{R}$ for the unit, and $\varepsilon^{\mathbf{R}} : F_{\mathbf{R}}U_{\mathbf{R}} \rightarrow \text{Id}_{\mathcal{C}}$ for the co-unit.

We require

$$\text{Dom } \mathbf{R} = \text{Fac}, \quad \text{Cod } \mathbf{R} = \text{Cod}, \quad \text{Cod } \tilde{\eta}^{\mathbf{R}} = \text{id}, \quad \text{Cod } U_{\mathbf{R}}\varepsilon^{\mathbf{R}} = \text{id}.$$

If $\hat{r} \in \mathcal{R}$, then we write $\vec{r} = (r_0 \xrightarrow{r_\uparrow} r_1) := U_{\mathbf{R}} \hat{r}$.

These must satisfy the following conditions:

- $\text{mor } \mathbf{L} = \text{Dom } \bar{\eta}^{\mathbf{R}} =: \ell : \text{Dom} \rightarrow \text{Fac}$.
- $\text{mor } \mathbf{R} = \text{Cod } \bar{\varepsilon}^{\mathbf{L}} =: \mathbf{r} : \text{Fac} \rightarrow \text{Cod}$.
- Left maps are precisely maps equipped with a natural lifting operation w.r.t. all right maps and vice versa, in the following sense:⁷

$$\mathcal{L} \cong \int_{\vec{x} \in \mathcal{C}^\uparrow} \forall (\hat{r} \in \mathcal{R}). (\vec{x} \pitchfork \vec{r}),$$

$$\mathcal{R} \cong \int_{\vec{x} \in \mathcal{C}^\uparrow} \forall (\hat{\ell} \in \mathcal{L}). (\vec{\ell} \pitchfork \vec{x}),$$

and in such a way that the operators associated to $\hat{\ell}$ and \hat{r} yield the same element of $\vec{\ell} \pitchfork \vec{r}$. Of course the carriers in the right-hand categories should be obtained via $U_{\mathbf{L}}$ and $U_{\mathbf{R}}$.

- For any $\hat{\ell} \in \mathcal{L}$, the canonical solution to the lifting problem $\bar{\eta}_{U_{\mathbf{L}} \hat{\ell}}^{\mathbf{R}} : U_{\mathbf{L}} \hat{\ell} \rightarrow \mathbf{R} U_{\mathbf{L}} \hat{\ell}$ is $\eta_{\hat{\ell}}^{\mathbf{L}} := \text{Cod}(U_{\mathbf{L}} \hat{\eta}_{\hat{\ell}}^{\mathbf{L}}) : \ell_1 \rightarrow \text{Fac } \vec{\ell}$.
- For any $\hat{r} \in \mathcal{R}$, the canonical solution to the lifting problem $\bar{\varepsilon}_{U_{\mathbf{R}} \hat{r}}^{\mathbf{L}} : \mathbf{L} U_{\mathbf{R}} \hat{r} \rightarrow U_{\mathbf{R}} \hat{r}$ is $\varepsilon_{\hat{r}}^{\mathbf{R}} := \text{Dom}(U_{\mathbf{R}} \bar{\varepsilon}_{\hat{r}}^{\mathbf{R}}) : \text{Fac } \vec{r} \rightarrow r_0$.

$$\begin{array}{ccc}
 \ell_0 & \xrightarrow{\ell_{\vec{\ell}}} & \text{Fac } \vec{\ell} \\
 \ell_{\uparrow} \downarrow & \nearrow \eta_{\hat{\ell}}^{\mathbf{L}} & \downarrow \mathbf{r}_{\vec{\ell}} \\
 \ell_1 & \xrightarrow{\text{id}} & \ell_1
 \end{array}
 \qquad
 \begin{array}{ccc}
 r_0 & \xrightarrow{\text{id}} & r_0 \\
 \ell_{\vec{r}} \downarrow & \nearrow \varepsilon_{\hat{r}}^{\mathbf{R}} & \downarrow r_{\uparrow} \\
 \text{Fac } \vec{r} & \xrightarrow{\mathbf{r}_{\vec{r}}} & r_1
 \end{array}$$

Proposition 2.4.17. The above implies that $\mathbf{r} \circ \ell = \text{mor} : \text{Dom} \rightarrow \text{Cod}$.

Proof. Since $\mathbf{r} \circ \ell = \text{mor } \mathbf{R} \circ \text{Dom } \bar{\eta}^{\mathbf{R}} = \text{Cod } \bar{\eta}^{\mathbf{R}} \circ \text{mor } \text{Id} = \text{mor}$. (The middle step uses naturality of $\text{mor} : \text{Dom} \rightarrow \text{Cod}$.) \square

Lemma 2.4.18. The adjoint factorization $\mathbf{R} = U_{\mathbf{R}} F_{\mathbf{R}}$ is isomorphic to the adjoint factorization over the Eilenberg-Moore category $\text{EM}(\mathbf{R})$. The dual result holds for \mathbf{L} .

⁷Recall that we take categories of elements over functors from the twisted arrow category (definition 2.2.37).

Proof. We only prove the theorem for \mathbf{R} . Without loss of generality, we assume that the isomorphism for \mathcal{R} in the definition, is an equality. By proposition 2.2.51, we get a unique morphism of adjoint factorizations $H : \mathcal{R} \rightarrow \mathbf{EM}(R) : (\tau, h) \mapsto (\tau, U_{\mathbf{R}}\eta_{(\tau, h)}^{\mathbf{R}})$. We need to provide an inverse $G : \mathbf{EM}(R) \rightarrow \mathcal{R}$, which is then automatically also a morphism of adjoint factorizations.

We first explain intuitively why this is going to work. Notice that a general lifting problem $\tilde{\varphi} : U_{\mathbf{L}}\hat{\ell} \rightarrow U_{\mathbf{R}}\hat{r}$ factors as $\tilde{\varphi} = \tilde{\varepsilon}_{U_{\mathbf{R}}\hat{r}}^{\mathbf{L}} \circ (\mathbf{L}\tilde{\varphi} \circ U_{\mathbf{L}}\hat{\eta}_{\hat{\ell}}^{\mathbf{L}})$ over $\mathbf{L}U_{\mathbf{R}}\hat{r}$:

$$\begin{array}{ccccc}
 & & \varphi_0 & & \\
 & \curvearrowright & & \curvearrowleft & \\
 \ell_0 & \xrightarrow{\text{id}} & \ell_0 & \xrightarrow{\varphi_0} & r_0 & \xrightarrow{\text{id}} & r_0 \\
 \ell_{\uparrow} \downarrow & & \downarrow \ell_{\tilde{\varepsilon}} & & \downarrow \ell_{\tilde{r}} & & \downarrow r_{\uparrow} \\
 \ell_1 & \xrightarrow{\eta_{\hat{\ell}}^{\mathbf{L}}} & \text{Fac } \vec{\ell} & \xrightarrow{\text{Fac } \tilde{\varphi}} & \text{Fac } \vec{r} & \xrightarrow{r_{\tilde{r}}} & r_1 \\
 & & & & & & \varphi_1 \\
 & \curvearrowleft & & \curvearrowright & & &
 \end{array} \tag{2.9}$$

So by naturality, h is entirely determined by $h(\mathbf{L}\vec{r}, \tilde{\varepsilon}_{\vec{r}}^{\mathbf{L}})$, which according to the definition of an NWFS is $\varepsilon_{\vec{r}}^{\mathbf{R}}$, the domain of $U_{\mathbf{R}}\hat{\varepsilon}_{\vec{r}}^{\mathbf{R}}$, which is the algebra structure! In other words, H is lossless.

It is then clear how to define G :

$$G(\vec{x}, \vec{\chi}) := (\vec{x}, h_{\vec{\chi}}), \quad h_{\vec{\chi}}(\hat{\ell}, \tilde{\varphi}) = \chi_0 \circ \text{Fac } \tilde{\varphi} \circ \eta_{\hat{\ell}}^{\mathbf{L}}.$$

From the above, it is clear that $h_{\vec{\chi}}$ provides appropriate liftings and that $G = H^{-1}$. □

Lemma 2.4.19. In an NWFS, the solution to a lifting problem $\tilde{\varphi} : U_{\mathbf{L}}\hat{\ell} \rightarrow U_{\mathbf{R}}\hat{r}$ is given by $\tilde{\varepsilon}_{\hat{r}}^{\mathbf{R}} \circ \text{Fac } \tilde{\varphi} \circ \eta_{\hat{\ell}}^{\mathbf{L}}$.

Proof. Clear from eq. (2.9). □

We will now prove equivalence to Grandis and Tholen’s original definition of an NWFS:

Definition 2.4.20. A Grandis-Tholen NWFS [GT06] consists of:

- A right replacement monad $(\mathbf{R}, \vec{\eta}^{\mathbf{R}}, \vec{\mu}^{\mathbf{R}})$ on \mathcal{C}^{\uparrow} which is trivial at the codomain,
- A left coreplacement comonad $(\mathbf{L}, \tilde{\varepsilon}^{\mathbf{L}}, \tilde{\delta}^{\mathbf{L}})$ on \mathcal{C}^{\uparrow} which is trivial at the domain,

such that

- $\text{Dom } \mathbf{R} = \text{Cod } \mathbf{L} =: \text{Fac}$,
- $\text{mor } \mathbf{L} = \text{Dom } \eta^{\mathbf{R}} =: \ell : \text{Dom} \rightarrow \text{Fac}$,
- $\text{mor } \mathbf{R} = \text{Cod } \varepsilon^{\mathbf{L}} =: \mathbf{r} : \text{Fac} \rightarrow \text{Cod}$.

Theorem 2.4.21. A Grandis-Tholen NWFS determines a unique NWFS (up to isomorphism).

Proof. Uniqueness is immediate from lemmas 2.4.18 and 2.4.19.

For existence, of course we pick $\mathcal{L} = \text{EM}(L)$ and $\mathcal{R} = \text{EM}(R)$. We need to show the last three properties in the definition, but we already have all the objects and can use the notations from definition 2.4.16. Of course, we solve lifting problems in the way prescribed by lemma 2.4.19. This gives us a functor

$$G : \text{EM}(R) \rightarrow \int_{\vec{x} \in \mathcal{C}^\uparrow} \forall (\ell \in \mathcal{L}). (\vec{\ell} \dashv \vec{x}).$$

Before we define the inverse, we show that for any $\dot{r} = (\vec{r}, \vec{\chi}) \in \mathcal{R}$, the canonical solution to the lifting problem $\varepsilon_{\vec{r}}^{\mathbf{L}} : \mathbf{L} \vec{r} \rightarrow \vec{r}$ is $\varepsilon_{\vec{r}}^{\mathbf{R}} = \text{Dom } U_{\mathbf{R}} \varepsilon_{\vec{r}}^{\mathbf{R}}$, which equals $\text{Dom } U_{\mathbf{R}} \vec{\chi} = \chi_0$ because as we saw in the proof of proposition 2.2.50, the co-unit in $\text{EM}(\mathbf{R})$ is just the algebra structure. This is actually trivial, because for $\vec{\varphi} = \varepsilon_{\vec{r}}^{\mathbf{L}}$ the left two squares in eq. (2.9) disappear thanks to the monad laws, so we are left with just the dotted diagonal $\varepsilon_{\vec{r}}^{\mathbf{R}}$.

We define the inverse H to G by $H(\vec{r}, h) = (\vec{r}, (h(\ell_{\vec{r}}, \varepsilon_{\vec{r}}^{\mathbf{L}}), \text{id}))$. Then we get, for $\dot{r} = (\vec{r}, \vec{\chi})$:

$$HG\dot{r} = (\vec{r}, (\varepsilon_{\vec{r}}^{\mathbf{R}}, \text{id})) = (\vec{r}, U_{\mathbf{R}} \varepsilon_{\vec{r}}^{\mathbf{R}}) = (\vec{r}, U_{\mathbf{R}} \varepsilon_{(\vec{r}, \vec{\chi})}^{\mathbf{R}}) = (\vec{r}, U_{\mathbf{R}} \vec{\chi}) = (\vec{r}, \vec{\chi}) = \dot{r}.$$

We prove that $GH = \text{Id}$ as in the proof of lemma 2.4.18.

The dual requirements are proven dually. □

Example 2.4.22 (Surjective functions). Continuing examples 2.4.11 and 2.4.15, we can build a Grandis-Tholen NWFS which factors $(A \xrightarrow{f} B)$ over $A \uplus B$. Right maps are then functions equipped with an \mathbf{R} -algebra structure, which is essentially a section. Morphisms of right maps are commutative squares *compatible with the section*. Left maps will be functions equipped with an \mathbf{L} -coalgebra structure (which uniquely exists if the function is injective), but morphisms $(h, k) : (A \xrightarrow{f} B) \rightarrow (C \xrightarrow{g} D)$ will be coalgebra morphisms, which requires that $k(b) \in g(C)$ if and only if $b \in f(A)$.

Solutions s to lifting problems $(h, k) : (A \xrightarrow{f} B) \rightarrow (C \xrightarrow{g} D)$ are given by case distinction: if $b = f(a)$, then $s(b) = h(a)$. Otherwise, i.e. if $b \notin f(A)$, then we use the section of g . Thus, these solutions are natural, but *only* w.r.t. morphisms of left/right maps that respect the above criteria.

OFSs as NWFSs

Proposition 2.4.23. An OFS can be equivalently defined as an NWFS for which \mathbf{L} and \mathbf{R} are idempotent. The classes of the NWFS are then automatically full subcategories of \mathcal{C}^\uparrow .

Proof. Given an OFS, let $\mathbf{r}_{\vec{x}} \circ \ell_{\vec{x}}$ be the unique factorization of $x_\uparrow : x_0 \rightarrow x_1$. This creates a monad and a comonad.⁸ Uniqueness implies that $\ell_{\mathbf{R}\vec{x}}$ (and hence $\bar{\eta}^{\mathbf{R}\mathbf{R}}$) and $\mathbf{r}_{\mathbf{L}\vec{x}}$ (and hence $\bar{\varepsilon}^{\mathbf{L}\mathbf{L}}$) are invertible, i.e. that \mathbf{L} and \mathbf{R} are idempotent. Then we know that $\mathbf{EM}(\mathbf{L})$ and $\mathbf{EM}(\mathbf{R})$ are fully faithful subcategories. The objects of $\mathbf{EM}(\mathbf{L})$ are precisely those whose factorization yields an invertible right morphism, i.e. exactly the left morphisms of the OFS, and similarly on the right.

Conversely, assume we have an NWFS with idempotent \mathbf{L} and \mathbf{R} . We first show that solutions to lifting problems are unique. We already know that any lifting problem $\vec{\varphi} : \vec{\ell} \rightarrow \vec{r}$ factors over the square

$$\begin{array}{ccc}
 r_0 & \xrightarrow{\text{id}} & r_0 \\
 \ell_{\vec{r}} \downarrow \cong & \nearrow \cong & \downarrow r_\uparrow \\
 \text{Fac } \vec{r} & \xrightarrow{\mathbf{r}_{\vec{r}}} & r_1.
 \end{array}$$

Now since this square is full of isomorphisms, there is really only one way to solve problems that factor through it.

Next, we show that factorizations are unique. Suppose $\varphi = \rho' \circ \lambda = \rho \circ \lambda'$. Then we can do:

$$\begin{array}{ccc}
 \bullet & \xrightarrow{\lambda} & \bullet \\
 \lambda' \downarrow & & \downarrow \rho' \\
 \bullet & \xrightarrow{\rho} & \bullet
 \end{array}$$

⁸The functorial action of the factorization on $\vec{\varphi} : \vec{x} \rightarrow \vec{y}$ is given by the unique lifting of $\ell_{\vec{x}}$ against $\mathbf{r}_{\vec{y}}$.

Now this square is both horizontally and vertically a lifting problem that has precisely one solution. This shows that the category of factorizations of φ has singleton Hom-sets. Hence, all factorizations are isomorphic. \square

Example 2.4.24. The factorizations for the examples in section 2.4.2 are all easily seen to yield idempotent (co)monads.

2.4.6 Left Generation in Presheaf Categories

We prove the small object argument specialized to presheaf categories. An in-depth treatment is given by Garner [Gar07; Gar09]. Note that the algebraic version of the small object argument does not actually involve an argument about a small object.

Theorem 2.4.25 (Small object argument). Write $\mathcal{C} = \text{Psh}(\mathcal{W})$. Assume given a category \mathcal{G} of **generating left maps** equipped with a functor $U_{\mathcal{G}} : \mathcal{G} \rightarrow \mathcal{C}^{\uparrow}$. If $\check{g} \in \mathcal{G}$, we write $\vec{g} := U_{\mathcal{G}}\check{g}$.

Then there exists a **left generated** NWFS on \mathcal{C} such that

$$\mathcal{R} = \int_{\vec{x} \in \mathcal{C}^{\uparrow}} \forall \check{g} \in \mathcal{G}. \vec{g} \pitchfork \vec{x},$$

$$\mathcal{L} = \int_{\vec{x} \in \mathcal{C}^{\uparrow}} \forall \hat{r} \in \mathcal{R}. \vec{x} \pitchfork \hat{r}.$$

We write $I_{\mathcal{G}}$ for the canonical functor $\mathcal{G} \rightarrow \mathcal{L}$ and $\acute{g} := I_{\mathcal{G}}\check{g}$. We have $U_{\mathcal{L}} \circ I_{\mathcal{G}} = U_{\mathcal{G}}$.

Remark 2.4.26. Note that uncurrying reveals that $\text{Psh}(\mathcal{W})^{\uparrow} = (\text{Set}^{\mathcal{W}^{\text{op}}})^{\uparrow} \cong \text{Set}^{\mathcal{W}^{\text{op}} \times \uparrow} = \text{Psh}(\mathcal{W} \times \uparrow^{\text{op}})$ is itself a presheaf category. Thus, it satisfies all properties of presheaf categories, e.g. we can take (co)limits pointwise.

Proof. We first construct a Grandis-Tholen NWFS.

R We define a functor $\vec{R} : \mathcal{C}^{\uparrow} \rightarrow \mathcal{C}^{\uparrow}$ as follows. Pick $\vec{x} \in \mathcal{C}^{\uparrow}$. Then define $\vec{\ell}(\vec{x})$ as follows:

$$\vec{\ell}(\vec{x}) = \text{colim}_{(\check{g}, \vec{\varphi}: \vec{g} \rightarrow \vec{x})} \vec{g},$$

in other words:

$$(W \Rightarrow \ell_i(\vec{x})) \cong \exists (\check{g} \in \mathcal{G}). (\vec{g} \rightarrow \vec{x}) \times (W \Rightarrow x_i).$$

Then clearly any lifting problem of a generating left map against \vec{x} factors canonically over $\vec{\varphi}(\vec{x}) : \vec{\ell}(\vec{x}) \rightarrow \vec{x}$, and this factorization is natural.

We now define $R_0\vec{x}$ to be the following pushout:

$$\begin{array}{ccc} \ell_0(\vec{x}) & \xrightarrow{\varphi_0(\vec{x})} & x_0 \\ x^\uparrow \downarrow & & \downarrow \\ \ell_1(\vec{x}) & \longrightarrow & R_0\vec{x}. \end{array}$$

This way, the lifting problem $\varphi(\vec{x}) : \vec{\ell}(\vec{x}) \rightarrow \vec{x}$ gets a wannabe solution $\ell_1(\vec{x}) \rightarrow R_0\vec{x}$:

$$\begin{array}{ccccc} g_0 & \longrightarrow & \ell_0(\vec{x}) & \xrightarrow{\varphi_0(\vec{x})} & x_0 \\ \downarrow g^\uparrow & & \downarrow x^\uparrow & & \downarrow \\ & & & \nearrow & R_0\vec{x} \\ & & & & \downarrow R^\uparrow\vec{x} \\ g_1 & \longrightarrow & \ell_1(\vec{x}) & \xrightarrow{\varphi_1(\vec{x})} & x_1, \end{array}$$

and with that so does every lifting problem $\vec{g} \rightarrow \vec{x}$ for $\vec{g} \in \mathcal{G}$.

One might hope that this puts $\vec{R}\vec{x} = (R_0\vec{x} \xrightarrow{R^\uparrow\vec{x}} x_1)$ in \mathcal{R} , but this is not the case, as the set of lifting problems for $\vec{R}\vec{x}$ is larger than that of \vec{x} . Instead, we define $(\mathbf{R}, \vec{\eta}^{\mathbf{R}}, \vec{\mu}^{\mathbf{R}})$ to be the free monad over the pointed functor \vec{R} , which exists and can be constructed as a colimit (proposition 2.2.53).

Now $\mathbf{R}\vec{x}$ lifts generating left maps. Indeed, pick a lifting problem $\vec{\varphi} : \vec{g} \rightarrow \mathbf{R}\vec{x}$ where $\vec{g} \in \mathcal{G}$. Then $\vec{\varphi}$ has a wannabe solution in $\vec{R}\mathbf{R}\vec{x}$ and we can absorb $\vec{R}\mathbf{R} \rightarrow \mathbf{R}$ yielding an actual solution.

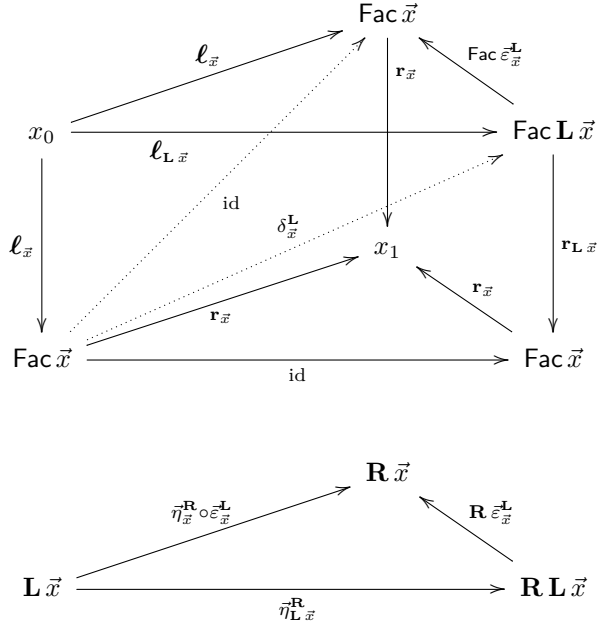
- L** The definition of \mathbf{R} immediately produces a copointed endofunctor $(\mathbf{L}, \vec{\varepsilon}^{\mathbf{L}})$. We still need to define the comonadic duplication $\vec{\delta}^{\mathbf{L}}$ and prove the comonad laws. In order to define it, we argue that the lifting problem $\vec{\eta}_{\mathbf{L}\vec{x}}^{\mathbf{R}} : \mathbf{L}\vec{x} \rightarrow \mathbf{R}\mathbf{L}\vec{x}$ has a solution, naturally in \vec{x} :

$$\begin{array}{ccc} x_0 & \xrightarrow{\ell_{\mathbf{L}\vec{x}}} & \text{Fac } \mathbf{L}\vec{x} \\ \ell_{\vec{x}} \downarrow & \nearrow \delta_{\vec{x}}^{\mathbf{L}} & \downarrow \mathbf{r}_{\mathbf{L}\vec{x}} \\ \text{Fac } \vec{x} & \xrightarrow{\text{id}} & \text{Fac } \vec{x}. \end{array}$$

Indeed, the lower left corner $\text{Fac } \vec{x}$ was defined via a procédé that only involved colimits. In order to provide a lifting, it is sufficient to provide a natural lifting for all the constituents of the colimits, and a careful analysis reveals that these are all generating left maps, which are indeed lifted by $\mathbf{R} \mathbf{L} \vec{x}$. This defines $\bar{\delta}_{\vec{x}}^{\mathbf{L}} := (\text{id}_{x_0}, \delta_{\vec{x}}^{\mathbf{L}}) : \mathbf{L} \vec{x} \rightarrow \mathbf{L} \mathbf{L} \vec{x}$.

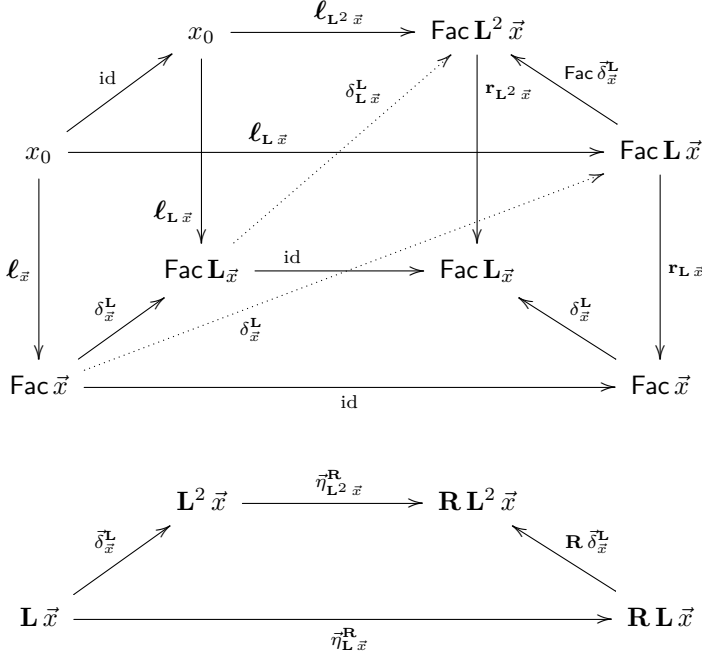
Note that the upper *right* triangle of the above square is $\bar{\varepsilon}_{\mathbf{L} \vec{x}}^{\mathbf{L}} : \mathbf{L} \mathbf{L} \vec{x} \rightarrow \mathbf{L} \vec{x}$, revealing that $\bar{\varepsilon}_{\mathbf{L} \vec{x}}^{\mathbf{L}} \circ \bar{\delta}_{\vec{x}}^{\mathbf{L}} = \text{id}_{\mathbf{L} \vec{x}}$.

To prove the other co-unit law, consider the following diagram, above expanded in \mathcal{C}^\uparrow , below in \mathcal{C} , which commutes by naturality of $\bar{\eta}^{\mathbf{R}}$:



Here, in front, we see the lifting problem that we solved to define $\bar{\delta}^{\mathbf{L}}$. In the back, we get another lifting problem, which has a solution by the same reasoning that we applied above. Recall how we constructed the lifting: we disassembled $\text{Fac } \vec{x}$ on the left, finding out that it is basically x_0 with liftings of generating left maps added explicitly via colimits, and the arrow on the right lifts generating left maps, so we can deal with those added liftings. If we then recall why $\text{Fac } \vec{x}$ lifts generating left maps, it's because we explicitly added them! Basically, this is a laborious construction of the identity function. By naturality of the reasoning (disassembling the left arrow) that lead to the liftings, we find that $\text{Fac } \bar{\varepsilon}_{\vec{x}}^{\mathbf{L}} \circ \delta_{\vec{x}}^{\mathbf{L}} = \text{id}_{\text{Fac } \vec{x}}$. If we write this equation in the arrow category with constant x_0 at the domain, we get $\mathbf{L} \bar{\varepsilon}_{\vec{x}}^{\mathbf{L}} \circ \delta_{\vec{x}}^{\mathbf{L}} = \text{id}_{\mathbf{L}}$, the other co-unit law.

To prove associativity, we pull off a similar trick. Consider the following diagram, above expanded in \mathcal{C}^\uparrow , below in \mathcal{C} , which commutes by naturality of $\bar{\eta}^{\mathbf{R}}$:



In front, once more, we see the lifting problem that we solved to define $\bar{\delta}_{\vec{x}}^{\mathbf{L}}$. In the back, we see another instance of the same problem, yielding $\bar{\delta}_{\vec{x}}^{\mathbf{L}}$. The liftings are compatible by naturality of the construction of the factorization. This means that $\text{Fac } \bar{\delta}_{\vec{x}}^{\mathbf{L}} \circ \delta_{\vec{x}}^{\mathbf{L}} = \delta_{\text{Fac } \vec{x}}^{\mathbf{L}} \circ \delta_{\vec{x}}^{\mathbf{L}}$. If we state this in the arrow category with x_0 at the domain, we get $\mathbf{L} \bar{\delta}_{\vec{x}}^{\mathbf{L}} \circ \delta_{\vec{x}}^{\mathbf{L}} = \delta_{\vec{x}}^{\mathbf{L}} \mathbf{L} \circ \delta_{\vec{x}}^{\mathbf{L}}$, the associativity law.

Thus, we gave a Grandis-Tholen NWFS. By the proof of theorem 2.4.21, this yields an NWFS whose left and right classes are the Eilenberg-Moore categories of \mathbf{L} and \mathbf{R} . For the left class, our work is done. For the right class, it remains to be shown that $\text{EM}(\mathbf{R}) \cong \int_{\vec{x} \in \mathcal{C}^\uparrow} \forall (\vec{y} \in \mathcal{G}). (\vec{y} \pitchfork \vec{x})$, i.e. that an \mathbf{R} -algebra structure is essentially the same thing as a lifting operation for all generating left maps.

We already know that an \mathbf{R} -algebra structure is essentially the same thing as a lifting operation for *all* left maps (including the generating ones), since the structure is the solution to a universal lifting problem. Conversely, this universal lifting problem given by the algebra structure on \hat{r} is against $\ell_{\vec{r}} : r_0 \rightarrow \text{Fac } \vec{r}$,

which is (by construction of \mathbf{R}) a colimit of generating left maps, so it is also uniquely determined by the action on generating left maps. \square

Remark 2.4.27 (Generating OFSs). An easy way to make sure that the NWFS you are generating, is an OFS, is by including generating left maps that enforce the uniqueness of the liftings of other generating left maps. First of all, notice that if we intend to generate an OFS, then the morphism part of \mathcal{G} is unimportant, so let us assume it will be a full subcategory of \mathcal{C}^\uparrow . If we have $\vec{g} \in \mathcal{G}$, then by also adding the map $\vec{h} = (g_1 \uplus_{g_0} g_1 \xrightarrow{[\text{id}, \text{id}]} g_1)$, we ensure uniqueness of liftings of the former. Indeed, if $\vec{\varphi} : \vec{g} \rightarrow \vec{r}$ has two liftings σ, τ , then from these two liftings we can create a lifting problem $([\sigma, \tau], \varphi_1) \vec{h} \rightarrow \vec{r}$, the lifting of which witnesses that $\sigma = \tau$.

Examples

Example 2.4.28 (Surjective functions). Continuing examples 2.4.11, 2.4.15 and 2.4.22, the NWFS on \mathbf{Set} that factorizes $f : A \rightarrow B$ over $A \uplus B$ can easily be generated from the left. Indeed, let \mathcal{G} be the point category, and $U_{\mathcal{G}}^* = (\emptyset \rightarrow \{\bullet\})$. Then a lifting structure against this single generating left map is exactly a section. Moreover, the monad \mathbf{R} that explicitly adds all liftings of generating left maps is exactly (isomorphic to) $\mathbf{R}(A \xrightarrow{f} B) = (A \uplus B \xrightarrow{[f, \text{id}]} B)$.

Example 2.4.29 (Injective presheaf morphisms). Continuing example 2.4.3, the OFS on any presheaf category whose right maps are injective presheaf morphisms can be left generated by letting \mathcal{G} contain an arrow $\vec{g}_W = (\mathbf{y}W \uplus \mathbf{y}W \xrightarrow{[\text{id}, \text{id}]} \mathbf{y}W)$ for every base category object W . Indeed, if x_\uparrow maps $\gamma, \gamma' : W \Rightarrow x_0$ to the same cell $\delta : W \Rightarrow x_1$, then the solution to the lifting problem $([\gamma, \gamma'], \delta) : \vec{g}_W \rightarrow \vec{x}$ witnesses that $\gamma = \gamma'$.

Example 2.4.30 (Fully faithful functors). Continuing example 2.4.4, the OFS on \mathbf{Cat} whose right maps are fully faithful functors, can be left generated by asking unique liftings of the functor $\{\bullet \ \bullet\} \rightarrow \{\bullet \rightarrow \bullet\}$.

Example 2.4.31 (Codiscrete graphs). Similarly, continuing example 2.4.5, the OFS on $\mathbf{Psh}(\mathbf{RG})$ whose right maps are codiscrete fibrations, can be left generated by asking unique liftings of the graph morphism $(\mathbf{y}\mathbf{N} \uplus \mathbf{y}\mathbf{N} \xrightarrow{[\mathbf{ys}, \mathbf{yt}]} \mathbf{y}\mathbb{I})$.

Example 2.4.32 (Discrete graph morphisms). Continuing example 2.4.6, the OFS on $\mathbf{Psh}(\mathbf{RG})$ whose right maps are discrete fibrations, can be left generated by asking (automatically unique) liftings of $(\mathbf{y}\mathbb{I} \xrightarrow{\mathbf{y}\mathbf{r}} \mathbf{y}\mathbf{N})$.

Example 2.4.33 (0-discreteness). Continuing example 2.4.7, the OFS on $\mathbf{Psh}(\mathbf{DCube}_d)$ (for $d > 0$) whose right maps are 0-discrete fibrations, can be left generated by asking (automatically unique) liftings of $(\mathbf{y}(W, i : \mathbb{0})) \xrightarrow{\mathbf{y}(i/\mathbb{0})} \mathbf{y}W$.

For $d = -1$, we are back at example 2.4.29.

Example 2.4.34 (Clock-irrelevance). In guarded type theory, one is interested in **clock-irrelevant** presheaves over \mathbf{Clock} [BM18]. The clock object $\mathbb{0}$ is obtained as $\mathbb{0} = \text{colim}_k \mathbf{y}(i : \mathbb{0}_k)$. It is internalized as a closed type and the intention is that non-dependent functions $\mathbb{0} \rightarrow A$ are all constant. In order to model this property, one restricts to clock-irrelevant types.

The OFS whose right maps are clock-irrelevant fibrations, is defined by asking unique liftings against morphisms $\mathbf{y}W \times \mathbb{0} \rightarrow \mathbf{y}W$.⁹ We call an object Γ clock-irrelevant if $\Gamma \rightarrow \top$ is a clock-irrelevant fibration.

Example 2.4.35 (Segal fibrations). [From Nuy18b] A category can be defined as a simplicial set that satisfies the Segal condition (due to Grothendieck) [Seg68], which states that any chain of n consecutive 1-simplices, can be extended to an n -simplex in a unique way. The 0-simplices then serve as objects, the 1-simplices as morphisms, and higher simplices as commutative diagrams. The underlying simplicial set is called the category's *nerve*.

It is possible to define an OFS on the category $\mathbf{Psh}(\mathbf{Simplex})$ of simplicial sets such that a simplicial set Γ satisfies the Segal condition if and only if $\Gamma \rightarrow \top$ is a right map, which we will call a **Segal fibration**¹⁰. Indeed, we simply ask that Segal fibrations uniquely lift inclusions $\Lambda_n \rightarrow \Delta_n$ where $\Delta_n = \mathbf{y}[n]$ and Λ_n is a chain of n consecutive lines, i.e.

$$\Lambda_n := \mathbf{y}[1] \uplus_{\mathbf{y}[0]} \mathbf{y}[1] \uplus_{\mathbf{y}[0]} \dots \uplus_{\mathbf{y}[0]} \mathbf{y}[1].$$

Example 2.4.36 (Kan fibrations). In cubical approaches to HoTT [AHH18; BCH14; CMS20; Coh+17; Hub16; Ort18; OP18], one is interested in Kan fibrations of cubical sets. There are already multiple notions of cubical sets to begin with section 2.3.2, and each notion of cubical set may inspire multiple notions of Kan fibrancy, so we will give a high-level discussion.

Kan fibrations are the right maps of an NWFS left generated by maps $\mathbf{y}(W, i : \mathbb{1}).(\varphi \vee (i \doteq \epsilon)) \rightarrow \mathbf{y}(W, i : \mathbb{1})$ where $\epsilon \in \{0, 1\}$ in some treatments and $\epsilon = 0$ in others, and where φ is a predicate on $\mathbf{y}W$ (i.e. $\mathbf{y}W.\varphi$ is a subpresheaf

⁹In fact, Bizjak and Møgelberg [BM18] impose a stronger condition in their model, namely that there be unique liftings against $\mathbf{y}(W, i : \mathbb{0}_k) \rightarrow \mathbf{y}W$ for all k ; however, as the authors acknowledge, this is stronger than what is necessary to model the type system.

¹⁰This seems to be semi-standard terminology at best.

Chapter 3

Formal Systems and Dependent Type Theory

In section 3.1, we use generalized algebraic theories (GATs) to define what a type theory is. In section 3.2, we proceed to define dependent type theory (DTT) as a GAT. Finally, in section 3.3, we review the Curry-Howard correspondence between propositions and types.

3.1 Formal Systems, Syntax and Models

Before defining the syntax of a specific type system, it is worthwhile to define the very concept of a formal system, its syntax and its models. To this end, we use Cartmell's notion of generalized algebraic theories (GATs) [Car86; Car78], which is general enough to encompass dependently typed systems. We will introduce it gently, starting with simple algebraic theories, subsequently considering multisorted algebraic theories and finally generalized algebraic theories. We will not consider notions of equivalence of different (multisorted/generalized) algebraic theories.

3.1.1 Motivation

Traditionally, one views type theoretic judgements as predicates. For example, the type judgement $\Gamma \vdash T$ type which states that T is a type in context Γ , or

rather its derivability, is regarded as a binary relation relating contexts and types. Derivability of these judgements is then defined by mutual induction, and the constructors of these inductive predicates are then called inference rules, e.g. the rule

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \text{ Bool type}}$$

asserts that if $\Gamma \text{ ctx}$ is derivable (Γ is a context), then $\Gamma \vdash \text{ Bool type}$ must be derivable (Bool is a well-formed type in context Γ).

This approach to type theory has a few drawbacks:

- In order to view judgements as predicates, we need something for them to be predicates over. That is, we need prior notions of precontexts, pretypes, preterms etc. before we can speak about derivability. But these pre-objects are meaningless and when modelling a type system we take care to avoid them completely and instead define an interpretation function on derivations as these pre-objects have no semantics. Worse, the fact that we need these prior notions, creates the impression that we have any choice in defining them. Conversely, if we do not have any choice, then we should not be burdened with defining them.

In a GAT, every inference rule defines its own operator and this operator takes all of its prerequisites as arguments. Thus, there is only well-typed syntax, and this syntax is defined by the typing rules, and judgements only speak about well-typed syntax.

- If we regard $\Gamma \vdash T \text{ type}$ as a binary relation, then that puts Γ and T on equal footing. However, the meaning of the judgement (T is a type in context Γ) clearly *states* that T is a type and already takes for granted that Γ is a context. It does not assume that Γ is a context (i.e. it does not say: if Γ is a context, then T is a type) and it does not assert it (Γ is a context and T is a type in context Γ). Rather, it is nonsense if Γ is not a context.

In a GAT, every judgement speaks about exactly one object:

- The context judgement $\Gamma \text{ ctx}$ expresses that Γ is an element of the set of contexts Ctx .
- The type judgement $\Gamma \vdash T \text{ type}$ expresses that T is an element of the set $\text{Ty}(\Gamma)$ of types in context Γ . It takes as input a *context* $\Gamma \in \text{Ctx}$, which is well-formed because it exists.
- The term judgement $\Gamma \vdash t : T$ expresses that t is an element of the set $\text{Tm}(\Gamma, T)$ of terms of type T in context Γ . It takes as input a *context* $\Gamma \in \text{Ctx}$ and a *type* $T \in \text{Ty}(\Gamma)$.

- Traditionally, dependently typed systems come with equality judgements for types and terms and sometimes also for contexts. There are inference rules asserting that judgemental equality is an equivalence relation, and every other inference rule is accompanied by a congruence rule that expresses that it respects equality. These congruence rules are rarely ever spelled out because they are completely uninteresting.

When we formulate type theory as a GAT, an equality judgement is not just another judgement. Rather, it can be seen as the equality relation on syntax. For example, if $\Gamma \vdash t : T$ means $t \in \text{Tm}(\Gamma, T)$, then $\Gamma \vdash s = t : T$ just means $s = t \in \text{Tm}(\Gamma, T)$.

- Finally, specifying something as a GAT has the advantage that it automatically comes with a category of models. Soundness and completeness results can then be stated w.r.t. that category, and categorical gluing [KHS19] can take place in this category.

3.1.2 Simple Algebraic Theories

Examples of simple algebraic theories are group theory, linear algebra and the untyped λ -calculus.

Definition 3.1.1. A **simple algebraic theory** \mathfrak{A} consists of

- a set of operators, each equipped with an arity which is a natural,
- a set of axioms, which are pairs (t_1, t_2) of n -ary terms ($n \in \mathbb{N}$), denoted as $t_1 = t_2$.

An n -ary term of \mathfrak{A} ($n \in \mathbb{N}$) is either a metavariable \mathbf{m}_i ($i < n$) or an expression of the form $o(t_1, \dots, t_k)$ where o is an operator of arity k and all t_i are also n -ary terms.

An **algebra** or **model** A of \mathfrak{A} is a set A equipped with a function $\llbracket o \rrbracket : A^k \rightarrow A$ for every k -ary operator o , satisfying all the axioms. Given $\vec{a} = (a_1, \dots, a_n) \in A^n$, the interpretation of an n -ary term t is defined recursively by $\llbracket o(t_1, \dots, t_k) \rrbracket_{\vec{a}} = \llbracket o \rrbracket(\llbracket t_1 \rrbracket_{\vec{a}}, \dots, \llbracket t_k \rrbracket_{\vec{a}})$ and $\llbracket \mathbf{m}_i \rrbracket_{\vec{a}} = a_i$. We say that A satisfies the axiom $t_1 = t_2$ if $\llbracket t_1 \rrbracket_{\vec{a}} = \llbracket t_2 \rrbracket_{\vec{a}}$ for all $\vec{a} \in A^n$.

A **morphism of algebras/models** is a function $f : A \rightarrow B$ such that $f \circ \llbracket o \rrbracket = \llbracket o \rrbracket \circ f^{\times k} : A^k \rightarrow B$ for every k -ary operator o .

The **syntax** of \mathfrak{A} is the initial object of the category of \mathfrak{A} -algebras. It is the set of nullary terms of \mathfrak{A} , divided by the equivalence relation generated by the axioms.

Remark 3.1.2. A simple algebraic theory can be reorganized into a category, whose objects are natural numbers, and where $\text{Hom}(n, m)$ is the set of m -tuples of n -ary terms, up to the axioms. This yields a more obscure definition of an algebraic theory, also called Lawvere theory [nLa20d; Law63]. An advantage of Lawvere theories is that they have no ‘hair’: fundamentally, we care about terms and the equality relation on terms, and not about which operators and axioms generate them. Lawvere theories put all terms and equalities on equal footing.

Example 3.1.3. Group theory has operators e (nullary), \sqsubset^{-1} (unary) and $*$ (binary), and a bunch of axioms, such as $x^{-1} * x = e$. An algebra for group theory is just a group. The syntax is the trivial group $\{e\}$.

Example 3.1.4. Fix a set S . If we add to group theory a nullary operator for every $s \in S$, then we get a new theory whose syntax is the free group over S .

Example 3.1.5. The untyped λ -calculus has operators:

- abstraction λ (unary),
- application (binary; denoted as juxtaposition),
- for every $n \in \mathbb{N}$:
 - a variable \mathbf{v}_n (nullary),
 - shift \uparrow_n (unary),
 - substitution $\sqsubset[\sqsubset/\mathbf{v}_n]$ (binary),

and axioms such as $(\lambda t_1)t_2 = t_2[t_1/\mathbf{v}_0]$ and $(\lambda t_1)[t_2/\mathbf{v}_n] = \lambda(t_1[\uparrow_0 t_2/\mathbf{v}_{n+1}])$. Its syntax is the set of untyped λ -terms up to computation rules.

3.1.3 Multisorted Algebraic Theories

The concept of a simple algebraic theory is not strong enough to capture, e.g., category theory. For this reason, we will incrementally enrich our notion of algebraic theories.

Definition 3.1.6. A multisorted algebraic theory \mathfrak{A} consists of:

- a set of sorts,
- a set of operators, each equipped with an arity, which is a list of sorts, and an output sort,
- a set of axioms $t_1 = t_2$ where t_1 and t_2 are terms of the same arity and output sort.

An \vec{s} -ary term of output sort s is either a metavariable \mathbf{m}_i where $s_i = s$, or an expression of the form $o(t_1, \dots, t_k)$, where o has arity \vec{r} of length k and output sort s and every t_i is an \vec{s} -ary term of output sort r_i .

An **algebra** or **model** A of \mathfrak{A} consists of a set A_s for every sort s , and functions $\llbracket o \rrbracket : (\prod_i A_{r_i}) \rightarrow A_r$ for every \vec{r} -ary operator o of output sort r , satisfying the axioms. The interpretation of terms and the concept of satisfying an axiom are defined similarly as in definition 3.1.1.

Again, we define the syntax as the initial \mathfrak{A} -algebra.

Remark 3.1.7. A remark analogous to remark 3.1.2 applies.

We could now attempt to define category theory as a multisorted algebraic theory. As sorts, we would have Obj and Hom , and there would be operators $\text{id} : \text{Obj} \rightarrow \text{Hom}$, $\text{dom}, \text{cod} : \text{Hom} \rightarrow \text{Obj}$ and $\circ : \text{Hom} \times \text{Hom} \rightarrow \text{Hom}$. However, this does not allow us to exclude ill-typed compositions. To this end, we need to move to generalized algebraic theories.

3.1.4 Generalized Algebraic Theories (GATs)

We give only a brief and somewhat handwaving definition, and refer to Cartmell's original work for details [Car86; Car78].

Definition 3.1.8. A **generalized algebraic theory** (GAT) [Car86; Car78] \mathfrak{A} consists of:

- a set of **sort operators**, each equipped with a metacontext,
- a set of **operators**, each equipped with
 - a metacontext,
 - an output sort, which is a well-formed sort application,
- a set of **axioms** $t_1 = t_2$ where t_1 and t_2 are terms of the same sort in the same metacontext.

Equality of metacontexts and sorts is always considered up to the congruence generated by the axioms.

A **metacontext**¹ is either the empty metacontext $()$, or of the form $(\Sigma, x : S)$, where Σ is another metacontext, x is a metavariable (possibly implemented via de Bruijn indices) and S is a sort in metacontext Σ .

¹Cartmell calls this just a context, but that becomes confusing when we start defining type systems as GATs.

A **sort** $\Sigma' \Vdash S$ **sort** in metacontext Σ is of the form $S = O[\sigma]$ where O is a sort operator in metacontext Σ' and $\sigma : \Sigma \rightarrow \Sigma'$ is a substitution.

A **substitution** $\sigma : \Sigma \rightarrow \Sigma'$ is either $() : \Sigma \rightarrow ()$ or $(\sigma, t/x) : \Sigma \rightarrow (\Sigma', x : S)$ where $\sigma : \Sigma \rightarrow \Sigma'$ and $\Sigma \Vdash t : S[\sigma]$ is a term.

A **term** $\Sigma \Vdash t : S$ is either a metavariable or an operator application $o[\sigma]$ where o is an operator in metacontext Σ' of sort S' , $\sigma : \Sigma \rightarrow \Sigma'$ and $S = S'[\sigma]$.

Sorts and terms are substituted (written $S[\sigma]$ and $t[\sigma]$ respectively) by substituting variables as prescribed.

An **algebra** or **model** of \mathfrak{A} consists of:

- for every sort operator O in metacontext Σ a family of sets $\llbracket O \rrbracket : \llbracket \Sigma \rrbracket \rightarrow \mathbf{Set}$,
- for every operator o in metacontext Σ of output sort S , a dependent function $\llbracket o \rrbracket : (s : \llbracket \Sigma \rrbracket) \rightarrow \llbracket S \rrbracket(s)$, satisfying the axioms.

Metacontexts, sorts, substitutions and terms are interpreted the obvious way.

The **syntax** is again the initial object of the category of models.

Notation 3.1.9. We will often omit the interpretation brackets $\llbracket \dots \rrbracket$ and directly use the syntax of the GAT as operators on the model.

Example 3.1.10. We can now define category theory as a GAT with sort operators

- $\Vdash \mathbf{Obj}$ **sort**,
- $x : \mathbf{Obj}, y : \mathbf{Obj} \Vdash \mathbf{Hom}(x, y)$ **sort**,

and operators

- $x : \mathbf{Obj} \Vdash \mathbf{id}_x : \mathbf{Hom}(x, x)$,
- $x, y, z : \mathbf{Obj}, \varphi : \mathbf{Hom}(x, y), \chi : \mathbf{Hom}(y, z) \Vdash \chi \circ \varphi : \mathbf{Hom}(x, z)$,

and axioms such as $\mathbf{id} \circ \varphi = \varphi$.

Example 3.1.11. We can define the simply typed λ -calculus (STLC) as a GAT with sort operators

- $\Vdash \mathbf{T}_Y$ **sort**,
- $\Vdash \mathbf{C}_{Tx}$ **sort**,
- $\Gamma, \Delta : \mathbf{C}_{Tx} \Vdash \mathbf{Sub}(\Gamma, \Delta)$ **sort**,
- $\Gamma : \mathbf{C}_{Tx}, T : \mathbf{T}_Y \Vdash \mathbf{T}_m(\Gamma, T)$ **sort**,

and operators such as

- $\Vdash \text{Base} : \text{Ty}$,
- $A, B : \text{Ty} \Vdash A \rightarrow B : \text{Ty}$,
- $\Vdash () : \text{Ctx}$,
- $\Gamma : \text{Ctx}, A : \text{Ty} \Vdash \Gamma.A : \text{Ctx}$,
- $\Gamma : \text{Ctx} \Vdash () : \text{Sub}(\Gamma, ())$,
- $\Gamma, \Delta : \text{Ctx}, A : \text{Ty}, \sigma : \text{Sub}(\Gamma, \Delta), a : \text{Tm}(\Gamma, A) \Vdash (\sigma, a) : \text{Sub}(\Gamma, \Delta.A)$
- $\Gamma : \text{Ctx} \Vdash \text{id}_\Gamma : \text{Sub}(\Gamma, \Gamma)$,
- $\Gamma, \Delta, \Theta : \text{Ctx}, \sigma : \text{Sub}(\Gamma, \Delta), \tau : \text{Sub}(\Delta, \Theta) \Vdash \tau \circ \sigma : \text{Sub}(\Gamma, \Theta)$
- $\Gamma : \text{Ctx}, A, B : \text{Ty}, b : \text{Tm}(\Gamma.A, B) \Vdash \lambda b : \text{Tm}(\Gamma, A \rightarrow B)$,
- $\Gamma : \text{Ctx}, A, B : \text{Ty}, f : \text{Tm}(\Gamma, A \rightarrow B), a : \text{Tm}(\Gamma, A) \Vdash f a : \text{Tm}(\Gamma, B)$,
- ...

with axioms such as $(\sigma, a) \circ \rho = (\sigma \circ \rho, a[\rho])$.

3.1.5 Judgements and Typing Rules

Typically, when dealing with type systems such as the STLC, we will use slightly different notations and terminology:

- The metacontext will be written above a line; the assumptions in it are called **premises**,
- Instead of sort operators, we will speak of **judgement forms**. Sort ascriptions will be called **judgements**, and for every judgement form we will introduce a special notation, e.g.
 - A **type** instead of $A : \text{Ty}$,
 - A **ctx** instead of $A : \text{Ctx}$,
 - $\sigma : \Gamma \rightarrow \Delta$ instead of $\sigma : \text{Sub}(\Gamma, \Delta)$,
 - $\Gamma \vdash a : A$ instead of $a : \text{Tm}(\Gamma, A)$.
- When two GAT-terms are equal up to the axioms, we will denote this as an equality judgement.
- When our algebraic theory contains an encoding of variables using de Bruijn indices, we will take the liberty to use variable names.

Hence, operators become **typing rules** such as

$$\frac{\Gamma \text{ ctx} \quad \Delta \text{ ctx} \quad A \text{ type} \quad \sigma : \Gamma \rightarrow \Delta \quad \Gamma \vdash a : A}{(\sigma, a/x) : \Gamma \rightarrow (\Delta, x : A)} \quad (3.1)$$

(where x should be desugared using de Bruijn indices) and axioms become rules for deriving equality judgements such as

$$\frac{\Theta \text{ ctx} \quad \Gamma \text{ ctx} \quad \Delta \text{ ctx} \quad A \text{ type} \quad \rho : \Theta \rightarrow \Gamma \quad \sigma : \Gamma \rightarrow \Delta \quad \Gamma \vdash a : A}{(\sigma, a/x) \circ \rho = (\sigma \circ \rho, a[\rho]/x) : \Theta \rightarrow (\Delta, x : A)}. \quad (3.2)$$

Somewhat unusually, we will introduce judgement forms with their presuppositions like so:

$$\frac{\Gamma \text{ ctx} \quad A \text{ type}}{(\Gamma \vdash a : A) \text{ jud}}. \quad (3.3)$$

Sometimes, we will omit premises that are required by well-formedness of other judgements, e.g.

$$\frac{\sigma : \Gamma \rightarrow \Delta \quad \Gamma \vdash a : A}{(\sigma, a/x) : \Gamma \rightarrow (\Delta, x : A)}. \quad (3.4)$$

One thing that is not yet possible, is having equalities in the metacontext, i.e. equality judgements above the line. We can solve this by modifying definition 3.1.8, or by extending the specific GAT. Indeed, we can add to a GAT for every sort operator $\Sigma \Vdash O \text{ sort}$ an equality sort operator $\Sigma, x : O, y : O \Vdash x =_O y \text{ sort}$, with an operator $\Sigma, x : O \Vdash \text{refl}_x : x =_O x$ as well as axioms $\Sigma, x : O, y : O, e : x =_O y \Vdash x = y : O$ and $\Sigma, x, y : O, e_1, e_2 : x =_O y \Vdash e_1 = e_2 : x =_O y$. Then the equality sort is fully specified in models as being a singleton if and only if the equated terms are equal, and internally in the GAT we can make equality assumptions using the equality sort. For the rest of the text, it does not matter which approach we take. In any case, we take the liberty to put equality judgements in the premises of an inference rule.

Notation 3.1.12. In the spirit of notation 3.1.9, we will often populate judgements not with terms but with elements of the model, and use their notation to make claims about the model.

If the judgement's subject is not present, e.g. if we write $(\Gamma \vdash T)$ rather than $(\Gamma \vdash t : T)$, then we are talking about the set of elements of the model interpreting the sort $\text{Tm}(\Gamma, T)$.

3.2 Dependent Type Theory: Syntax and Models

In this section, we define dependent type theory (DTT) [Mar84; CH88] as a GAT. We introduce it in different parts, so that we can also discuss the complexity of models step by step, and so that we can enable and disable extensions of dependent type theory at will.

3.2.1 Contexts and Substitutions

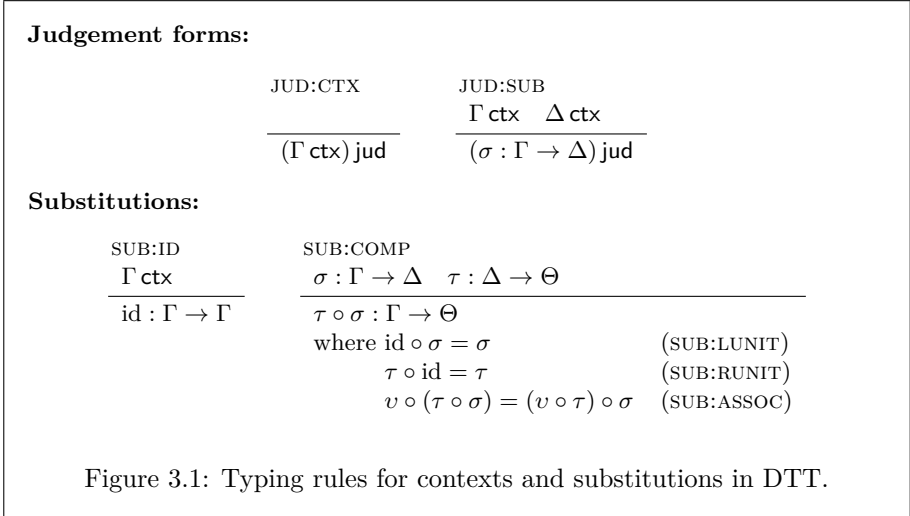


Figure 3.1 defines a GAT. It has judgement forms for contexts and substitutions, which we give the structure of a category. The equations listed below the composition rule should in principle be spelled out, each with their own premises, e.g.:

$$\frac{\sigma : \Gamma \rightarrow \Delta}{\text{id} \circ \sigma = \sigma : \Gamma \rightarrow \Delta}$$

Proposition 3.2.1. A model of the GAT defined in fig. 3.1 is exactly a category, with the contexts serving as objects and the substitutions as morphisms. More precisely, the category of models of the GAT is *isomorphic* to *Cat*. □

3.2.2 Structural Rules, Categories with Families (CwFs) and Natural Models

In fig. 3.2 we extend fig. 3.1 with judgement forms for types and terms. Types and terms can be substituted, yielding types and terms in a different context. We postulate that there is an empty context, to which there is a unique substitution. We also postulate that contexts can be extended with a variable of any type depending on the preceding variables. Substitutions to the extended context, correspond to pairs of a substitution to the smaller context and a term of the added type. The components constituting $\text{id} : (\Gamma, x : T) \rightarrow (\Gamma, x : T)$ are called

Judgement forms:

$$\frac{\text{JUD:TY} \quad \Gamma \text{ ctx}}{(\Gamma \vdash T \text{ type}) \text{ jud}} \qquad \frac{\text{JUD:TM} \quad \Gamma \text{ ctx} \quad \Gamma \vdash T \text{ type}}{(\Gamma \vdash t : T) \text{ jud}}$$

Type and term substitution:

$$\frac{\text{TY:SUB} \quad \Gamma \vdash T \text{ type} \quad \sigma : \Delta \rightarrow \Gamma}{\Delta \vdash T[\sigma] \text{ type}} \quad \text{where } T[\text{id}] = T \quad T[\sigma \circ \rho] = T[\sigma][\rho]$$

$$\frac{\text{TM:SUB} \quad \Gamma \vdash t : T \quad \sigma : \Delta \rightarrow \Gamma}{\Delta \vdash t[\sigma] : T[\sigma]} \quad \text{where } t[\text{id}] = t \quad t[\sigma \circ \rho] = t[\sigma][\rho]$$

Empty context:

$$\frac{\text{EMPTY-CTX}}{() \text{ ctx}} \qquad \frac{\text{EMPTY-CTX:INTRO} \quad \Gamma \text{ ctx}}{() : \Gamma \rightarrow ()} \quad \text{where } \sigma = () \quad (\text{EMPTY-CTX:ETA})$$

Context extension:

$$\frac{\text{CTX-EXT} \quad \Gamma \text{ ctx} \quad \Gamma \vdash T \text{ type}}{(\Gamma, x : T) \text{ ctx}} \qquad \frac{\text{CTX-EXT:INTRO} \quad \sigma : \Delta \rightarrow \Gamma \quad \Delta \vdash t : T[\sigma]}{(\sigma, t/x) : \Delta \rightarrow (\Gamma, x : T)} \quad \text{where } (\sigma, t/x) \circ \rho = (\sigma \circ \rho, t[\rho]/x) \quad \tau = (\pi_x \circ \tau, x[\tau]/x) \quad (\text{CTX-EXT:ETA})$$

$$\frac{\text{CTX-EXT:WKN} \quad \Gamma \text{ ctx} \quad \Gamma \vdash T \text{ type}}{\pi_x : (\Gamma, x : T) \rightarrow \Gamma} \quad \text{where } \pi_x \circ (\sigma, t/x) = \sigma \quad (\text{CTX-EXT:WKN:BETA})$$

$$\frac{\text{CTX-EXT:VAR} \quad \Gamma \text{ ctx} \quad \Gamma \vdash T \text{ type}}{\Gamma, x : T \vdash x : T} \quad \text{where } x[\sigma, t/x] = t \quad (\text{CTX-EXT:VAR:BETA})$$

Figure 3.2: Structural rules of DTT.

π_x (also called weakening) and x (a variable). Using these, we can extract the components of a general substitution $\sigma : \Delta \rightarrow (\Gamma, x : T)$ as $\pi_x \circ \sigma$ and $x[\sigma]$.

Notation 3.2.2. We will often omit applications of, and compositions with π_x (e.g. the rule `CTX-EXT:VAR` should type the term x as $T[\pi_x]$). The substitution $(\text{id}_\Gamma, t/x)$ (as well as e.g. $(\pi_x, t/x)$) is written more briefly as (t/x) . We do not write the round parentheses around substitutions when we extend them further, or when we apply them using square brackets.

Models: Categories with Families (CwFs)

Models of figs. 3.1 and 3.2 combined can be structured as categories with families:

Definition 3.2.3. [From Nuy18a] A **category with families** (CwF) [Dyb96] consists of:

1. A category `Ctx` whose objects we call **contexts**, and whose morphisms we call **substitutions**. We also write $\Gamma \text{ ctx}$ to say that Γ is a context.
2. A contravariant functor $\text{T}_y : \text{Ctx}^{\text{op}} \rightarrow \mathbf{Set}$. The elements $T \in \text{T}_y(\Gamma)$ are called **types** over Γ (also denoted $\Gamma \vdash T \text{ type}$). The action $\text{T}_y(\sigma) : \text{T}_y(\Gamma) \rightarrow \text{T}_y(\Delta)$ of a substitution $\sigma : \Delta \rightarrow \Gamma$ is denoted $\sqsubset[\sigma]$, i.e. if $\Gamma \vdash T \text{ type}$ then $\Delta \vdash T[\sigma] \text{ type}$.
3. A contravariant functor $\text{T}_m : (\text{T}_y/\text{Ctx})^{\text{op}} \rightarrow \mathbf{Set}$ from the category of elements of T_y to \mathbf{Set} . The elements $t \in \text{T}_m(\Gamma, T)$ are called **terms** of T (also denoted $\Gamma \vdash t : T$). The action $\text{T}_m(\sigma) : \text{T}_m(\Gamma, T) \rightarrow \text{T}_m(\Delta, T[\sigma])$ of $\sigma : (\Delta, T[\sigma]) \rightarrow (\Gamma, T)$ is denoted $\sqsubset[\sigma]$, i.e. if $\Gamma \vdash t : T$, then $\Delta \vdash t[\sigma] : T[\sigma]$.
4. A terminal object $()$ of `Ctx` called the **empty context**.
5. A **context extension** operation: if $\Gamma \text{ ctx}$ and $\Gamma \vdash T \text{ type}$, then there is a context $\Gamma.T$, a substitution $\pi : \Gamma.T \rightarrow \Gamma$ and a term $\Gamma.T \vdash \xi : T[\pi]$, such that for all Δ , the map

$$\text{Hom}(\Delta, \Gamma.T) \rightarrow \Sigma(\sigma : \text{Hom}(\Delta, \Gamma)).\text{T}_m(\Delta, T[\sigma]) : \tau \mapsto (\pi\tau, \xi[\tau])$$

is invertible. We call the inverse (\sqsupset, \sqsubset) . Note that for more precision and less readability, we could write $\pi_{\Gamma.T}$, $\xi_{\Gamma.T}$ and $(\sqsupset, \sqsubset)_{\Gamma.T}$.

If $\sigma : \Delta \rightarrow \Gamma$, then we will write $\sigma+ = (\sigma\pi, \xi) : \Delta.T[\sigma] \rightarrow \Gamma.T$.

Sometimes, for clarity, we will use variable names: we write $\Gamma, x : T$ instead of $\Gamma.T$, and $\pi_x : (\Gamma, x : T) \rightarrow \Gamma$ and $\Gamma, x : T \vdash x : T[\pi_x]$ for π and ξ . Their joint inverse will be called $(\sqsupset, \sqsubset/x)$.

Lemma 3.2.4. In a CwF, given $T \in \text{Ty}(\Gamma)$ and $\sigma : \Delta \rightarrow \Gamma$, the set $\text{Tm}(\Delta, T[\sigma])$ is naturally isomorphic to the set of morphisms $\tau : \Delta \rightarrow \Gamma.T$ such that $\pi \circ \tau = \sigma : \Delta \rightarrow \Gamma$.

$$\begin{array}{ccc} \Delta & \xrightarrow{\tau} & \Gamma.T \\ & \searrow \sigma & \downarrow \pi \\ & & \Gamma \end{array}$$

Proof. Such τ is necessarily of the form (σ, t) , with $t \in \text{Tm}(\Delta, T[\sigma])$. \square

Definition 3.2.5. [From Nuy18a] A **(weak) morphism of CwFs**² $F : \mathcal{C} \rightarrow \mathcal{D}$ consists of:

1. A functor $F_{\text{Ctx}} : \mathcal{C} \rightarrow \mathcal{D}$ (also denoted F),
2. A natural transformation $F_{\text{Ty}} : \text{Ty}_{\mathcal{C}} \rightarrow \text{Ty}_{\mathcal{D}} \circ F_{\text{Ctx}}$ (also denoted F),
3. A natural transformation $F_{\text{Tm}} : \text{Tm}_{\mathcal{C}} \rightarrow \text{Tm}_{\mathcal{D}} \circ F_{\int}$ (also denoted F),
where $F_{\int} : \int_{\mathcal{C}} \text{Ty}_{\mathcal{C}} \rightarrow \int_{\mathcal{D}} \text{Ty}_{\mathcal{D}}$ is easily constructed from F_{Ctx} and F_{Ty} ,
4. such that $F_{\text{Ctx}}()$ is terminal,
5. such that $(F_{\text{Ctx}}\pi, F_{\text{Tm}}\xi) : F_{\text{Ctx}}(\Gamma.T) \rightarrow (F_{\text{Ctx}}\Gamma).(F_{\text{Ty}}T)$ is an isomorphism.

The images of a context Γ , a substitution σ , a type T and a term t are also denoted $F\Gamma$, $F\sigma$, FT and Ft respectively.

Given $\sigma : \Delta \rightarrow F\Gamma$ and $\Delta \vdash t : (FT)[\sigma]$, we write $(\sigma, t)_F := (F\pi, F\xi)^{-1}(\sigma, t) : \Delta \rightarrow F(\Gamma.T)$. In particular, $(\pi, \xi)_F = (F\pi, F\xi)^{-1}$. In variable notation, we will write $(\sigma, t/Fx)_F : \Delta \rightarrow F(\Gamma, x : T)$. Similarly, we will write $()_F$ for the unique substitution to the terminal object $F()$.

A morphism of CwFs is called **strict** if

4. $F() = ()$,
5. $F(\Gamma.T) = (F\Gamma).(FT)$, $F\pi = \pi$ and $F\xi = \xi$.

In variable notation, we write $F(\Gamma, x : T) = (F\Gamma, Fx : FT)$. The law $F\xi = \xi$ implies that the choice of variable name in the latter context is sensible.

Proposition 3.2.6. [From Nuy18a] If G is a weak CwF morphism, and we have a natural isomorphism $\zeta : F \cong G$, then F is a weak CwF morphism.

²Strict CwF morphisms are due to Dybjer [Dyb96].

Proof. Given a type $\Gamma \vdash T$ type, we define $F\Gamma \vdash FT$ type by $FT = (GT)[\zeta]$.

Given a term $\Gamma \vdash t : T$, we define $F\Gamma \vdash Ft : FT$ by $Ft = (Gt)[\zeta]$.

Clearly, $F()$ is terminal as $F() \cong G()$.

The substitution $(F\pi, F\xi) = (F\pi, (G\xi)[\zeta]) : F(\Gamma.T) \rightarrow F\Gamma.FT = F\Gamma.(GT)[\zeta]$ is an isomorphism, because the following diagram commutes and the other trajectory consists of isomorphisms:

$$\begin{array}{ccc}
 & \xrightarrow{(F\pi, (G\xi)[\zeta])} & \\
 F(\Gamma.T) & \xrightarrow{(F\pi, F\xi)} F\Gamma.FT & \equiv F\Gamma.(GT)[\zeta] \\
 \downarrow \zeta & & \downarrow \zeta+ \\
 G(\Gamma.T) & \xrightarrow{(G\pi, G\xi)} & G\Gamma.GT
 \end{array}$$

□

It will be a corollary of the results of chapter 5, that a functor is a morphism of CwFs in at most one way up to isomorphism (corollary 5.1.8).

Proposition 3.2.7. A model of the GAT defined in figs. 3.1 and 3.2 is exactly a CwF. More precisely, the category of models of the GAT is *isomorphic* to the category of CwFs and *strict* CwF morphisms. □

Definition 3.2.8. A CwF is **democratic** if contexts can be promoted to closed types: for every context Γ there should be a closed type $\Gamma \in \text{Ty}()$ (denoted by the same symbol) such that $\Gamma \cong ().\Gamma$. [CD11]

A CwF \mathcal{C} is **locally democratic** if slices can be promoted to open types: for every context Γ , every slice $(\Delta, \sigma) \in \mathcal{C}/\Gamma$ is isomorphic to a slice of the form $(\Gamma.T, \pi)$.

Natural Models

CwFs can be structured even more succinctly using Awodey's notion of natural models [Awo18]. These have a higher level of abstraction, which makes them more obscure. The advantage is that constructions and proofs in natural models can also be given at a higher abstraction level, whereas in CwFs we typically first prove that something exists, and then prove that it is stable under substitution. Proofs of the latter are generally tedious and left to the reader, but can be more easily included in the same argument when using natural models.

Definition 3.2.9. A **natural model** [Awo18] consists of a category Ctx equipped with presheaves $\widetilde{\text{Ty}}, \text{Ty} : \text{Ctx}^{\text{op}} \rightarrow \text{Set}$ and a representable natural transformation $\tau : \widetilde{\text{Ty}} \rightarrow \text{Ty}$. We denote the base pullback of $T \in \text{Ty}(\Gamma)$ as $\Gamma.T$.

Once more, the objects and morphisms of Ctx serve as contexts and substitutions. Again, $\text{Ty}(\Gamma)$ is the set of types in context Γ (so it is the exact same functor as in the CwF modelling the same type system). The functor $\widetilde{\text{Ty}}$ sends Γ to the set of *typed terms* $\widetilde{\text{Ty}}(\Gamma)$, and $\tau_\Gamma : \widetilde{\text{Ty}}(\Gamma) \rightarrow \text{Ty}(\Gamma)$ forgets the term. As suggested by the notation, context extension is given by the base pullbacks along the representable morphism τ :

$$\begin{array}{ccc}
 \Gamma.T & \xrightarrow{\pi} & \Gamma \\
 \downarrow (\xi, T[\pi]) & \lrcorner & \downarrow T \\
 \widetilde{\text{Ty}} & \xrightarrow{\tau} & \text{Ty}.
 \end{array}$$

The fact that $\Gamma.T$ is a base pullback exactly says that substitutions $\sigma : \Delta \rightarrow \Gamma.T$ consist of a substitution $\pi \circ \sigma : \Delta \rightarrow \Gamma$ paired up with a term $\Delta \vdash \xi[\sigma] : T[\pi \circ \sigma]$. The existence of a terminal object modelling the empty context is not part of the definition of a natural model.

Definition 3.2.10. A **(weak) morphism of natural models** consists of a functor $F_{\text{Ctx}} : \mathcal{C} \rightarrow \mathcal{D}$, natural transformations $F_{\widetilde{\text{Ty}}} : \widetilde{\text{Ty}}_{\mathcal{C}} \rightarrow \widetilde{\text{Ty}}_{\mathcal{D}} \circ F_{\text{Ctx}}$ and $F_{\text{Ty}} : \text{Ty}_{\mathcal{C}} \rightarrow \text{Ty}_{\mathcal{D}} \circ F_{\text{Ctx}}$ such that $(\tau_{\mathcal{D}} F_{\text{Ctx}}) \circ F_{\widetilde{\text{Ty}}} = F_{\text{Ty}} \circ \tau_{\mathcal{C}}$. It is **strict** if it preserves the choice of base pullbacks on the nose.

Proposition 3.2.11. The category of CwFs and weak/strict CwF morphisms is equivalent to the category of natural models with a terminal object and weak/strict terminal object preserving morphisms of natural models.

Proof. When sending a CwF to a natural model, we define $\widetilde{\text{Ty}}(\Gamma) = (T \in \text{Ty}(\Gamma)) \times \text{Tm}(\Gamma, T)$. When sending a natural model to a CwF , we define $\text{Tm}(\Gamma, T) = \left\{ t \in \widetilde{\text{Ty}}(\Gamma) \mid \tau(t) = T \right\}$. \square

3.2.3 Universe Levels for Size Stratification

In many models, the concept of a type, i.e. an element of $\text{Ty}(\Gamma)$, has a size-agnostic definition. Hence, for any ordinal number ℓ , we can consider types

$T \in \text{Ty}(\Gamma)$ of size ℓ . If the sets of such types again constitute a functor $\text{Ty} : \text{Ctx}^{\text{op}} \rightarrow \text{Set}$, we shall denote this functor as Ty_ℓ . Clearly, if $k \leq \ell$, then Ty_k is a subpresheaf of Ty_ℓ . We get corresponding judgement forms $\Gamma \vdash T \text{ type}_\ell$ for every size ℓ with a typing rule

$$\frac{\Gamma \vdash T \text{ type}_k \quad k \leq \ell}{\Gamma \vdash T \text{ type}_\ell}. \quad (3.5)$$

This typing rule violates the definition of a GAT, as we are not applying an operator to T . From a syntactical point of view, we shall understand this rule as silently applying a coercion operator \uparrow_k^ℓ which satisfies $\uparrow_k^\ell T = T$ and $\uparrow_k^\ell \uparrow_j^k T = \uparrow_j^\ell T$, as well as other equalities that can be expected from an operator that semantically does nothing. Semantically, we will just make sure that $\text{Ty}_k \subseteq \text{Ty}_\ell$ if $k \leq \ell$ and that size-polymorphic operators semantically do not depend on the size. For types, we will usually speak of **levels** or **universe levels** rather than sizes.

3.2.4 Type Formers

In this section, we introduce typing rules for various important type formers in dependent type theory. Each of these type formers may be seen as an extension to the structural rules of DTT (section 3.2.2). Occasionally, we will discuss how the automatic semantics of these extensions can be simplified in the context of CwF models. For more abstract formulations in the context of natural models, we refer to Awodey's work [Awo18].

General Pattern

In general, the specification of a type consists of the following parts:

- A formation rule, expressing how the type may be formed. The corresponding operator is called the type former.
 - A rule expressing how to compute substitutions applied to this type former.
- Zero or more introduction rules, expressing how to create elements of the type. The corresponding operators are called constructors of the type.
 - A substitution rule for each constructor.
- Zero or more elimination rules, expressing how we can eliminate (destruct, analyze, use) elements of the type. The corresponding operators are called eliminators or destructors.

- A substitution rule for each eliminator.
- A computation rule, reduction rule, β -reduction rule or β -rule for every constructor-eliminator pair, expressing what happens when we apply that eliminator to that constructor.
- Optionally, an η -expansion rule or η -rule expressing how an arbitrary element of the type can be rebuilt by eliminating and reconstructing it.

An η -rule is usually only imposed on types for which it does not harm decidability of equality.

Amongst types introduced according to the above pattern, we can roughly distinguish two (overlapping) families:

Data types In a data type D , elements are defined by the constructor they were created with, and the arguments passed to that constructor. Data types have a single eliminator, typically written as a `let`- or `case`-expression, which allows us to create a function $(d : D) \rightarrow C$ by providing, for every constructor c_i with arguments $\vec{x} : \vec{A}$, a function $(\vec{x} : \vec{A}) \rightarrow C (c_i \vec{x})$. Hence, the eliminator just lets us distinguish between all the constructors and provides us with the arguments that have been provided to that constructor to create the eliminee d . Data types are also called inductive types, especially if one of the constructors again takes an argument of type D . Data types are generally not equipped with an η -rule.

Codata types In a codata type D , elements are defined by what can be observed of them using the eliminators. Codata types have a single constructor, which allows us to construct an element of D by specifying the output of every eliminator. Codata types are also called co-inductive types, especially if one of the eliminators again produces an element of type D . With an exception for some extraordinary codata types such as Π -types, we can also call them record types, calling the eliminators fields. Codata types are generally equipped with an η -rule.

Π -types

Dependent function types or Π -types³ (fig. 3.3) are codata types: their elements are functions $f : (x : A) \rightarrow B$ that can be applied to any element of A . As such, the constructor (called abstraction or λ -abstraction) needs to provide, for

³The name comes from the alternative notation $\Pi(x : A).B$.

$\frac{\text{PI}}{\frac{\Gamma \vdash A \text{ type}_\ell}{\Gamma, x : A \vdash B \text{ type}_\ell}}{\Gamma \vdash (x : A) \rightarrow B \text{ type}_\ell}$	$\frac{\text{PI:INTRO}}{\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).b : (x : A) \rightarrow B}} \quad \text{where } f = \lambda(x : A).f \text{ } x : (x : A) \rightarrow B \quad (\text{PI:ETA})$
$\frac{\text{PI:ELIM}}{\frac{\Gamma \vdash f : (x : A) \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B[a/x]}} \quad \text{where } (\lambda(x : A).b) a = b[a/x] \quad (\text{PI:BETA})$	
$\begin{aligned} ((x : A) \rightarrow B)[\sigma] &= (x : A[\sigma]) \rightarrow B[\sigma \circ \pi_x, x/x] \\ (\lambda(x : A).b)[\sigma] &= \lambda(x : A[\sigma]).b[\sigma \circ \pi_x, x/x] \\ (f a)[\sigma] &= f[\sigma] a[\sigma] \end{aligned}$	

Figure 3.3: Typing rules for Π -types in DTT.

arbitrary $x : A$, the result of that application. With some fantasy, they can be regarded as record types with A -many fields.

Definition 3.2.12. The (non-dependent) function type

$$\frac{\Gamma \vdash A, B \text{ type}_\ell}{\Gamma \vdash A \rightarrow B \text{ type}_\ell} \quad (3.6)$$

is defined as $(A \rightarrow B) := (x : A) \rightarrow B[\pi_x]$.

Proposition 3.2.13. In models, instead of modelling the application rule with substitution, β - and η -rules, it is sound and complete to provide an inverse to abstraction:

$$\frac{\text{PI:UNLAMBDA}}{\frac{\Gamma \vdash f : (x : A) \rightarrow B}{\Gamma, x : A \vdash \text{un}\lambda_x f : B}} \quad (3.7)$$

where $\text{un}\lambda_x(\lambda(x : A).b) = b \quad (\text{PI:UNLAMBDA:BETA})$
 $\lambda(x : A).\text{un}\lambda_x f = f \quad (\text{PI:UNLAMBDA:ETA})$

Proof. We can mutually define

$$\text{un}\lambda_x f := f[\pi_x] x, \quad f a := (\text{un}\lambda_x f)[a/x]. \quad (3.8)$$

It is straightforward to check that this satisfies all the necessary equalities. \square

Notation 3.2.14. In variable-free notation, we will write ΠAB for $(x : A) \rightarrow B$, λb for $\lambda(x : A).b$ and $\text{un}\lambda b$ for $\text{un}\lambda_x b$.

Σ -types

$\frac{\text{SIGMA}}{\frac{\Gamma \vdash A \text{ type}_\ell}{\Gamma, x : A \vdash B \text{ type}_\ell}}}{\Gamma \vdash (x : A) \times B \text{ type}_\ell}$	$\frac{\text{SIGMA:INTRO}}{\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : (x : A) \times B}}}{\text{where } c = (\text{fst } c, \text{snd } c) : (x : A) \times B \text{ (SIGMA:ETA)}}$
$\frac{\text{SIGMA:FST}}{\frac{\Gamma \vdash c : (x : A) \times B}{\Gamma \vdash \text{fst } c : A}}}{\text{where } \text{fst } (a, b) = a \text{ (SIGMA:FST:BETA)}}$	$\frac{\text{SIGMA:SND}}{\frac{\Gamma \vdash c : (x : A) \times B}{\Gamma \vdash \text{snd } c : B[\text{fst } c/x]}}}{\text{where } \text{snd } (a, b) = b \text{ (SIGMA:SND:BETA)}}$
$\begin{aligned} ((x : A) \times B)[\sigma] &= (x : A[\sigma]) \times B[\sigma \circ \pi_x, x/x] \\ (a, b)[\sigma] &= (a[\sigma], b[\sigma]) \\ (\text{fst } a)[\sigma] &= \text{fst } a[\sigma] \\ (\text{snd } a)[\sigma] &= \text{snd } a[\sigma] \end{aligned}$	
<p>Figure 3.4: Typing rules for Σ-types in DTT.</p>	

Dependent pair types or Σ -types⁴ (fig. 3.4) are codata types: an element c has two fields $\text{fst } c : A$ and $\text{snd } c : B[\text{fst } c/x]$. As such, the pair constructor needs to provide two components.

Definition 3.2.15. The (non-dependent) pair type

$$\frac{\Gamma \vdash A, B \text{ type}_\ell}{\Gamma \vdash A \times B \text{ type}_\ell} \tag{3.9}$$

is defined as $(A \times B) := (x : A) \times B[\pi_x]$.

Notation 3.2.16. In variable-free notation, we will write ΣAB for $(x : A) \times B$.

$\frac{}{\Gamma \vdash \mathbf{Unit} \mathit{type}_\ell}$	$\frac{}{\Gamma \vdash \mathbf{unit} : \mathbf{Unit}}$ <p style="text-align: center;">where $u = \mathbf{unit} : \mathbf{Unit}$ (UNIT:ETA)</p> $\begin{aligned} (\mathbf{Unit})[\sigma] &= \mathbf{Unit} \\ \mathbf{unit}[\sigma] &= \mathbf{unit} \end{aligned}$
---	--

Figure 3.5: Typing rules for the unit type in DTT.

Unit Type

The unit type (fig. 3.5) is a record type with zero fields. As such, the constructor takes zero arguments. We remark that the η -rule $u = \mathbf{unit}$ implies that all elements of the unit type are equal.

Coproduct Types

$\frac{}{\Gamma \vdash A, B \mathit{type}_\ell}$	$\frac{}{\Gamma \vdash \mathit{inl} a : A \uplus B}$	$\frac{}{\Gamma \vdash \mathit{inr} b : A \uplus B}$
COPROD:ELIM $\Gamma, z : A \uplus B \vdash T \mathit{type}$ $\Gamma, x : A \vdash t_{\mathit{inl}} : T[\mathit{inl} x/z]$ $\Gamma, y : B \vdash t_{\mathit{inr}} : T[\mathit{inr} y/z]$ $\Gamma \vdash c : A \uplus B$ <hr style="width: 100%;"/> $\Gamma \vdash t := \mathit{case} c \mathit{of} \left\{ \begin{array}{ll} \mathit{inl} x & \mapsto t_{\mathit{inl}} \\ \mathit{inr} y & \mapsto t_{\mathit{inr}} \end{array} \right\} : T[c/z]$ <p style="text-align: center;">where $t[\mathit{inl} a/c] = t_{\mathit{inl}}[a/x]$ (COPROD:INL:BETA)</p> <p style="text-align: center;">$t[\mathit{inr} b/c] = t_{\mathit{inr}}[b/x]$ (COPROD:INR:BETA)</p>		

Figure 3.6: Typing rules for the coproduct type in DTT. Substitution rules are omitted.

The coproduct type $A \uplus B$ (fig. 3.6) is a data type with two constructors $\mathit{inl} : A \rightarrow A \uplus B$ and $\mathit{inr} : B \rightarrow A \uplus B$. The eliminator allows the creation of

⁴The name comes from the alternative notation $\Sigma(x : A).B$.

functions $(z : A \uplus B) \rightarrow T z$ by handling two cases: z may be of the form $\text{inl } x$ or $\text{inr } y$, so we need to provide functions $(x : A) \rightarrow T(\text{inl } x)$ and $(y : B) \rightarrow T(\text{inr } y)$. Note that in the β -rule, the notation $t[\text{inl } a/c]$ is not a valid substitution (since c is a term and not a variable) but instead an ad hoc abbreviation for

$$\text{case inl } a \text{ of } \left\{ \begin{array}{l} \text{inl } x \mapsto t_{\text{inl}} \\ \text{inr } y \mapsto t_{\text{inr}} \end{array} \right\},$$

and similar for $t[\text{inr } b/c]$.

Notation 3.2.17 (Pattern matching). When an eliminator of a data type is applied to a variable, we will often abbreviate this by directly writing a pattern where this variable is bound. For example, the following definition of $h : (z : A \uplus B) \rightarrow T z$:

$$h(\text{inl } x) = f x$$

$$h(\text{inr } y) = g y$$

is an abbreviation of

$$h z = \text{case } z \text{ of } \left\{ \begin{array}{l} \text{inl } x \mapsto f x \\ \text{inr } y \mapsto g y \end{array} \right\}.$$

Empty Type

$\frac{\text{EMPTY}}{\Gamma \vdash \text{Empty type}_\ell}$	$\frac{\text{EMPTY:ELIM} \quad \Gamma, x : \text{Empty} \vdash T \text{ type} \quad \Gamma \vdash e : \text{Empty}}{\Gamma \vdash \text{case } e \text{ of } \{\} : T[e/x]}$
---	--

Figure 3.7: Typing rules for the empty type in DTT. Substitution rules are omitted.

The empty type **Empty** (fig. 3.7) is a data type with zero constructors. The eliminator allows the creation of functions $(x : \text{Empty}) \rightarrow T x$ by handling zero cases.

Identity Types

DTT typically features an identity type $a \equiv_A b$ (fig. 3.8). This comes with some special terminology: if $a \equiv_A b$ is inhabited, then we say that a and b are

General rules:

$$\begin{array}{c}
\text{ID} \\
\frac{\Gamma \vdash A \text{ type}_\ell \quad \Gamma \vdash a, b : A}{\Gamma \vdash a \equiv_A b \text{ type}_\ell} \\
\text{ID:INTRO} \\
\frac{\Gamma \vdash a : A}{\Gamma \vdash \text{refl}_a : a \equiv_A a}
\end{array}$$

Extensional identity rules:

$$\begin{array}{c}
\text{ID:REFLECTION} \\
\frac{\Gamma \vdash e : a \equiv_A b}{\Gamma \vdash a = b : A} \\
\text{ID:ETA} \\
\frac{\Gamma \vdash e : a \equiv_A b}{\Gamma \vdash e = \text{refl}_a : a \equiv_A a}
\end{array}$$

Intensional identity rules:*Eliminator (J-rule):*

$$\begin{array}{c}
\text{ID:J} \\
\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : A \quad \Gamma, y : A, w : a \equiv_A y \vdash T \text{ type} \quad \Gamma \vdash e : a \equiv_A b \quad \Gamma \vdash t_{\text{refl}} : T[a/y, \text{refl}_a/w]}{\Gamma \vdash t := \text{case } (b, e) \text{ of } \{(a, \text{refl}_a) \mapsto t_{\text{refl}}\} : T[b/y, e/w]} \\
\text{where } t[a/b, \text{refl}_a/e] = t_{\text{refl}} \quad (\text{ID:J:BETA})
\end{array}$$

Function extensionality:

$$\begin{array}{c}
\text{PI:FUNEXT} \\
\frac{\Gamma \vdash f, g : (x : A) \rightarrow B \quad \Gamma \vdash h : (x : A) \rightarrow f x \equiv_B g x}{\Gamma \vdash \text{funext } h : f \equiv_{(x:A) \rightarrow B} g}
\end{array}$$

Judgemental uniqueness of identity proofs (UIP):

$$\begin{array}{c}
\text{ID:UIP} \\
\frac{\Gamma \vdash e_1, e_2 : a \equiv_A b}{\Gamma \vdash e_1 = e_2 : a \equiv_A b}
\end{array}$$

Figure 3.8: Typing rules for the identity type in DTT. Substitution rules are omitted.

propositionally equal, and we call the inhabitants **equality proofs**. If the judgement $a = b : A$ is derivable, then we say that a and b are **judgementally equal**.⁵ We also use this terminology for other equality judgements, e.g. for $A = B$ type. There are two versions of the identity type.

Extensional identity type The extensional identity type can be seen as a codata type, which has a single eliminator (called the **reflection** rule) that deduces judgemental equality from a propositional equality proof. One expects that the extensional identity type, as other codata types, has a single constructor which only requires that we provide content to the eliminator. In this case, one would expect that judgemental equality is required:

$$\frac{\Gamma \vdash a = b : A}{\Gamma \vdash \text{refl}_{a,b} : a \equiv_A b} \quad (3.10)$$

Actually, this constructor is equally powerful as refl_a . Indeed, we can mutually define

$$\text{refl}_a := \text{refl}_{a,a}, \quad \text{refl}_{a,b} := \text{refl}_a. \quad (3.11)$$

The former is well-typed because judgemental equality is an equivalence relation by definition of a GAT. The latter makes sense because, if we can write $\text{refl}_{a,b}$, then we know that $a = b$ judgementally, so that the type $a \equiv_A a$ of refl_a is judgementally equal to the required type $a \equiv_A b$. An extensional identity type can be equipped with an η -rule: if $e : a \equiv_A b$, then we can eliminate and deduce that $a = b$ judgementally, and we can reconstruct $\text{refl}_a : a \equiv_A b$. The η -rule asserts that $e = \text{refl}_a$. There is no β -rule as the eliminator does not produce a term.

It is clear that the constructor states that judgemental equality implies propositional equality, whereas the eliminator states that propositional equality implies judgemental equality. The η -rule expresses that propositional equality is proof-irrelevant (all proofs are the same), so in extensional DTT (DTT with an extensional identity type) we need not distinguish between judgemental and propositional equality.

Intensional identity type Unless otherwise mentioned, in this thesis we always use DTT with an intensional identity type.

The intensional identity type is a data type *family*. That is, for any type A and element $a : A$ (these are the **parameters** of the type family), we get a family

⁵Seeing DTT as a GAT, recall that the equality judgement in fact denotes the congruence closure of all axioms, i.e. all typing rules ending in an equality judgement. Hence, judgemental equality is reflexive, transitive, and respected by everything.

of types $a \equiv_A y$ depending on $y : A$ (this is the **index** of the type family). For every choice of parameters of a type family, we get a set of constructors, and each constructor gets to choose the value of the indices depending on its inputs. For the identity type, given parameters A and a , we get a single zero-argument constructor $\text{refl}_a : a \equiv_A a$ which chooses y to be a .

When eliminating, we need to pattern-match simultaneously on the element of the data type and its indices. The eliminator allows us to create functions $(y : A) \rightarrow (w : a \equiv_A y) \rightarrow T y w$ by handling the case where $w = \text{refl}_a$ and therefore $y = a$, i.e. by providing just an element of $T a \text{refl}_a$. The β -rule simply states that we actually resort to that element when $w = \text{refl}_a$. There is no η -rule: we cannot equate an arbitrary equality proof $e : a \equiv_A b$ to refl_a , because this may not be well-typed as we cannot deduce judgemental equality from propositional equality.

Remark 3.2.18. One easily shows that $a \equiv_A b$ is isomorphic to $b \equiv_A a$, so that we will also take the liberty to pattern-match on (a, e) where $e : a \equiv_A b$.

In intensional DTT (DTT with an intensional identity type), judgemental equality is stronger than propositional equality.

The intensional identity type has a few annoying properties:

- We cannot deduce propositional **function extensionality**: the statement that if functions are pointwise propositionally equal, then they are propositionally equal [BPT17]. This property is however satisfied by many denotational models and also extremely useful, so we postulate it as an axiom.
- Mathematicians born into set theory who only later discovered type theory, may balk at the notion of having to prove equality of equality proofs. We comfort them by postulating that all equality proofs are definitionally equal. This endows our internal types with the structure of a setoid (a set equipped with an equivalence relation): judgemental equality has the meaning of actual equality, whereas propositional equality serves as the equivalence relation.

If we disable the uniqueness of identity proofs (UIP) rule, then we open the door to groupoid and ∞ -groupoid models and homotopy type theory (HoTT) [HS94; Uni13].

We remark that Sterling et al.'s extensional type theory XTT [SAG19] satisfies both function extensionality and judgemental uniqueness of identity proofs *without blocking computation* in the sense that the system satisfies canonicity: every closed boolean is either **true** or **false**.

Booleans

Booleans are not very exciting: they could be implemented as $\text{Unit} \uplus \text{Unit}$. However, for reasons of readability, we add a type Bool type_ℓ with constructors $\text{true} : \text{Bool}$ and $\text{false} : \text{Bool}$ and we write the case expression as $\text{if } b \text{ } t_{\text{true}} \text{ } t_{\text{false}}$.

Natural numbers

$$\begin{array}{l}
 \text{NAT:ELIM} \\
 \Gamma, v : \text{Nat} \vdash T \text{ type} \\
 \Gamma \vdash t_0 : T[0/v] \\
 \Gamma, v : \text{Nat}, s : T[v/v] \vdash t_{\text{suc}} : T[\text{suc } v/v] \\
 \Gamma \vdash n : \text{Nat} \\
 \hline
 \Gamma \vdash t := \text{case } n \text{ of } \left\{ \begin{array}{ll} 0 & \mapsto t_0 \\ \text{suc}(v \mapsto s) & \mapsto t_{\text{suc}} \end{array} \right\} : T[n/v] \\
 \text{where } t[0/n] = t_0 \\
 \quad t[\text{suc } m/n] = t_{\text{suc}}[m/v, t[m/n]/s]
 \end{array}$$

Figure 3.9: Elimination rule for the naturals in DTT.

The natural numbers are a data type Nat type_ℓ with constructors $0 : \text{Nat}$ and $\text{suc} : \text{Nat} \rightarrow \text{Nat}$. The elimination rule is given in fig. 3.9. It requires us to handle both constructors; in the case of suc , we are provided not only with the predecessor v , but also with a variable s that stands for the same case expression applied to the predecessor.

Universes

We extend DTT with universes à la Coquand [Coq13] (fig. 3.10). The universe U_ℓ of level $\ell \in \mathbb{N}$ (we could also consider levels higher than ω , but in practice you usually have either enough natural numbers for a practical purpose or too few ordinal numbers to attain generality), has a single **decoding** eliminator El that produces a type of level ℓ . Hence, the **encoding** constructor requires as argument a type of level ℓ . The β - and η -rules simply state that these operations are mutually inverse.

The advantage of universes à la Coquand is that they automatically inherit all constructions applicable to the typing judgement, simply by decoding, applying

$\frac{\text{UNI} \quad \Gamma \text{ctx} \quad \ell \in \mathbb{N}}{\Gamma \vdash \mathbf{U}_\ell \text{type}_{\ell+1}}$	$\frac{\text{UNI:INTRO} \quad \Gamma \vdash T \text{type}_\ell}{\Gamma \vdash \ulcorner T \urcorner : \mathbf{U}_\ell}$ <p style="text-align: center; margin: 0;">where $A = \ulcorner \text{El } A \urcorner : \mathbf{U}_\ell$ (UNI:ETA)</p>	$\frac{\text{UNI:ELIM} \quad \Gamma \vdash A : \mathbf{U}_\ell}{\Gamma \vdash \text{El } A \text{type}_\ell}$ <p style="text-align: center; margin: 0;">where $\text{El } \ulcorner T \urcorner = T \text{type}_\ell$ (UNI:BETA)</p>
---	---	---

Figure 3.10: Typing rules for universes in DTT. Substitution rules are omitted.

the construction, and re-encoding. In particular, we have explicit cumulativity coercions $\mathbf{U}_k \rightarrow \mathbf{U}_\ell$ for $k \leq \ell$.

Notation 3.2.19. We will omit applications of El and $\ulcorner _ \urcorner$.

3.2.5 An Auxiliary Definition

We needed to put the following definition somewhere:

Definition 3.2.20. The type of **isomorphisms** $A \cong B$ is defined as

$$\begin{aligned} (A \cong B) := & (f : A \rightarrow B) \times (g : B \rightarrow A) \\ & \times (g \circ f \equiv_{A \rightarrow A} \text{id}_A) \times (f \circ g \equiv_{B \rightarrow B} \text{id}_B). \end{aligned}$$

3.3 The Curry-Howard Correspondence

The judgement forms of DTT only allow us to say that something is a context, something is a type, something is a term of a certain type, and that contexts, terms and types are equal. This may create the impression that DTT is unusable for logic. However, the contrary is true: the Curry-Howard correspondence lets us encode propositions proof-relevantly as dependent types. By translating every logical operator to an operation on DTT types, we can inductively translate any logical formula to the type of its proofs. Conversely, and more boringly, every type T can be read as the proposition ‘ T is inhabited’.

So in type theory, to state a theorem T is to create a type $T \text{type}$, and to prove the theorem T is to create an element $t : T$. This means that the

above ‘inductive translation’ in type theory is effectively the identity function: the logical operators that we use in type theory *are* the type formers that translate them to types, so the translation function does nothing. For this reason, some people object to the often used term ‘Curry-Howard isomorphism’: the isomorphism is the identity. The translation is only necessary to translate theorems from classical logic to type theory, and to teach people familiar with classical logic how to do logic in type theory.

3.3.1 Encoding Propositions as Types

We shall now explain how to translate logical operators and related proving techniques to type theory. The introduction rules of the type correspond to ways of proving the proposition, whereas the elimination rules correspond to ways of using knowledge.

\top Logical truth is encoded as the **Unit** type.

Proving Proving truth is trivial. We have $\text{unit} : \text{Unit}$.

Using A proof of truth is useless. **Unit** has no eliminators.

\perp Logical falsehood is encoded as the **Empty** type.

Proving Proving falsehood is impossible (unless we start from contradictory assumptions). **Empty** has no constructors, so we have to eliminate the context until there are zero cases to complete.

Using Falsehood implies anything. The eliminator for **Empty** (fig. 3.7) allows us to prove anything.

\wedge The logical conjunction $P \wedge Q$ (P and Q) is encoded as the product type $P \times Q$.

Proving The conjunction is proven by proving P and Q . The proofs $p : P$ and $q : Q$ can be paired up as $(p, q) : P \times Q$.

Using The conjunction implies P and implies Q . From a proof $r : P \times Q$ we get $\text{fst } r : P$ and $\text{snd } r : Q$.

\vee The logical disjunction $A \vee B$ (A or B) is encoded as the coproduct $A \uplus B$.

Proving The disjunction is proven either by proving P or by proving Q . Indeed, given $p : P$ we get $\text{inl } p : P \uplus Q$ and similar for $q : Q$.

Using If we know that $P \vee Q$ holds, then we can reason by case distinction: if P holds, then ..., if Q holds, then This corresponds to eliminating $r : P \uplus Q$ (fig. 3.6).

\Rightarrow Logical implication $P \Rightarrow Q$ is encoded as the function type $P \rightarrow Q$.

Proving In order to prove $P \Rightarrow Q$, we assume P and prove Q .
Abstraction $\lambda(x : P).q$ brings a proof $x : P$ in scope, which we can use in the proof $q : Q$.

Using Modus ponens translates to application:

$$\frac{P \Rightarrow Q \quad P}{Q} \quad \frac{\Gamma \vdash f : P \rightarrow Q \quad \Gamma \vdash p : P}{\Gamma \vdash fp : Q}.$$

\neg Logical negation $\neg P$ is encoded as $P \rightarrow \text{Empty}$.

Proving We can prove $\neg P$ from the absurd: we assume P and derive a contradiction. This is achieved by λ -abstraction: $\lambda(x : P).e$ brings a proof $x : P$ in scope, which we can use in the proof $e : \text{Empty}$.

Using If P and $\neg P$ are both true, then we can deduce \perp . This again translates to application.

$=$ Equality $a = b \in A$ is encoded as $a \equiv_A b$.

Proving Equality is reflexive: $\text{refl}_a : a \equiv_A a$.

Using If we know that $a = b \in A$, then we can henceforth replace a with b . Both the reflection rule and the eliminator of the intensional identity type (fig. 3.8) allows us to do this.

\forall Universal quantification $\forall(x \in A).P(x)$ is encoded as $(x : A) \rightarrow P(x)$. Here, P is a proposition (i.e. a type) depending on $x : A$.

Proving We need to pick an arbitrary $x \in A$ and prove $P(x)$. This is done by λ -abstraction.

Using If we have $a \in A$, then we can conclude $P(a)$. This translates to application.

\exists Existential quantification $\exists(x \in A).P(x)$ is encoded as $(x : A) \times P(x)$.

Proving We need to find some $a \in A$ satisfying $P(a)$. The element $a : A$ and the proof $p : P(a)$ can be paired up as $(a, p) : (x : A) \times P(x)$.

Using We get to use some $a \in A$ satisfying $P(a)$. Indeed, from a proof $c : (x : A) \times P(x)$, we can extract $\text{fst } c : A$ and $\text{snd } c : P(\text{fst } c)$.

3.3.2 Constructivity

We point out that proofs in DTT are constructive. For example, from a proof of $\exists(x \in A).P(x)$, we can extract an element $a \in A$ satisfying $P(a)$. In practice, this element a is a dependently typed program that can be computed to actually find a normal form of type A (though we do not treat normalization in this thesis).

In general, when we make a claim in DTT, we are not merely saying that the claim is true, but that it is constructively provable. When we negate a claim, we are not merely saying that it is false, but that it is refutable.

This leads to some differences with classical logic. In classical logic, for any proposition P , we know $P \vee \neg P$ (law of excluded middle) and $P \Leftrightarrow \neg\neg P$ (principle of double negation). In DTT, we can prove that for all propositions P , we have $P \Rightarrow \neg\neg P$ (provability implies refutability of refutability) in the following sense:

$$\lambda P.\lambda p.\lambda p^*.p^* p : (P : \mathbf{U}) \rightarrow P \rightarrow ((P \rightarrow \mathbf{Empty}) \rightarrow \mathbf{Empty}).$$

However, parametric models of DTT [e.g. AGJ14] prove that there are no general functions

$$(P : \mathbf{U}) \rightarrow ((P \rightarrow \mathbf{Empty}) \rightarrow \mathbf{Empty}) \rightarrow P, \quad (3.12)$$

$$(P : \mathbf{U}) \rightarrow P \uplus (P \rightarrow \mathbf{Empty}). \quad (3.13)$$

This means that DTT provides some nuances that are absent in classical logic:

- The proposition $P \uplus (P \rightarrow \mathbf{Empty})$ is read as ‘ P is decidable’: to prove it is to write a terminating functional program that computes to either $\text{inl } p$, which contains a proof p of P , or $\text{inr } p^*$, which contains a refutation p^* .
- The proposition $(P \rightarrow \mathbf{Empty}) \rightarrow \mathbf{Empty}$ is read as ‘ P holds non-constructively’. As there can be at most one function to the empty type, all proofs of this proposition are equal and devoid of content, so we cannot extract information.

We remark that it is easy to prove (by writing terms of DTT) that the general statements in eqs. (3.12) and (3.13) are logically equivalent, i.e. there are functions in both directions which do not necessarily constitute an isomorphism. Postulating either statement essentially amounts to assuming the axiom of choice: if we know that a type P is not empty, then we can obtain a concrete construction of type P from thin air. Of course no computer or algorithm is capable of computing these constructions in general, so these axioms cannot be endowed with computational behaviour and postulating them makes DTT non-constructive.

Chapter 4

Presheaf Models of Dependent Type Theory

In this chapter, we explain how the presheaf category $\mathbf{Psh}(\mathcal{W})$ over an arbitrary base category \mathcal{W} constitutes a model of DTT with all type formers listed in section 3.2.4. The entire content of this chapter is based on Hofmann’s work [Hof97], except for the construction of the universe, which is due to Hofmann and Streicher [HS97]. The notations are polished from prior work [Nuy18a].

We remind the reader of notation 3.1.12: we will use the syntax and judgements of DTT directly to act on and to make claims about models, omitting the use of interpretation brackets.

Readers will also want to learn about our presheaf notations (notation 2.3.2).

4.1 Modelling Structural Rules: Presheaf Categories are CwFs

We start by modelling the GAT specified by the structural rules of DTT (sections 3.2.1 and 3.2.2), i.e. by explaining how any presheaf category $\mathbf{Psh}(\mathcal{W})$ can be endowed with the structure of a CwF. We will postpone any discussion of sizes and universe levels to section 4.2.

4.1.1 Category of Contexts

As category of contexts, of course, we pick $\mathbf{Psh}(\mathcal{W})$. Thus, a context is a presheaf $\Gamma : \mathcal{W}^{\text{op}} \rightarrow \mathbf{Set}$, and a substitution is a presheaf morphism (natural transformation) $\Gamma \rightarrow \Delta$.

4.1.2 Types Functor

We need to define the functor $\text{Ty} : \mathbf{Psh}(\mathcal{W})^{\text{op}} \rightarrow \mathbf{Set}$.

Set of types We start by defining the action of Ty on an object Γ , i.e. the meaning of the judgement $\Gamma \vdash T$ **type**, as the set of presheaves over the category of elements \mathcal{W}/Γ :

$$\text{Ty}(\Gamma) := \text{Obj}(\mathbf{Psh}(\mathcal{W}/\Gamma)). \quad (4.1)$$

Concretely, a type $\Gamma \vdash T$ **type** consists of:

- For every cell $\gamma : W \rightrightarrows \Gamma$ (yielding an object (W, γ) of \mathcal{W}/Γ to which we must be able to apply T), a set $((W, \gamma) \rightrightarrows T)$ of cells over γ which we will denote as $(W \triangleright T[\gamma])$ in order to stay closer to the intuitions of DTT. We will write $W \triangleright t : T[\gamma]$ to say that t is an element of this set.
- For every morphism $\varphi : V \rightarrow W$ and every cell $\gamma : W \rightrightarrows \Gamma$ (hence for every morphism $\varphi : (V, \gamma \circ \varphi) \rightarrow (W, \gamma)$ in the category of elements) a restriction map which we will denote $\sqsubset(\varphi) : (W \triangleright T[\gamma]) \rightarrow (V \triangleright T[\gamma \circ \varphi])$.

Type substitution The action of Ty on a substitution $\sigma : \Gamma \rightarrow \Delta$ defines the semantics of type substitution. We simply define this by precomposing $T : (\mathcal{W}/\Delta)^{\text{op}} \rightarrow \mathbf{Set}$ with $\Sigma/\sigma : (W, \gamma) \mapsto (W, \sigma \circ \gamma)$ (definition 2.2.37). Concretely, $T[\sigma]$ is defined as follows:

- For $\gamma : W \rightrightarrows \Gamma$, we have $(W \triangleright T[\sigma][\gamma]) = (W \triangleright T[\sigma \circ \gamma])$,
- The restriction of a cell $(W \triangleright t : T[\sigma][\gamma])$ along $\varphi : V \rightarrow W$ is simply given by the restriction in T , yielding $(V \triangleright t(\varphi) : T[\sigma][\gamma \circ \varphi])$, which is well-typed by naturality of σ .

Example 4.1.1 (Sets). Continuing example 2.3.7, a context in the CwF $\mathbf{Psh}(\mathbf{Point})$ of sets is just a set Γ . A type $\Gamma \vdash T$ **type** is just a function $T : \Gamma \rightarrow \mathbf{Set}$. Substitution is given by composition.

Example 4.1.2 (Reflexive graphs). Continuing example 2.3.8, a context in the CwF $\mathbf{Psh}(\mathbf{RG})$ of reflexive graphs, is just a reflexive graph Γ . A type $\Gamma \vdash T$ **type** consists of:

- For every node $\gamma : \mathbb{N} \Rightarrow \Gamma$ a set $(\mathbb{N} \triangleright T[\gamma])$ of nodes over γ ,
- For every edge $\gamma : \mathbb{I} \Rightarrow \Gamma$ a set $(\mathbb{I} \triangleright T[\gamma])$ of edges over γ ,
- A source map sending $(\mathbb{I} \triangleright t : T[\gamma])$ to $(\mathbb{N} \triangleright t\langle \mathbf{s} \rangle : T[\gamma \circ \mathbf{s}])$,
- A target map sending $(\mathbb{I} \triangleright t : T[\gamma])$ to $(\mathbb{N} \triangleright t\langle \mathbf{t} \rangle : T[\gamma \circ \mathbf{t}])$,
- A reflexivity map sending $(\mathbb{N} \triangleright t : T[\gamma])$ to $(\mathbb{I} \triangleright t\langle \mathbf{r} \rangle : T[\gamma \circ \mathbf{r}])$, such that $t\langle \mathbf{r} \rangle\langle \mathbf{s} \rangle = t\langle \mathbf{r} \rangle\langle \mathbf{t} \rangle = t$.

It is worth remarking that, given a node $\gamma : \mathbb{N} \Rightarrow \Gamma$, the nodes above γ and the edges above $\gamma \circ \mathbf{r}$ constitute a reflexive graph:

$$\begin{array}{ccc}
 & \curvearrowleft \langle \mathbf{s} \rangle & \\
 & \text{---} & \\
 (\mathbb{N} \triangleright T[\gamma]) & \xrightarrow{\quad \langle \mathbf{r} \rangle \quad} & (\mathbb{I} \triangleright T[\gamma \circ \mathbf{r}]) \\
 & \text{---} & \\
 & \curvearrowright \langle \mathbf{t} \rangle &
 \end{array}$$

The nodes/edges of $T[\sigma]$ over γ are the nodes/edges of T over $\sigma \circ \gamma$.

Example 4.1.3 (Topos of trees). Continuing example 2.3.14, a type $\Gamma \vdash T$ type in the topos of trees consists of:

- For every $\gamma : i \Rightarrow \Gamma$ a set $(i \triangleright T[\gamma])$,
- Whenever $i \leq j$, a restriction map sending $(j \triangleright t : T[\gamma])$ to $(i \triangleright t\langle i^j \rangle : T[\gamma \circ i^j])$, and these maps commute.

4.1.3 Terms Functor

We need to define the functor $\text{Tm} : \text{Psh}(\mathcal{W}/\mathcal{T}\mathbf{y})^{\text{op}} \rightarrow \text{Set}$.

Set of terms We start by defining the action of Tm on an object (Γ, T) , i.e. the meaning of the judgement $\Gamma \vdash t : T$, as the following set:

$$\text{Tm}(\Gamma, T) \quad := \quad \forall W. (\gamma : W \Rightarrow \Gamma) \rightarrow (W \triangleright T[\gamma]) \quad (4.2)$$

and we denote the action of t on (W, γ) as $t[\gamma]$, i.e. if $\Gamma \vdash t : T$ and $\gamma : W \Rightarrow \Gamma$, then $W \triangleright t[\gamma] : T[\gamma]$. Note that the \forall -symbol denotes a dependent end, i.e. we have a naturality condition which requires that for $\varphi : V \rightarrow W$, we get $t[\gamma]\langle \varphi \rangle = t[\gamma \circ \varphi]$.

Term substitution Given $\sigma : \Gamma \rightarrow \Delta$ and $\Delta \vdash T$ type, we need to define term substitution $\sqsubset[\sigma] : \text{Tm}(\Delta, T) \rightarrow \text{Tm}(\Gamma, T[\sigma])$. This is done simply by precomposition, i.e. $t[\sigma][\gamma] := t[\sigma \circ \gamma]$.

Theorem 4.1.4. We have a natural isomorphism $(W \triangleright T[\gamma]) \cong (\mathbf{y}W \vdash T[\gamma])$.¹

Proof. This follows from expanding definitions and applying the dependent Yoneda-lemma (theorem 2.3.20):

$$\begin{aligned} (\mathbf{y}W \vdash T[\gamma]) &= \forall V. (\varphi : V \Rightarrow \mathbf{y}W) \rightarrow (V \triangleright T[\gamma \circ \varphi]) \\ &= \forall V. (\varphi : V \rightarrow W) \rightarrow (V \triangleright T[\gamma \circ \varphi]) \\ &\cong (W \triangleright T[\gamma]). \quad \square \end{aligned}$$

Example 4.1.5 (Sets). Continuing examples 2.3.7 and 4.1.1, a term $\Gamma \vdash t : T$ is just a function $(\gamma \in \Gamma) \rightarrow (* \triangleright T[\gamma])$.

Example 4.1.6 (Reflexive graphs). Continuing examples 2.3.8 and 4.1.2, a term $\Gamma \vdash t : T$ consists of:

- an action on nodes, sending $\gamma : \mathbb{N} \Rightarrow \Gamma$ to $\mathbb{N} \triangleright t[\gamma] : T[\gamma]$,
- an action on edges, sending $\gamma : \mathbb{I} \Rightarrow \Gamma$ to $\mathbb{I} \triangleright t[\gamma] : T[\gamma]$,
- such that source, target and reflexivity are respected:

$$t[\gamma]\langle \mathbf{s} \rangle = t[\gamma \circ \mathbf{s}], \quad t[\gamma]\langle \mathbf{t} \rangle = t[\gamma \circ \mathbf{t}], \quad t[\gamma]\langle \mathbf{r} \rangle = t[\gamma \circ \mathbf{r}].$$

Example 4.1.7 (Topos of trees). Continuing examples 2.3.14 and 4.1.3, a term $\Gamma \vdash t : T$ consists of:

- actions sending $\gamma : i \Rightarrow \Gamma$ to $i \triangleright t[\gamma] : T[\gamma]$,
- such that restriction is respected: $t[\gamma]\langle i'_i \rangle = t[\gamma \circ i'_i]$.

4.1.4 Empty Context and Context Extension

Presheaf categories have a terminal object $()$ which models the empty context, given by $(W \Rightarrow ()) = \{()\}$.

The extended context $\Gamma.T$ is the presheaf defined by:

$$\begin{aligned} (W \Rightarrow \Gamma.T) &:= (\gamma : W \Rightarrow \Gamma) \times (W \triangleright T[\gamma]), \\ (\gamma, t) \circ \varphi &:= (\gamma \circ \varphi, t\langle \varphi \rangle). \end{aligned}$$

¹See notation 2.3.18 for the use of γ .

Then we can pair up a substitution $\sigma : \Gamma \rightarrow \Delta$ and a term $\Gamma \vdash t : T[\sigma]$ to a substitution $(\sigma, t) : \Gamma \rightarrow \Delta.T$ by defining $(\sigma, t) \circ \gamma := (\sigma \circ \gamma, t[\gamma])$. Conversely, we define $\pi : \Gamma.T \rightarrow \Gamma$ by $\pi \circ (\gamma, t) := \gamma$, and $\Gamma.T \vdash \xi : T[\pi]$ by $\xi[\gamma, t] := t$.

4.2 Universe Levels for Size Stratification

We assume that we want, in our type system, ω universe levels. If another number is desired, one can replace without trouble.

Then as category of contexts, we should take the category of presheaves $\mathcal{W}^{\text{op}} \rightarrow \mathbf{Set}_\omega$ producing sets of size ω . The set $\text{Ty}_\ell(\Gamma)$ of types of level ℓ is simply given by the set of presheaves $(\mathcal{W}/\Gamma)^{\text{op}} \rightarrow \mathbf{Set}_\ell$ producing sets of size ℓ . The set $\text{Ty}(\Gamma)$ of types of arbitrary size is given by the set of presheaves $(\mathcal{W}/\Gamma)^{\text{op}} \rightarrow \mathbf{Set}_\omega$.

4.3 Type Formers

We will now describe how to model the type formers listed in section 3.2.4. We only give the construction of the relevant operators, without always proving that β -, η - and substitution rules are satisfied. The reader can either check this for themselves or consult Hofmann and Streicher's work [Hof97; HS97]. We will also omit considerations of size except for the universe.

4.3.1 Π -types

We need to define the type $\Gamma \vdash \Pi AB$ type (notation 3.2.14), i.e. for any $\gamma : W \Rightarrow \Gamma$ we need to define $(W \triangleright (\Pi AB)[\gamma])$, contravariantly in $(W, \gamma) \in \mathcal{W}/\Gamma$.

By theorem 4.1.4, this is isomorphic to the set $(\mathbf{y}W \vdash (\Pi AB)[\gamma]) = (\mathbf{y}W \vdash \Pi(A[\gamma])(B[\gamma+]))$, which by proposition 3.2.13 is isomorphic to $(\mathbf{y}W.A[\gamma] \vdash B[\gamma+])$. So we *must* define

$$\begin{aligned} (W \triangleright (\Pi AB)[\gamma]) &:= (\mathbf{y}W.A[\gamma] \vdash B[\gamma+]) \\ &\cong \forall V. (\varphi : V \rightarrow W) \rightarrow (V \triangleright A[\gamma\varphi]) \rightarrow (V \triangleright B[\gamma\varphi, a]). \end{aligned}$$

In line with notation 2.0.1, we write $\lambda : (\mathbf{y}W.A[\gamma] \vdash B[\gamma+]) \rightarrow (W \triangleright (\Pi AB)[\gamma])$. Restriction is given by $(\lambda b)\langle \varphi \rangle := \lambda(b[\mathbf{y}\varphi+])$.

We define $(\lambda b)[\gamma] := \lambda(b[\gamma+])$.

Example 4.3.1 (Sets). Continuing examples 2.3.7, 4.1.1 and 4.1.5, an element of $* \triangleright \lambda b : (\Pi AB)[\gamma]$ is a term $\mathbf{y}*.A[\gamma] \vdash b : B[\gamma+]$, i.e. a function sending $* \triangleright a : A[\gamma]$ to $* \triangleright b[\text{id}_*, a] : B[\gamma, x]$.

Example 4.3.2 (Reflexive graphs). Continuing examples 2.3.8, 4.1.2 and 4.1.6, a node $\mathbb{N} \triangleright \lambda b : (\Pi AB)[\gamma]$ is a term $\mathbf{y}\mathbb{N}.A[\gamma] \vdash b : B[\gamma+]$, which consists of:

- An action $b[\text{id}_{\mathbb{N}}, \sqcup]$ on nodes above γ :

$$\mathbf{y}\mathbb{N} \triangleright a : A[\gamma] \quad \mapsto \quad \mathbf{y}\mathbb{N} \triangleright b[\text{id}_{\mathbb{N}}, a] : B[\gamma, a],$$

- An action $b[\mathbf{r}, \sqcup]$ on edges above $\gamma \circ \mathbf{r}$:

$$\mathbf{y}\mathbb{I} \triangleright a : A[\gamma \circ \mathbf{r}] \quad \mapsto \quad \mathbf{y}\mathbb{I} \triangleright b[\mathbf{r}, a] : B[\gamma \circ \mathbf{r}, a],$$

- Respecting source, target and reflexivity:

$$b[\mathbf{r}, a]\langle \mathbf{s} \rangle = b[\text{id}_{\mathbb{N}}, a\langle \mathbf{s} \rangle],$$

$$b[\mathbf{r}, a]\langle \mathbf{t} \rangle = b[\text{id}_{\mathbb{N}}, a\langle \mathbf{t} \rangle], \quad b[\text{id}_{\mathbb{N}}, a]\langle \mathbf{r} \rangle = b[\mathbf{r}, a\langle \mathbf{r} \rangle].$$

Meanwhile, an edge $\mathbb{I} \triangleright \lambda b : (\Pi AB)[\gamma]$ is a term $\mathbf{y}\mathbb{I}.A[\gamma] \vdash b : B[\gamma+]$, which consists of:

- An action $b[\text{id}_{\mathbb{I}}, \sqcup]$ on edges above γ :

$$\mathbf{y}\mathbb{I} \triangleright a : A[\gamma] \quad \mapsto \quad \mathbf{y}\mathbb{I} \triangleright b[\text{id}_{\mathbb{I}}, a] : B[\gamma, a],$$

- An action $b[\mathbf{s}, \sqcup]$ on nodes above the source $\gamma \circ \mathbf{s}$ of γ :

$$\mathbf{y}\mathbb{N} \triangleright a : A[\gamma \circ \mathbf{s}] \quad \mapsto \quad \mathbf{y}\mathbb{N} \triangleright b[\mathbf{s}, a] : B[\gamma \circ \mathbf{s}, a],$$

- An action $b[\mathbf{t}, \sqcup]$ on nodes above the target $\gamma \circ \mathbf{t}$ of γ ,

- An action $b[\mathbf{s} \circ \mathbf{r}, \sqcup]$ on edges above the reflexive edge $\gamma \circ \mathbf{s} \circ \mathbf{r}$ on the source of γ :

$$\mathbf{y}\mathbb{I} \triangleright a : A[\gamma \circ \mathbf{s} \circ \mathbf{r}] \quad \mapsto \quad \mathbf{y}\mathbb{I} \triangleright b[\mathbf{s} \circ \mathbf{r}, a] : B[\gamma \circ \mathbf{s} \circ \mathbf{r}, a],$$

- An action $b[\mathbf{t} \circ \mathbf{r}, \sqcup]$ on edges above the reflexive edge $\gamma \circ \mathbf{t} \circ \mathbf{r}$ on the target of γ ,

- Respecting source, target and reflexivity:

$$b[\text{id}_{\mathbb{I}}, a]\langle \mathbf{s} \rangle = b[\mathbf{s}, a\langle \mathbf{s} \rangle], \quad b[\mathbf{s}, a]\langle \mathbf{r} \rangle = b[\mathbf{s} \circ \mathbf{r}, a\langle \mathbf{r} \rangle],$$

$$b[\text{id}_{\mathbb{I}}, a]\langle \mathbf{t} \rangle = b[\mathbf{t}, a\langle \mathbf{t} \rangle], \quad b[\mathbf{t}, a]\langle \mathbf{r} \rangle = b[\mathbf{t} \circ \mathbf{r}, a\langle \mathbf{r} \rangle],$$

$$b[\mathbf{s} \circ \mathbf{r}, a]\langle \mathbf{s} \rangle = b[\mathbf{s}, a\langle \mathbf{s} \rangle], \quad b[\mathbf{t} \circ \mathbf{r}, a]\langle \mathbf{s} \rangle = b[\mathbf{t}, a\langle \mathbf{s} \rangle],$$

$$b[\mathbf{s} \circ \mathbf{r}, a]\langle \mathbf{t} \rangle = b[\mathbf{s}, a\langle \mathbf{t} \rangle], \quad b[\mathbf{t} \circ \mathbf{r}, a]\langle \mathbf{t} \rangle = b[\mathbf{t}, a\langle \mathbf{t} \rangle].$$

Given an edge λb in the Π -type above $\gamma : \mathbb{I} \Rightarrow \Gamma$, its source is $(\lambda b)\langle \mathbf{s} \rangle = \lambda(b[\mathbf{ys}+])$. Thus:

- The action on nodes is $b[\mathbf{ys}+][\text{id}_{\mathbb{N}}, \sqcup] = b[\mathbf{s}, \sqcup]$, i.e. the action of b on nodes above the source of γ ,
- The action on edges is $b[\mathbf{ys}+][\mathbf{r}, \sqcup] = b[\mathbf{s} \circ \mathbf{r}, \sqcup]$, i.e. the action of b on edges above the reflexive edge on the source of γ .

The target of λb is obtained in a similar way.

Given a node λb in the Π -type above $\gamma : \mathbb{N} \Rightarrow \Gamma$, its reflexive edge is $(\lambda b)\langle \mathbf{r} \rangle = \lambda(b[\mathbf{yr}+])$ and lives above $\gamma \circ \mathbf{r}$. Thus:

- The actions on edges are:

$$b[\mathbf{yr}+][\text{id}_{\mathbb{I}}, \sqcup] = b[\mathbf{r}, \sqcup],$$

$$b[\mathbf{yr}+][\mathbf{s} \circ \mathbf{r}, \sqcup] = b[\mathbf{r}, \sqcup],$$

$$b[\mathbf{yr}+][\mathbf{t} \circ \mathbf{r}, \sqcup] = b[\mathbf{r}, \sqcup],$$

i.e. they are all equal to the action of b on edges.

- The actions on nodes are:

$$b[\mathbf{yr}+][\mathbf{s}, \sqcup] = b[\mathbf{yr}+][\mathbf{t}, \sqcup] = b[\text{id}_{\mathbb{N}}, \sqcup],$$

i.e. they are both equal to the action of b on nodes.

Example 4.3.3 (Topos of trees). Continuing examples 2.3.14, 4.1.3 and 4.1.7, a cell $i \triangleright \lambda b : (\Pi AB)[\gamma]$ is a term $\mathbf{y}i.A[\gamma] \vdash b : B$, which consists of, for all $j \leq i$, an action $b[\uparrow_j^i, \sqcup]$ on j -cells of A above $\gamma \circ \uparrow_j^i$:

$$j \triangleright a : A[\gamma \circ \uparrow_j^i] \quad \mapsto \quad j \triangleright b[\uparrow_j^i, a] : B[\gamma \circ \uparrow_j^i, a].$$

such that restriction is respected: $b[\uparrow_j^i, a]\langle \uparrow_k^j \rangle = b[\uparrow_k^i, a]\langle \uparrow_k^j \rangle$.

Restriction of λb is given by $(\lambda b)\langle \uparrow_j^i \rangle = \lambda(b[(\mathbf{y} \uparrow_j^i)+])$ whose actions are given by $b[\mathbf{y} \uparrow_j^i+][\uparrow_k^j, \sqcup] = b[\uparrow_k^i, \sqcup]$.

4.3.2 Σ -types

We need to define the type $\Gamma \vdash \Sigma AB$ type, i.e. for any $\gamma : W \Rightarrow \Gamma$ we need to define $(W \triangleright (\Sigma AB)[\gamma])$, contravariantly in $(W, \gamma) \in \mathcal{W}/\Gamma$:

$$(W \triangleright (\Sigma AB)[\gamma]) := (W \triangleright a : A[\gamma]) \times (W \triangleright B[\gamma, a]). \quad (4.3)$$

The constructor and eliminators are defined by

$$(a, b)[\gamma] := (a[\gamma], b[\gamma]), \quad (\text{fst } c)[\gamma] := \pi_1(c[\gamma]), \quad (\text{snd } c)[\gamma] := \pi_2(c[\gamma]).$$

4.3.3 Identity types

We model the extensional identity type $\Gamma \vdash a \equiv_A b$, which trivially satisfies the typing rules of the intensional identity type including function extensionality and UIP. To this end, we need to define, for any $\gamma : W \Rightarrow \Gamma$, the set $(W \triangleright (a \equiv_A b)[\gamma])$, contravariantly in $(W, \gamma) \in \mathcal{W}/\Gamma$.

By theorem 4.1.4, this is isomorphic to the set $(\mathbf{y}W \vdash (a \equiv_A b)[\gamma]) = (\mathbf{y}W \vdash a[\gamma] \equiv_{A[\gamma]} b[\gamma])$. From the reflexivity and reflection rules and the η -rule, it is evident that the latter has a (unique) element if and only if $\mathbf{y}W \vdash a[\gamma] = b[\gamma] : A[\gamma]$, which by theorem 4.1.4 is equivalent to $W \triangleright a[\gamma] = b[\gamma] : A[\gamma]$. So we define:

$$(W \triangleright (a \equiv_A b)[\gamma]) := \{\text{refl} \mid W \triangleright a[\gamma] = b[\gamma] : A[\gamma]\}. \quad (4.4)$$

4.3.4 Universes

We need to define the type $\Gamma \vdash \mathbf{U}_\ell \text{type}_{\ell+1}$, i.e. for any $\gamma : W \Rightarrow \Gamma$ we need to define $(W \triangleright \mathbf{U}_\ell[\gamma]) \in \text{Set}_{\ell+1}$, contravariantly in $(W, \gamma) \in \mathcal{W}/\Gamma$.

By theorem 4.1.4, this is isomorphic to the set $(\mathbf{y}W \vdash \mathbf{U}_\ell[\gamma]) = (\mathbf{y}W \vdash \mathbf{U}_\ell)$, which via encoding/decoding is isomorphic to $(\mathbf{y}W \vdash \text{type}_\ell) = \text{Ty}_\ell(\mathbf{y}W)$ (notation 3.1.12). So we *must* define

$$(W \triangleright \mathbf{U}_\ell[\gamma]) := \text{Ty}_\ell(\mathbf{y}W) = \text{Obj}(\text{Psh}(\mathcal{W}/\mathbf{y}W)) \cong \text{Obj}(\text{Psh}(\mathcal{W}/W)), \quad (4.5)$$

which does not depend on γ , confirming that we have properly modelled the universe as a closed type. In line with notation 2.0.1, we write $\text{El} : (W \triangleright \mathbf{U}_\ell[\gamma]) \rightarrow \text{Ty}_\ell(\mathbf{y}W)$ with inverse $\ulcorner _ \urcorner$. Restriction is given by $\ulcorner T \urcorner \langle \varphi \rangle := \ulcorner T[\mathbf{y}\varphi] \urcorner$.

We define $\ulcorner T \urcorner[\gamma] := \ulcorner T[\gamma] \urcorner$. Hence, we have $(\text{El}A)[\gamma] = (\text{El}A)[\gamma][\text{id}] = (\text{El}(A[\gamma]))[\text{id}]$.

This construction is called the **Hofmann-Streicher universe** [HS97].

Example 4.3.4 (Sets). Continuing examples 2.3.7, 4.1.1, 4.1.5 and 4.3.1, an element $* \triangleright \ulcorner T \urcorner : \mathbf{U}[_]$ is a type $\mathbf{y}* \vdash T \text{type}$, i.e. a set.

Example 4.3.5 (Reflexive graphs). Continuing examples 2.3.8, 4.1.2, 4.1.6 and 4.3.2, a node $\mathbb{N} \triangleright \lceil T^\top : \mathbb{U}[_] \rceil$ is a type $\mathbf{y}\mathbb{N} \vdash T$ type, which is a diagram

$$\begin{array}{ccc}
 & \xleftarrow{\sqcup\langle \mathbf{s} \rangle} & \\
 (\mathbb{N} \triangleright T[\mathbf{id}_{\mathbb{N}}]) & \xrightarrow{\sqcup\langle \mathbf{r} \rangle} & (\mathbb{I} \triangleright T[\mathbf{r}]) \\
 & \xleftarrow{\sqcup\langle \mathbf{t} \rangle} &
 \end{array}$$

i.e. a reflexive graph.

An edge $\mathbb{N} \triangleright \lceil T^\top : \mathbb{U}[_] \rceil$ is a type $\mathbf{y}\mathbb{I} \vdash T$ type, which is a diagram

$$\begin{array}{ccc}
 (\mathbb{N} \triangleright T[\mathbf{s}]) & \xleftarrow{\sqcup\langle \mathbf{s} \rangle} & (\mathbb{I} \triangleright T[\mathbf{id}_{\mathbb{I}}]) & \xrightarrow{\sqcup\langle \mathbf{t} \rangle} & (\mathbb{N} \triangleright T[\mathbf{t}]) & (4.6) \\
 \begin{array}{c} \uparrow \quad \downarrow \quad \uparrow \\ \sqcup\langle \mathbf{s} \rangle \quad \sqcup\langle \mathbf{r} \rangle \quad \sqcup\langle \mathbf{t} \rangle \\ \downarrow \quad \uparrow \quad \downarrow \\ (\mathbb{I} \triangleright T[\mathbf{s} \circ \mathbf{r}]) \end{array} & & & & \begin{array}{c} \uparrow \quad \downarrow \quad \uparrow \\ \sqcup\langle \mathbf{s} \rangle \quad \sqcup\langle \mathbf{r} \rangle \quad \sqcup\langle \mathbf{t} \rangle \\ \downarrow \quad \uparrow \quad \downarrow \\ (\mathbb{I} \triangleright T[\mathbf{t} \circ \mathbf{r}]) \end{array}
 \end{array}$$

i.e. it consists of a reflexive graph at the source, a reflexive graph at the target, and a set of edges spanning across, whose source is a node at the source and whose target is a node at the target.

The source of $\mathbb{I} \triangleright \lceil T^\top : \mathbb{U}[_] \rceil$ is given by $\lceil T^\top \rceil \langle \mathbf{s} \rangle = \lceil T[\mathbf{ys}] \rceil$, which clearly extracts the reflexive graph at the source, and similarly for the target.

The reflexive edge on $\mathbb{N} \triangleright \lceil T^\top : \mathbb{U}[_] \rceil$ is given by $\lceil T^\top \rceil \langle \mathbf{r} \rangle = \lceil T[\mathbf{yr}] \rceil$, which creates a diagram of the latter shape from one of the former, by using $(\mathbb{N} \triangleright T[\mathbf{id}_{\mathbb{N}}])$ for both sets of nodes, and $(\mathbb{I} \triangleright T[\mathbf{r}])$ for every set of edges.

Example 4.3.6 (Cubical sets). We remark that in the category of affine cubical sets $\mathbf{Psh}(\mathbf{Cube}_{\square})$ (example 2.3.11), an edge $(i : \mathbb{I}) \triangleright \lceil T^\top : \mathbb{U}[_] \rceil$ consists of a cubical set at the source, a cubical set at the target, and an entire *cubical* set in the middle, whose n -cubes are actually $(n + 1)$ -cubes spanning across.

In the category of cartesian cubical sets $\mathbf{Psh}(\mathbf{Cube})$, the situation is a bit more funny: the cubical set at the center becomes ternary, because we cannot only take sources and targets of lines (which are actually squares spanning across), but also diagonals.

Example 4.3.7 (Topos of trees). Continuing examples 2.3.14, 4.1.3, 4.1.7 and 4.3.3, an i -cell $i \triangleright \lceil T^\top : \mathbb{U}[_] \rceil$ is a type $\mathbf{y}i \vdash T$ type, which is a presheaf over $\omega/\mathbf{y}i \cong \omega/i \cong i + 1$.

4.3.5 Other types

The unit type $\Gamma \vdash \mathbf{Unit}$ type is modelled by the terminal presheaf over \mathcal{W}/Γ . The empty type $\Gamma \vdash \mathbf{Empty}$ type is modelled by the empty presheaf. The coproduct $\Gamma \vdash A \uplus B$ type is modelled as the coproduct of presheaves A and B . The naturals and the booleans are modelled by constant presheaves of the sets of naturals and booleans.

Part II

Contributions

Chapter 5

Multimode Type Theory

Parts of the introduction of this chapter are taken from the abstract on Menkar [ND19b].

In modal type theory, all functions and all variables are annotated with a *modality* describing the behaviour of the dependency. Applications include: modal logic (eponymously) [PD01], variance of functors [Abe06; Abe08; LH11], intensionality vs. extensionality [Pfe01], irrelevance [Pfe01; Miq01; BB08; MS08; Ree03; AS12; AVW17; ND18a], shape-irrelevance [AVW17; ND18a], parametricity [NVD17a], axiomatic cohesion [LS16] and globality [Lic+18]. In order to annotate identity and composite functions, there are identity and composite modalities, turning the set of modalities into a monoid. The fact that some modalities are weaker than others, makes this an *ordered* monoid.

Sometimes, the set of available modalities μ for functions $(\mu \mid x : A) \rightarrow B$ depends on the types A and B . For example, in the type system RelDTT for Degrees of Relatedness [ND18a] (chapter 9), functions from \mathbb{N} to \mathbf{Bool} are either ad hoc or irrelevant, whereas functions from the universe to \mathbf{Bool} can also be parametric and functions from \mathbb{N} to the universe can also be shape-irrelevant. In System F, there is always at most one modality applicable, but it is not always the same: functions from a type to a type are always ad hoc, while functions from a kind to a type are always parametric. Recently, Licata and Shulman [LS16] have explained these phenomena by moving from an ordered monoid to a 2-category, whose objects are called **modes** and whose morphisms serve as modalities. If there happens to be only a single mode, then we are essentially back in the ordered monoid setting. In case there are or may be multiple modes, we speak of multimode type theory, which is thus a generalization of modal type theory. Here, one assigns a mode to every type, and the modality of a

function must match the domain and codomain modes. For System F, we could have 2 modes: **data** for types classifying data and **type** for kinds classifying types. The modes of RelDTT are called $-1, 0, 1, 2, \dots$ but could be read as **proof**, **data**, **type**, **kind**, etc.

Typically, a mode is interpreted as a CwF, and a modality as a CwF morphism. The modal function type $(\mu \mid x : A) \rightarrow B$ is then interpreted as an ordinary function type, with the functor μ applied to its domain.

In this chapter, we present 3 approaches to modal type theory, of which I contributed to the first and the third.

Section 5.1 starts from the semantics of dependent type theory given in section 3.2, i.e. the concept of a CwF, and asks the very natural question: what additional things can we do internally if we have a CwF morphism $F : \mathcal{C} \rightarrow \mathcal{D}$, a natural transformation $\nu : F \rightarrow G$ between such morphisms, or an adjunction $L \dashv R$ where at least R is a CwF morphism. Each of these questions leads to an extension of DTT which is syntactically not well-behaved (e.g. it is not clear how to write a type-checker) but which is sound and complete for the assumed semantic situation. This theory was developed as part of the semantics of ParamDTT and RelDTT [Nuy17; Nuy18a].

Section 5.2, presents Birkedal et al.’s independently developed notion of dependent right adjoints (DRAs) [Bir+20] (which I have no contribution in), which could be seen as CwF morphisms R that have a left adjoint L but which lack an action on contexts and terms. Instead, they are operated using transposition-like rules. These DRAs are syntactically still ill-behaved because one of its typing rules (DRA:ELIM) has a conclusion with a non-general context. DRAs are compared to the situation in section 5.1 where we have adjoint functors $L \dashv R$ where R is a CwF morphism.

Section 5.3 subsumes our paper on MTT [Gra+20b], which presents a general multimode type theory parametrized by an arbitrary external 2-category called the *mode theory*. This type system contains a modal type that is slightly less expressive than a DRA – we call it a weak DRA – and solves the syntactic problems by providing an inductive eliminator.

5.1 Internalizing Transformations of Semantics

This section is heavily based on prior work [Nuy17; Nuy18a, ch. 2].

In chapter 3, we defined dependent type theory (DTT) as a GAT and then took that definition as a starting point to investigate what models and morphisms of

models of DTT look like. This lead to the definition of categories with families (CwF, definitions 3.2.3 and 3.2.5) and natural models (definition 3.2.9).

With these notions at hand, we now ask the following question: if we have multiple CwFs, some strict or weak CwF morphisms between them (section 5.1.1), and perhaps natural transformations (section 5.1.2) or even adjunctions (section 5.1.3) between those, what bigger GAT does that model? More precisely, we will derive syntax that soundly and completely internalizes the properties of the aforementioned enlargements of the model.

5.1.1 Internalizing Functors and CwF Morphisms

Given a functor $F : \mathcal{C} \rightarrow \mathcal{D}$ between CwFs \mathcal{C} and \mathcal{D} , we start with a GAT consisting of twice DTT, once with all judgements postfixed with $@ \mathcal{C}$ and taking place in \mathcal{C} , once with $@ \mathcal{D}$. Figure 5.1 lists typing rules involving F that can be added to this GAT. If F is merely a functor, then we can apply it to objects (contexts) and morphisms (substitutions) and it respects identity and composition of substitutions. Moreover, this action preserves identity and composition of functors.

If F is a CwF morphism, then it follows immediately from the definition that F can be applied to types and terms in a manner stable under substitution (i.e. natural in the context). Again, this action preserves identity and composition of functors.

If F is a weak CwF morphism, then we know that it preserves the empty context and context extension up to isomorphism. We internalize this by postulating the same universal properties for the images of the empty context $F()$ and context extension $F(\Gamma.T)$ as we have for $()$ and $F\Gamma.FT$. The weakening and variable operators for $F(\Gamma.T)$ are not new operators, as they can be obtained by applying F to the weakening and variable operators for $\Gamma.T$. The pairing operator and $()_F$ are however new; their notations were already established in definition 3.2.5. The images of pairs and $()$ can be written in terms of these new operators; trying it the other way around, e.g. defining $()_F$ as $F()$, would be ill-dominated in general.

Proposition 5.1.1. We have $F() = ()_F : F\Gamma \rightarrow F()$ and $F(\sigma, t/x) = (F\sigma, Ft/Fx)_F : F\Delta \rightarrow F(\Gamma, x : T)$.

Proof. This follows from the η -rule for substitutions to $F()$ and $F(\Gamma, x : T)$. \square

If F is a strict CwF morphism, we simply postulate that F respects the empty context and context extension and the corresponding operations on the nose.

Prerequisites: DTT @ \mathcal{C} , DTT @ \mathcal{D} (fig. 3.2). Some equality rules require an additional functor $G : \mathcal{D} \rightarrow \mathcal{E}$ (fig. 5.1).

Functor:

$\frac{\text{FTR:CTX} \quad \Gamma \text{ ctx } @ \mathcal{C}}{F\Gamma \text{ ctx } @ \mathcal{D}}$	$\frac{\text{FTR:SUB} \quad \sigma : \Gamma \rightarrow \Delta @ \mathcal{C}}{F\sigma : F\Gamma \rightarrow F\Delta @ \mathcal{D}}$
<p>where $\text{Id } \Gamma = \Gamma$</p> <p>(FTR:ID:CTX)</p> <p>$(GF)\Gamma = G(F\Gamma)$ (FTR:COMP:CTX)</p>	<p>where $\text{Id } \sigma = \sigma$ (FTR:ID:SUB)</p> <p>$(GF)\sigma = G(F\sigma)$ (FTR:COMP:SUB)</p> <p>$F \text{ id }_{F\Gamma} = \text{id}_{F\Gamma}$ (FTR:SUB:ID)</p> <p>$F(\tau \circ \sigma) = F\tau \circ F\sigma$ (FTR:SUB:COMP)</p>

CwF morphism:

Types and terms:

$\frac{\text{FTR:TY} \quad \Gamma \vdash T \text{ type } @ \mathcal{C}}{F\Gamma \vdash FT \text{ type } @ \mathcal{D}}$	$\frac{\text{FTR:TM} \quad \Gamma \vdash t : T @ \mathcal{C}}{F\Gamma \vdash Ft : FT @ \mathcal{D}}$
<p>where $F(T[\sigma]) = (FT)[F\sigma]$</p> <p>$\text{Id } T = T$ (FTR:ID:TY)</p> <p>$(GF)T = G(FT)$ (FTR:COMP:TY)</p>	<p>where $F(t[\sigma]) = (Ft)[F\sigma]$</p> <p>$\text{Id } t = t$ (FTR:ID:TM)</p> <p>$(GF)t = G(Ft)$ (FTR:COMP:TM)</p>

Weak CwF morphism:

Empty context:

$\frac{\text{IMG:EMPTY-CTX:INTRO} \quad \Gamma \text{ ctx } @ \mathcal{D}}{()_F : \Gamma \rightarrow F()}$	<p>where $\sigma = ()_F : \Gamma \rightarrow F()$ (IMG:EMPTY-CTX:ETA)</p>
--	--

Context extension:

$\frac{\text{IMG:CTX-EXT:INTRO} \quad \Gamma \vdash T \text{ type } @ \mathcal{C} \quad \sigma : \Delta \rightarrow F\Gamma @ \mathcal{D} \quad \Delta \vdash t : (FT)[\sigma] @ \mathcal{D}}{(\sigma, t/Fx)_F : \Delta \rightarrow F(\Gamma, x : T) @ \mathcal{D}}$	<p>where $(\sigma, t/Fx)_F \circ \rho = (\sigma \circ \rho, t[\rho]/Fx)_F$</p> <p>$F\pi_x \circ (\sigma, t/Fx)_F = \sigma$ (IMG:CTX-EXT:WKN:BETA)</p> <p>$(Fx)[(\sigma, t/Fx)_F] = t$ (IMG:CTX-EXT:VAR:BETA)</p> <p>$\tau = (F\pi_x \circ \tau, (Fx)[\tau]/Fx)_F : \Delta \rightarrow F(\Gamma, x : T)$ (IMG:CTX-EXT:ETA)</p>
--	---

Strict CwF morphism:

$\frac{\text{FTR:EMPTY-CTX} \quad F() = () \text{ ctx } @ \mathcal{D}}{\Gamma \vdash T \text{ type } @ \mathcal{C}}$	<p>(FTR:EMPTY-CTX)</p>
$\frac{F(\Gamma, x : T) = (F\Gamma, Fx : FT) \text{ ctx } @ \mathcal{D} \quad F\pi_x = \pi_{F_x} : (F\Gamma, Fx : FT) \rightarrow F\Gamma @ \mathcal{D} \quad F\Gamma, Fx : FT \vdash F(x) = (FT)[\pi_{F_x}] @ \mathcal{D}}{F(\sigma, t) = (F\sigma, FT) : F\Gamma \rightarrow (F\Delta, Fx : FT) @ \mathcal{D}}$	<p>(FTR:CTX-EXT)</p> <p>(FTR:CTX-EXT:WKN)</p> <p>(FTR:CTX-EXT:VAR)</p>
$\frac{\text{FTR:EMPTY-CTX:INTRO} \quad \Gamma \text{ ctx } @ \mathcal{C}}{F() = () : F\Gamma \rightarrow () @ \mathcal{D}}$	$\frac{\text{FTR:CTX-EXT:INTRO} \quad \sigma : \Gamma \rightarrow \Delta @ \mathcal{C} \quad \Gamma \vdash t : T[\sigma] @ \mathcal{C}}{F(\sigma, t) = (F\sigma, FT) : F\Gamma \rightarrow (F\Delta, Fx : FT) @ \mathcal{D}}$

Figure 5.1: Typing rules internalizing a CwF morphism $F : \mathcal{C} \rightarrow \mathcal{D}$ [Nuy17].

As already remarked in definition 3.2.5, in variable notation we need to decide a name for the variable x after applying F . The fact that strict CwF morphisms preserve ξ on the nose ($F\xi = \xi$) suggests the name Fx . This convention is fine except when expressing $F\xi = \xi$ in variable notation, as we then get $Fx = Fx$.

Of course strict CwF morphisms are also weak, but then we can simply derive the weak CwF morphism rules from the strict ones, e.g. $(\sigma, t/Fx)_F := (\sigma, t/Fx)$.

Proposition 5.1.2 (Soundness and completeness). A model of the GAT in fig. 5.1 (including prerequisites, excluding rules requiring additional prerequisites) consists precisely of two CwFs \mathcal{C} and \mathcal{D} , together with:

- A functor $F : \mathcal{C} \rightarrow \mathcal{D}$ (if we only add the functor rules, leaving out the rules for identity and composition of functors),
- A weak/strict CwF morphism $F : \mathcal{C} \rightarrow \mathcal{D}$ (if we also add the rules for a weak/strict CwF morphism).

By ‘consists precisely’, we mean that the category of models of the described GAT is isomorphic to the category of triples $(\mathcal{C}, \mathcal{D}, F)$ with as morphisms triples (C, D) where C and D are strict CwF morphisms causing a square to commute strictly. \square

Remark 5.1.3. We call a CwF morphism F **level-preserving** if it sends $T \in \text{Ty}_\ell(\Gamma) \subseteq \text{Ty}(\Gamma)$ to $FT \in \text{Ty}_\ell(F\Gamma) \subseteq \text{Ty}(F\Gamma)$. Of course level-preserving CwF morphisms model the following rule:

$$\frac{\Gamma \vdash T \text{ type}_\ell @ \mathcal{C}}{F\Gamma \vdash FT \text{ type}_\ell @ \mathcal{D}}. \quad (5.1)$$

5.1.2 Internalizing Natural Transformations

In this section, we consider functors $F, G : \mathcal{C} \rightarrow \mathcal{D}$ and a natural transformation $\alpha : F \rightarrow G$ between the underlying functors. Figure 5.2 lists rules that can be added to a GAT already consisting of twice DTT extended with twice the rules from fig. 5.1; once for F and once for G .

It is clear that for any context Γ , we get a substitution $\alpha_\Gamma : F\Gamma \rightarrow G\Gamma$ between its respective images. Just like we did for π and ξ , we will omit the index Γ on α .

If F and G are weak CwF morphisms, then without imposing additional requirements on α , we can also apply it to types $\Gamma \vdash T \text{ type} @ \mathcal{C}$, yielding a function $F\Gamma \vdash \alpha : FT \rightarrow (GT)[\alpha] @ \mathcal{D}$. We prefer our typing rules to end in

Prerequisites: DTT @ \mathcal{C} , DTT @ \mathcal{D} (fig. 3.2), weak CwF morphisms $F, G : \mathcal{C} \rightarrow \mathcal{D}$ (fig. 5.1). Some equality rules require an additional weak CwF morphism $H : \mathcal{D} \rightarrow \mathcal{E}$ (fig. 5.1) or an additional natural transformation $\beta : G \rightarrow K : \mathcal{C} \rightarrow \mathcal{D}$ (fig. 5.2).

For every context a substitution:

NATTRANS:CTX
 $\Gamma \text{ ctx @ } \mathcal{C}$

$\alpha_\Gamma : F\Gamma \rightarrow G\Gamma @ \mathcal{D}$
 where $\alpha_\Delta \circ F\sigma = G\sigma \circ \alpha_\Gamma$ (NATTRANS:SUB)
 $(\text{id}_F)_\Gamma = \text{id}_{F\Gamma} : F\Gamma \rightarrow F\Gamma$ (NATTRANS:ID:CTX)
 $(\beta \circ \alpha)_\Gamma = \beta_\Gamma \circ \alpha_\Gamma : F\Gamma \rightarrow K\Gamma$ (NATTRANS:COMP:CTX)
 $(H\alpha)_\Gamma = H(\alpha_\Gamma) : HF\Gamma \rightarrow HG\Gamma$ (NATTRANS:WHISKER:CTX)

For every type a function:

NATTRANS:TY

$\Gamma \vdash T \text{ type @ } \mathcal{C} \quad \sigma : \Delta \rightarrow F\Gamma @ \mathcal{D} \quad \Delta \vdash t : (FT)[\sigma] @ \mathcal{D}$

$\Delta \vdash \alpha_\sigma(t) : (GT)[\alpha_\sigma] @ \mathcal{D}$
 where $\alpha_\sigma(t)[\tau] = \alpha_{\sigma \circ \tau}(t[\tau])$ (NATTRANS:TY:SUB)
 $\alpha_{F\rho \circ \sigma}(t) = \alpha_\sigma(t)$ (NATTRANS:TY:COEND)
 $\alpha_{\text{id}}(Ft) = (Gt)[\alpha]$ (NATTRANS:TM)
 $\text{id}_\sigma(t) = t$ (NATTRANS:ID:TY)
 $(\beta \circ \alpha)_\sigma(t) = \beta_{\alpha \circ \sigma}(\alpha_\sigma(t))$ (NATTRANS:COMP:TY)
 $H(\alpha_\sigma(t)) = (H\alpha)_{H\sigma}(Ht)$ (NATTRANS:WHISKER:TY)

Figure 5.2: Typing rules internalizing a natural transformation $\alpha : F \rightarrow G : \mathcal{C} \rightarrow \mathcal{D}$ [Nuy17].

conclusions living in a general context (so that we can at least dream of building a type-checker) so we actively close this function under substitution. This yields the rule for α_σ .

Lemma 5.1.4. From the rules in fig. 5.2, we can derive

$$\frac{\Gamma \vdash T \text{ type @ } \mathcal{C}}{\alpha = (\alpha \circ F\pi_x, \alpha_{\text{id}}(Fx)/Gx)_G : F(\Gamma, x : T) \rightarrow G(\Gamma, x : T)}.$$

Proof. We have

$$\alpha = \text{id}_{G\Gamma} \circ \alpha = (G\pi_x, Gx/Gx)_G \circ \alpha = (G\pi_x \circ \alpha, (Gx)[\alpha]/Gx)_G$$

or in variable notation

$$\alpha_\sigma(t) := (Gx)[\alpha][(\sigma, t/Fx)_F]. \quad (5.3)$$

We now check all the required properties:

- (NATTRANS:TY:SUB) We have

$$\alpha_\sigma(t)[\tau] = (G\xi)[\alpha][(\sigma, t)_F][\tau] = (G\xi)[\alpha][(\sigma \circ \tau, t[\tau])_F] = \alpha_{\sigma \circ \tau}(t[\tau]).$$

- (NATTRANS:TY:COEND) We have

$$\begin{aligned} \alpha_{F\rho \circ \sigma}(t) &= (G\xi)[\alpha][(\rho \circ \sigma, t)_F] \\ &= (G\xi)[\alpha][F(\rho+)][(\sigma, t)_F] \\ &= (G\xi)[G(\rho+)][\alpha][(\sigma, t)_F] = (G\xi)[\alpha][(\sigma, t)_F]. \end{aligned}$$

- (NATTRANS:TM) We have

$$\begin{aligned} \alpha_{\text{id}}(Ft) &= (G\xi)[\alpha][(\text{id}, Ft)_F] = (G\xi)[\alpha][F(\text{id}, t)] \\ &= (G\xi)[G(\text{id}, t)][\alpha] = (Gt)[\alpha]. \end{aligned}$$

- (NATTRANS:ID:TY) We have

$$\text{id}_\sigma(t) = (F\xi)[\text{id}][(\sigma, t)_F] = t.$$

- (NATTRANS:COMP:TY) Let $\beta : G \rightarrow K : \mathcal{C} \rightarrow \mathcal{D}$. We have

$$\begin{aligned} \beta_{\alpha \circ \sigma}(\alpha_\sigma(t)) &= (K\xi)[\beta][(\alpha \circ \sigma, \alpha_\sigma(t))_G] \\ &= (K\xi)[\beta][(\alpha \circ \sigma, (G\xi)[\alpha][(\sigma, t)_F])_G] \\ &= (K\xi)[\beta][\alpha \circ (\sigma, t)_F] = (K\xi)[\beta \circ \alpha][(\sigma, t)_F] = (\beta \circ \alpha)_\sigma(t). \end{aligned}$$

In the first two steps, we simply expand definitions. In the third step, we observe that

$$(\alpha \circ \sigma, (G\xi)[\alpha][(\sigma, t)_F])_G = \alpha \circ (\sigma, t)_F : \Delta \rightarrow G(\Gamma.T)$$

because they yield equal results when postcomposing with $G\pi$ and when applying to $G\xi$.

- (NATTRANS:WHISKER:TY) Let $H : \mathcal{D} \rightarrow \mathcal{E}$, so that $H\alpha : HF \rightarrow HG : \mathcal{C} \rightarrow \mathcal{E}$. We have

$$\begin{aligned} H(\alpha_\sigma(t)) &= H(G\xi)[\alpha][(\sigma, t)_F] \\ &= (HG\xi)[H\alpha][(\sigma, t)_{HF}] = (H\alpha)_{H\sigma}(Ht). \quad \square \end{aligned}$$

Corollary 5.1.7 (Soundness and completeness). A model of the GAT in fig. 5.2 (including prerequisites; excluding rules requiring additional prerequisites) consists exactly of two CwFs, two weak CwF morphisms F and G between them, and a natural transformation $\alpha : \mathcal{C} \rightarrow \mathcal{D}$.

This statement is to be understood in a similar manner as proposition 5.1.2.

Proof. By proposition 5.1.5, the second rule adds nothing to the model. The first rule is quite obviously sound and complete for a natural transformation. \square

Corollary 5.1.8. If a functor $F : \mathcal{C} \rightarrow \mathcal{D}$ is a weak CwF morphism in two ways (called F_1 and F_2), then the actions on types and terms are essentially the same in the following sense: For any $T \in \text{Ty}_{\mathcal{C}}(\Gamma)$, we have $\text{Tm}_{\mathcal{D}}(FT, F_1T) \cong \text{Tm}_{\mathcal{D}}(FT, F_2T)$ in a manner compatible with the actions of F_1 and F_2 on terms.

Proof. We have a natural isomorphism $\alpha := \text{id} : F_1 \cong F_2$. This yields functions $\alpha_{\text{id}} : F_1T \rightarrow F_2T$ and $(\alpha^{-1})_{\text{id}} : F_2T \rightarrow F_1T$ which are mutually inverse by functoriality of the action of natural transformations on terms. Dependent naturality asserts compatibility with the actions on terms. \square

5.1.3 Internalizing Adjunctions

In this section, we consider functors $L : \mathcal{C} \rightarrow \mathcal{D}$ and $R : \mathcal{D} \rightarrow \mathcal{C}$, where L may be and R is a morphism of CwFs, such that $\alpha : L \dashv R$. If L is a CwF morphism, then fig. 5.2 gives us functions $\eta : T \rightarrow (RLT)[\eta]$ and $\varepsilon : LRT \rightarrow T[\varepsilon]$. Moreover, $\eta(t) = (RLt)[\eta]$ and $\varepsilon(LRt) = t[\varepsilon]$ by dependent naturality.

We do not need additional rules to transpose substitutions, as we can define

$$\mathbf{A}(\sigma) := R\sigma \circ \eta, \quad \mathbf{A}^{-1}(\tau) := \varepsilon \circ L\tau. \quad (5.4)$$

Interestingly, without further requirements on \mathbf{A} , we can also transpose terms!

Proposition 5.1.9. The rules TRANSPOSE and UNTRANSPOSE in fig. 5.3 and their associated equation rules are uniquely *definable/derivable* in the GAT consisting of the prerequisites and ADJ:RLR and ADJ:LRL.

Notation 5.1.10. Note that in the rule of $\mathbf{A}_{\sigma}(t)$, without loss of generality, we may assume that $\sigma = \text{id} : L\Delta \rightarrow L\Delta = \Gamma$. Hence we will suppress the annotation σ , unambiguously.

Proof. Uniqueness is obvious, as $\mathbf{A}_{\sigma}(t)$ is simply *defined* by $\mathbf{A}_{\sigma}(t) = (Rt)[\eta]$ and \mathbf{A}^{-1} is its inverse.

Prerequisites: DTT @ \mathcal{C} , DTT @ \mathcal{D} (fig. 3.2), functor $L : \mathcal{D} \rightarrow \mathcal{C}$, weak CwF morphism $R : \mathcal{C} \rightarrow \mathcal{D}$, identity and composite functors (fig. 5.1), natural transformations $\eta : \text{Id} \rightarrow RL$ and $\varepsilon : LR \rightarrow \text{Id}$ (fig. 5.2).

Adjunction laws:

$$R\varepsilon \circ \eta R = \text{id}_R \quad (\text{ADJ:RLR}),$$

$$\varepsilon L \circ L\eta = \text{id}_L \quad (\text{ADJ:LRL}).$$

For convenience, we write

$$\mathbf{A}(\sigma) := R\sigma \circ \eta, \quad \mathbf{A}^{-1}(\tau) := \varepsilon \circ L\tau.$$

Dependent transposition:

TRANSCOPE

$$\sigma : L\Delta \rightarrow \Gamma @ \mathcal{C} \quad \Gamma \vdash T \text{ type } @ \mathcal{C}$$

$$L\Delta \vdash t : T[\sigma] @ \mathcal{C}$$

$$\Delta \vdash \mathbf{A}_\sigma(t) : (RT)[\mathbf{A}(\sigma)] @ \mathcal{D}$$

$$\text{where } \mathbf{A}_\sigma(\mathbf{A}_\sigma^{-1}(s)) = s \quad (\text{TRANSCOPE:CANCEL})$$

$$\mathbf{A}_\sigma(t)[\tau] = \mathbf{A}(t[L\tau]) \quad (\text{TRANSCOPE:SUB})$$

$$\mathbf{A}_{\rho \circ \sigma}(t) = \mathbf{A}_\sigma(t) \quad (\text{TRANSCOPE:COEND})$$

$$\mathbf{A}_\sigma(t) = (Rt)[\eta] \quad (\text{TRANSCOPE:DEF})$$

UNTRANSCOPE

$$\sigma : L\Delta \rightarrow \Gamma @ \mathcal{C} \quad \Gamma \vdash T \text{ type } @ \mathcal{C}$$

$$\Delta \vdash s : (RT)[\mathbf{A}(\sigma)] @ \mathcal{D}$$

$$L\Delta \vdash \mathbf{A}_\sigma^{-1}(s) : T[\sigma] @ \mathcal{C}$$

$$\text{where } \mathbf{A}_\sigma^{-1}(\mathbf{A}_\sigma(t)) = t \quad (\text{UNTRANSCOPE:CANCEL})$$

$$\mathbf{A}_\sigma^{-1}(s)[L\tau] = \mathbf{A}(s[\tau]) \quad (\text{UNTRANSCOPE:SUB})$$

$$\mathbf{A}_{\rho \circ \sigma}^{-1}(s) = \mathbf{A}_\sigma^{-1}(s) \quad (\text{UNTRANSCOPE:COEND})$$

$$\mathbf{A}_\sigma^{-1}(s) = \varepsilon_{L\mathbf{A}(\sigma)}(Ls) \quad \text{if } L \text{ is a CwF morphism}$$

$$(\text{UNTRANSCOPE:DEF})$$

Figure 5.3: Typing rules internalizing an adjunction $\mathbf{A} : L \dashv R$ between CwFs where $R : \mathcal{C} \rightarrow \mathcal{D}$ is a weak CwF morphism [Nuy17].

To see existence, we need to prove that \mathbf{A}_σ is invertible and satisfies all the equality rules. To this end, we use lemma 3.2.4 to find a correspondence between terms:

$$\begin{array}{ccc}
 L\Delta & \xrightarrow{(\sigma, t)} & \Gamma.T \\
 \searrow \sigma & & \swarrow \pi \\
 & \Gamma &
 \end{array}
 \qquad
 \begin{array}{ccc}
 & R\Gamma.RT & \\
 (\mathbf{A}(\sigma), \mathbf{A}(t)) \nearrow & \downarrow \pi & \nwarrow (R\pi, R\xi) \\
 \Delta & \xrightarrow{\mathbf{A}(\sigma, t)} & R(\Gamma.T) \\
 \searrow \mathbf{A}(\sigma) & & \swarrow R\pi \\
 & R\Gamma &
 \end{array}$$

From this, we see that even if L is not a CwF morphism, we can still define $\mathbf{A}_\sigma^{-1}(s)$ as

$$\begin{aligned}
 \mathbf{A}_\sigma^{-1}(s) &:= \xi[\mathbf{A}^{-1}((\pi, \xi)_R \circ (\mathbf{A}(\sigma), s))] \\
 &= \xi[\mathbf{A}^{-1}((\mathbf{A}(\sigma), s)_R)].
 \end{aligned}$$

If L happens to be a CwF morphism, then we can further reduce this to:

$$\begin{aligned}
 &= \xi[\varepsilon \circ L((\mathbf{A}(\sigma), s)_R)] \\
 &= \varepsilon_{\text{id}}(LR\xi)[(L\mathbf{A}(\sigma), Ls)_{LR}] = \varepsilon_{(L\mathbf{A}(\sigma), Ls)_{LR}}(Ls) = \varepsilon_{L\mathbf{A}(\sigma)}(Ls),
 \end{aligned}$$

where in the last step we use NATTRANS:TY:COEND, using that $LR\pi \circ (L\mathbf{A}(\sigma), Ls)_{LR} = L\mathbf{A}(\sigma)$.

Conversely, we find

$$\begin{aligned}
 \mathbf{A}_\sigma(t) &:= \xi[(R\pi, R\xi) \circ \mathbf{A}(\sigma, t)] \\
 &= \xi[(R\pi, R\xi) \circ R(\sigma, t) \circ \eta] \\
 &= \xi[(R\sigma, Rt) \circ \eta] = (Rt)[\eta]
 \end{aligned}$$

so that we have indeed inverted the right operation.

For TRANSPOSE:SUB (which implies UNTRANSPOSE:SUB), we have

$$\mathbf{A}(t)[\tau] = (Rt)[\eta][\tau] = (Rt)[RL\tau][\eta] = (R(t[L\tau]))[\eta] = \mathbf{A}(t[L\tau]).$$

The rules TRANSPOSE:COEND and UNTRANSPOSE:COEND are blatantly obvious as σ does not even occur in the definition. \square

Corollary 5.1.11 (Soundness and completeness). A model of the GAT in fig. 5.3 (including prerequisites) consists exactly of two CwFs \mathcal{C} and \mathcal{D} , a functor $L : \mathcal{D} \rightarrow \mathcal{C}$, and a weak CwF morphism $R : \mathcal{C} \rightarrow \mathcal{D}$ such that $L \dashv R$.

This statement is to be understood in a similar manner as proposition 5.1.2.

Proof. By proposition 5.1.9, the rules TRANSPOSE and UNTRANSPOSE and their associated equation rules add nothing to the model. The existence of η and ε together with the adjunction laws are clearly sound and complete for the requirement that $L \dashv R$. \square

Corollary 5.1.12. [From Nuy17] We have naturality rules as for ordinary adjunctions:

$$\begin{array}{l|l} \mathbf{A}(\tau \circ \sigma \circ L\rho) = R\tau \circ \mathbf{A}(\sigma) \circ \rho & \mathbf{A}^{-1}(R\tau \circ \sigma \circ \rho) = \tau \circ \mathbf{A}^{-1}(\sigma) \circ L\rho \\ \mathbf{A}(t[\sigma][L\rho]) = (Rt)[\mathbf{A}(\sigma)][\rho] & \mathbf{A}^{-1}((Rt)[\sigma][\rho]) = t[\mathbf{A}^{-1}(\sigma)][L\rho] \\ \mathbf{A}(\alpha(s[L\rho])) = (R\alpha)(\mathbf{A}(s))[\rho] & \mathbf{A}^{-1}((R\alpha)s[\rho]) = \alpha(\mathbf{A}^{-1}(s))[L\rho] \\ \mathbf{A}(\beta(\alpha(Lr))) = (R\beta)(\mathbf{A}(\alpha)(r)) & \mathbf{A}^{-1}((R\beta)(\alpha(r))) = \beta(\mathbf{A}^{-1}(\alpha)(Lr)) \end{array}$$

where ρ, σ, τ denote substitutions, s, t denote terms and α, β denote natural transformations.

Proof. Each equation on the right is equivalent to its counterpart on the left. The first equation on the left is old news. The other equations follow from $\mathbf{A}(t) = (Rt)[\eta]$. \square

5.2 Dependent Right Adjoints (DRAs)

Independently of my own development of the material in section 5.1 [Nuy17, ch. 2], Birkedal et al. [Bir+20] have developed the notion of a dependent right adjoint (DRA). In section 5.2.1, we describe DRAs syntactically and semantically in the same framework of type theory as a GAT that we have been using in the rest of this dissertation. While Birkedal et al. only consider DRAs of endofunctors, we will immediately make the straightforward generalization to DRAs of arbitrary functors, as we also did in the technical report on MTT [Gra+20a]. In section 5.2.2, we adapt some results by Birkedal et al. which relate the notion of the DRA to the content of section 5.1.3.

5.2.1 Syntax and Semantics

Definition 5.2.1. Given CwFs \mathcal{C} and \mathcal{D} and a mere functor $L : \mathcal{D} \rightarrow \mathcal{C}$, a **dependent right adjoint (DRA)** R consists of:

- A natural transformation $\langle R \mid \sqcup \rangle : \text{Ty}_{\mathcal{C}} \circ L \rightarrow \text{Ty}_{\mathcal{D}} : \mathcal{D}^{\text{op}} \rightarrow \text{Set}$. Concretely, this is an operation that sends $T \in \text{Ty}_{\mathcal{C}}(L\Gamma)$ to $\langle R \mid T \rangle \in \text{Ty}_{\mathcal{D}}(\Gamma)$ so that $\langle R \mid T \rangle[\sigma] = \langle R \mid T[L\sigma] \rangle$.
- For every $\Gamma \in \mathcal{D}$ and $T \in \text{Ty}_{\mathcal{C}}(L\Gamma)$, an isomorphism $\mathbf{A} : \text{Tm}(L\Gamma, T) \cong \text{Tm}(\Gamma, \langle R \mid T \rangle)$ which is natural in Γ so that $\mathbf{A}(t)[\sigma] = \mathbf{A}(t[L\sigma])$.

Prerequisites: DTT @ \mathcal{C} , DTT @ \mathcal{D} (fig. 3.2), functor $L : \mathcal{D} \rightarrow \mathcal{C}$ (fig. 5.1).

Formation rule:

$$\begin{array}{c} \text{DRA} \\ \Gamma \text{ ctx @ } \mathcal{D} \\ L\Gamma \vdash T \text{ type @ } \mathcal{C} \\ \hline \Gamma \vdash \langle R \mid T \rangle \text{ type @ } \mathcal{D} \\ \text{where } \langle R \mid T \rangle[\sigma] = \langle R \mid T[L\sigma] \rangle \quad (\text{DRA:SUB}) \end{array}$$

Introduction and elimination rules:

$$\begin{array}{c} \text{DRA:INTRO} \\ \Gamma \text{ ctx @ } \mathcal{D} \quad L\Gamma \vdash t : T @ \mathcal{C} \\ \hline \Gamma \vdash \mathbf{A}(t) : \langle R \mid T \rangle @ \mathcal{D} \\ \text{where } \mathbf{A}(t)[\sigma] = \mathbf{A}(t[L\sigma]) \quad (\text{DRA:INTRO:SUB}) \\ \quad \quad \quad s = \mathbf{A}(\mathbf{A}^{-1}(s)) \quad (\text{DRA:ETA}) \end{array}$$

$$\begin{array}{c} \text{DRA:ELIM} \\ \Gamma \vdash s : \langle R \mid T \rangle @ \mathcal{D} \\ \hline L\Gamma \vdash \mathbf{A}^{-1}(s) : T @ \mathcal{C} \\ \text{where } \mathbf{A}^{-1}(s)[L\sigma] = \mathbf{A}^{-1}(s[\sigma]) \quad (\text{DRA:ELIM:SUB}) \\ \quad \quad \quad \mathbf{A}^{-1}(\mathbf{A}(t)) = t \quad (\text{DRA:BETA}) \end{array}$$

Figure 5.4: Typing rules for a DRA R of a functor $L : \mathcal{D} \rightarrow \mathcal{C}$ between CwFs [Bir+20].

Figure 5.4 lists typing rules for extending DTT with a DRA. We have adapted a few notations in order to fit with section 5.1:

Our notation	$L\Gamma$	$\langle R \mid T \rangle$	$\mathbf{A}(t)$	$\mathbf{A}^{-1}(s)$
[Bir+20]	Γ, \blacksquare	$\square T$	shut t	open t

There is also a difference in approach. Our formulation of DTT (fig. 3.2) uses explicit substitutions (there is a substitution judgement and a syntactic operation $t[\sigma]$ for applying substitutions). This gives us the luxury of providing a variable rule only for accessing the last variable; other variables should be accessed by explicitly applying a weakening substitution. Birkedal et al. do not use explicit substitutions and instead provide a variable rule for accessing any variable that is not behind a lock, i.e. not under L . Written in variable notation with silent weakening, we are allowed to do

$$L(\Gamma, x : A), y : B, z : C \vdash y : B @ \mathcal{C}$$

but not

$$L(\Gamma, x : A), y : B, z : C \vdash x : A @ \mathcal{C} \quad (\text{wrong!}).$$

Because we have generalized DRAs beyond endofunctors, the latter judgement is more obviously nonsense in our setting than it was in the original setting: A is a type in the CwF \mathcal{D} , so accessing its variables when it is under a lock yields a judgement in the wrong CwF.

A consequence of the absence of explicit weakening is also that the typing rule for $\mathbf{A}^{-1}(s)$, which produces a judgement in a non-general context, originally had a weakening implied:

$$\frac{\Gamma \vdash s : \langle R \mid T \rangle @ \mathcal{D} \quad L\Gamma, \Theta \text{ ctx } @ \mathcal{C}}{L\Gamma, \Theta \vdash \mathbf{A}^{-1}(s) : T @ \mathcal{C}}$$

In other words, when type-checking $\mathbf{A}^{-1}(s)$, we would discard all variable that are not under L , and then remove the application of L , before type-checking s .

Our development in sections 5.1 and 5.2 is aimed at getting *some* internalization of interesting situations in the model. Only in section 5.3 will we make a real attempt at obtaining a type-checker friendly type system.

Proposition 5.2.2 (Soundness and completeness). A model of the GAT in fig. 5.4 (including prerequisites) consists exactly of two CwFs \mathcal{C} and \mathcal{D} and a functor $L : \mathcal{D} \rightarrow \mathcal{C}$ with a DRA R . \square

5.2.2 Relation to Right Adjoints

Although Birkedal et al. [Bir+20] have not considered typing rules as in fig. 5.3, they did relate the semantics of DRAs to the existence of a right adjoint functor $L \dashv R$ [Bir+20, lemma 17 and cor. 23]. We generalize their results beyond endofunctors and internalize their lemma 17:

Lemma 5.2.3. In DTT equipped with an adjunction $\mathbf{A} : L \dashv R$ (fig. 5.3) where R is a weak CwF morphism, the rules of a DRA (fig. 5.4) are derivable by defining $\langle R \mid T \rangle := (RT)[\eta]$.

Proof. Then \mathbf{A} and \mathbf{A}^{-1} for the DRA are instances of the operators of the same name for the adjunction. □

Theorem 5.2.4. Given CwFs \mathcal{C} and \mathcal{D} where \mathcal{D} is democratic, and a functor $L : \mathcal{D} \rightarrow \mathcal{C}$ with a DRA R , it follows that the functor L has a right adjoint R which is a weak CwF morphism.

Proof. By straightforward adaptation of [Bir+20, cor. 23] to non-endo L . □

5.3 Multimode Type Theory (MTT)

Preamble This section is based partly on the submitted version and partly on the most up to date version of the intended camera-ready of the following paper:

[Gra+20b] D. Gratzer, G. A. Kavvos, A. Nuyts, and L. Birkedal. “Multimodal Dependent Type Theory”. In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by H. Hermanns, L. Zhang, N. Kobayashi, and D. Miller. ACM, 2020, pp. 492–506. DOI: 10.1145/3373718.3394736. URL: <https://doi.org/10.1145/3373718.3394736>

The most important differences with the original paper are:

- The content of section 5.3.2.a on ‘the type theory at each mode’ was mostly suppressed in favour of a referral to earlier chapters.
- We introduced the notation using *ticks* (section 5.3.2.b) in order to better support the content of chapter 7 on the transpension type.
- Hence, the figures containing typing rules were entirely redone and integrated with previous chapters, partly using material from the unpublished paper that chapter 7 is based on. This material is then suppressed there (section 7.3).

Personal contributions

I contributed

- the general idea of the type system, which was already apparent in Degrees of Relatedness (RelDTT) [ND18a] and of course based on earlier work (section 5.3.7),
- the idea that a left division *operation* could be replaced with a Fitch-style context *constructor* \mathbb{A} ,
- a semantics in which modalities are interpreted as pairs $L \dashv R$ with R a CwF morphism,
- an application section on Degrees of Relatedness in the technical report [Gra+20a], subsumed in chapter 9 of this thesis,
- ideas on internal adjoints which were already ubiquitous in RelDTT,
- the tick notation, which is not actually part of the LICS paper.

Together, we crafted an explicit substitution calculus that handles mode theories that are 2-categories rather than 2-posets (poset-enriched categories).

Almost everything else is by one or more of the co-authors, including the precise typing rules, the fully general semantics, the canonicity proof, almost all applications in the technical report [Gra+20a] and almost all writing.

Abstract We introduce MTT, a dependent type theory which supports multiple modalities. MTT is parametrized by a mode theory which specifies a collection of modes, modalities, and transformations between them.

We show that different choices of mode theory allow us to use the same type theory to compute and reason in many modal situations, including guarded recursion, axiomatic cohesion, and parametric quantification. We reproduce examples from prior work in guarded recursion and axiomatic cohesion – demonstrating that MTT constitutes a simple and usable syntax whose instantiations intuitively correspond to previous handcrafted modal type theories. In some cases, instantiating MTT to a particular situation unearths a previously unknown type theory that improves upon prior systems.

Finally, we investigate the metatheory of MTT. We prove the consistency of MTT and establish canonicity through an extension of recent type-theoretic gluing techniques. These results hold irrespective of the choice of mode theory, and thus apply to a wide variety of modal situations.

Acknowledgements We are grateful for productive conversations with Carlo Angiuli, Dominique Devriese, Adrien Guatto, Magnus Baunsgaard Kristensen,

Daniel Licata, Rasmus Ejlers Møgelberg, Matthieu Sozeau, Jonathan Sterling, and Andrea Vezzosi.

Alex Kavvos was supported in part by a research grant (12386, Guarded Homotopy Type Theory) from the VILLUM Foundation. Andreas Nuyts holds a PhD Fellowship from the Research Foundation - Flanders (FWO). This work was supported in part by a Villum Investigator grant (no. 25804), Center for Basic Research in Program Verification (CPV), from the VILLUM Foundation.

5.3.1 Introduction

In order to increase the expressivity of Martin-Löf Type Theory (MLTT) we often wish to extend it with new connectives, and in particular with unary type operators that we call *modalities* or *modal operators*. Some of these modal operators arise as shorthands, while others are introduced as a device for expressing structure that appears in particular models. Whereas the former class of modalities are internally definable [RSS20], the latter often require extensive modifications to the basic structure of type-theoretic judgments. In some cases we are even able to prove that these changes are necessary, by showing that the modality in question cannot be expressed internally: see e.g. the ‘no-go’ theorems by Shulman [Shu18, §4.1] and Licata et al. [Lic+18]. This paper is concerned with the development of a systematic approach to the formulation of type theories with multiple modalities.

The addition of a modality to a dependent type theory is a non-trivial exercise. Modal operators often interact with the context of a type or term in a complicated way, and naïve approaches lead to undesirable interplay with other type formers and substitution. However, the consequent gain in expressivity is substantial, and so it is well worth the effort. For example, modalities have been used to express guarded recursive definitions [Bir+12; Biz+16; BGM17; Gua18], parametric quantification [NVD17a; ND18a], proof irrelevance [Pfe01; AS12; ND18a], and to define operations which cannot be localized to an arbitrary context [Lic+18]. There has also been concerted effort towards the development of a dependent type theory corresponding to Lawvere’s *axiomatic cohesion* [Law07], which has many interesting applications [Sch13; SS14; Shu18; Gro+17; Kav19].

Despite this recent flurry of developments, a unifying account of modal dependent type theory has yet to emerge. Faced with a new modal situation, a type theorist must handcraft a brand new system, and then prove the usual battery of metatheorems. This introduces formidable difficulties on two levels. First, an increasing number of these applications are *multimodal*: they involve multiple interacting modalities, which significantly complicates the design of

the appropriate judgmental structure. Second, the technical development of each such system is entirely separate, so that one cannot share the burden of proof even between closely related systems. To take a recent example, there is no easy way to transfer the work done in the 80-page-long normalization proof for MLTT_▣ [GSB19a] to a normalization proof for the modal dependent type theory of Birkedal et al. [Bir+20], even though these systems are only marginally different. Put simply, if one wished to prove that type-checking is decidable for the latter, then one would have to start afresh.

We intend to avoid such duplication in the future. Rather than designing a new dependent type theory for some preordained set of modalities, we will introduce a system that is *parametrized* by a *mode theory*, i.e. an algebraic specification of a modal situation. This system, which we call MTT, solves both problems at once. First, by instantiating it with different mode theories we will show that MTT can capture a wide class of situations. Some of these, e.g. the one for guarded recursion, lead to a previously unknown system that improves upon earlier work. Second, the predictable behavior of our rules allows us to prove metatheoretic results about large classes of instantiations of MTT at once. For example, our canonicity theorem applies *irrespective* of the chosen mode theory. As a result, we only need to prove such results *once*. Returning to the previous example, careful choices of mode theory yield two systems that closely resemble the calculi of Birkedal et al. [Bir+20] and MLTT_▣ [GSB19a] respectively, so that our proof of canonicity applies to both.

In fact, we take things one step further: MTT is not just multimodal, but also *multimode*. That is, each judgment of MTT can be construed as existing in a particular *mode*. All modes have some things in common—e.g. there will be dependent sums in each—but some might possess distinguishing features. From a semantic point of view, different modes correspond to different context categories. In this light, modalities intuitively correspond to *functors* between those categories: in fact, they will be structures slightly weaker than *dependent right adjoints* (DRAs) [Bir+20].

Mode theories At a high level, MTT can be thought of as a machine that converts a concrete description of modes and modalities into a type theory. This description, which is often called a *mode theory*, is given in the form of a *small strict 2-category* [Ree09; LS16; LSR17]. A mode theory gives rise to the following correspondence:

object \sim mode

morphism \sim modality

2-cell \sim natural map between modalities

The equations between morphisms and between 2-cells in a mode theory can be used to precisely specify the interactions we want between different modalities. We will illustrate this point with an example.

Instantiating MTT Suppose we have a mode theory \mathcal{M} with a single object m , a single generating morphism $\mu : m \rightarrow m$, and no non-trivial 2-cells. Equipping MTT with \mathcal{M} produces a type theory with a single modal type constructor, $\langle \mu \mid \sqsubset \rangle$. This is the simplest non-trivial setting, and we can prove very little about it without additional 2-cells.

If we add a 2-cell $\varepsilon : \mu \Rightarrow 1$ to \mathcal{M} , we can define a function

$$\text{extract}_A : \langle \mu \mid A \rangle \rightarrow A$$

inside the type theory. If we also add a 2-cell $\delta : \mu \Rightarrow \mu \circ \mu$ then we can also define

$$\text{duplicate}_A : \langle \mu \mid A \rangle \rightarrow \langle \mu \mid \langle \mu \mid A \rangle \rangle$$

Furthermore, we can control the precise interaction between duplicate_A and extract_A by adding more equations that relate ε and δ . For example, we may ask that \mathcal{M} be the *walking comonad* [SS86] which leads to a type theory with a dependent S4-like modality [Pfe01; dPR15; Shu18]. We can be even more specific, e.g. by asking that $(\mu, \varepsilon, \delta)$ be *idempotent*.

Thus, a morphism $\mu : p \rightarrow q$ introduces a modality $\langle \mu \mid \sqsubset \rangle$, and a 2-cell $\alpha : \mu \Rightarrow \nu$ of \mathcal{M} allows the definition of a function of type $\langle \mu \mid A \rangle \rightarrow \langle \nu \mid A \rangle @ q$.

Relation to other modal type theories Most work on modal type theories still defies classification. However, we can informatively position MTT with respect to two qualitative criteria, viz. usability and generality.

Much of the prior work on modal type theory has focused on bolting a specific modality onto a type theory. The benefit of this approach is that the syntax can be designed to be as convenient as possible for the application at hand. For example, spatial/cohesive type theory [Shu18] features two modalities, \flat and \sharp , and is presented in a dual-context style. This judgmental structure, however, is applicable only because of the particular properties of \flat and \sharp . Nevertheless, the numerous pen-and-paper proofs in *op. cit.* demonstrate that the resulting system is easy to use.

At the other end of the spectrum, the framework of Licata-Shulman-Riley (LSR) [LSR17] comprises an extremely general toolkit for simply-typed,

substructural modal type theory. Its dependent generalization, which is currently under development, is able to handle a very large class of modalities. However, this generality comes at a price: its syntax is complex and unwieldy, even in the simply-typed case.

MTT attempts to strike a delicate balance between those two extremes. By avoiding substructural settings and some kinds of modalities we obtain a noticeably simpler apparatus. These restrictions imply that, unlike LSR, we do not need to annotate our term formers with delayed substitutions, and that our system straightforwardly extends to dependent types. Crucially, we ensure that no rule of MTT ‘trims’ the context, which would necessitate either delayed substitutions [Biz+16; LSR17] or often delicate admissible rules [BGM17; Bir+20; GSB19a] in order to ensure the validity of substitution. We also show that MTT can be used for many important examples, and that it is simple enough to be used in pen-and-paper calculations.

Contributions In summary, we make the following contributions:

- We introduce MTT, a general type theory for multiple modes and multiple interacting modalities.
- We define its semantics, which constitute a category of models.
- We prove that a slight extension of MTT satisfies *canonicity*, an important metatheoretic property, through a modern *gluing*¹ argument [Shu14; AK16; Coq18; KHS19].
- We instantiate MTT with various mode theories, and show its value in reasoning about guarded recursion [Biz+16], degrees of relatedness [ND18a], and other modal situations.

For want of space we omit many details and proofs, which can be found in the accompanying technical report [Gra+20a].

5.3.2 The Syntax of MTT

We now present the syntax of MTT. For the rest of section 5.3 we fix an *external* 2-category \mathcal{M} called the mode theory, and use o, p, q, r to stand for modes, μ, ν, τ for modalities, and α, β, γ for 2-cells.

In broad terms, MTT consists of a collection of type theories, one for each mode $m \in \mathcal{M}$. These type theories will eventually appear in one another, but only

¹We adopt the bizarre but seemingly widespread convention to use the spelling ‘gluing’ for the modelling technique and ‘glueing’ to refer to any usage of the unrelated Glue-type.

as spectres under a modality. We thus begin by describing the individual type theories at each mode, and only then discuss how modalities can be used to relate them.

5.3.2.a The Type Theory at Each Mode

Each mode in MTT is inhabited by a standard dependent type theory (DTT), and accordingly includes the usual judgments. For example, we have the judgment $\Gamma \text{ ctx } @ q$ which states that Γ is a well-formed context *in that particular mode* q . There are likewise judgments for types, terms, and substitutions at each mode. Concretely, MTT as a GAT contains a copy of DTT $@ q$ (fig. 3.2) for every mode q , with all the desired types (section 3.2.4).

5.3.2.b Introducing a Modality

Having defined the basic type theory inhabiting each mode, we now show how these type theories interact.

Suppose \mathcal{M} contains a modality $\mu : p \rightarrow q$. We would like to think of μ as a ‘map’ from mode p to mode q . Then, for each $\vdash A \text{ type } @ p$ we would like a type $\vdash \langle \mu \mid A \rangle \text{ type } @ q$. On the level of terms we would similarly like for each $\vdash a : A @ p$ an induced term $\vdash \text{mod}_{\mu} a : \langle \mu \mid A \rangle @ q$.

These constructs would be entirely satisfactory, were it not for the presence of *open terms*. To illustrate the problem, suppose we have a type $\Gamma \vdash A \text{ type } @ p$. We would hope that the corresponding modal type would live in the same context, i.e. that $\Gamma \vdash \langle \mu \mid A \rangle \text{ type } @ q$. However, this is not possible, as Γ is only a context at mode p , and cannot be carried over verbatim to mode q . Hence, the only pragmatic option is to introduce an operation that allows a context to cross over to another mode.

Forming a modal type There are several different proposed solutions to this problem in the literature [e.g. PD01; Clo18]. In the case of MTT we will use a *Fitch-style* discipline [BGM17; Bir+20; GSB19a]: we will require that μ induce an operation on contexts in the *reverse* direction, which we will denote by a *lock*:

$$\frac{\Gamma \text{ ctx } @ q}{\Gamma, \blacksquare_{\mu} \text{ ctx } @ p}$$

Intuitively, \blacksquare_{μ} behaves like a left adjoint to $\langle \mu \mid \sqsubset \rangle$. However, $\langle \mu \mid \sqsubset \rangle$ acts on types while $(\sqsubset, \blacksquare_{\mu})$ acts on contexts, so this cannot be an adjunction. Birkedal

et al. [Bir+20] call this situation a *dependent right adjoint* (DRA). A DRA essentially consists of a type former $\langle R \mid \sqcup \rangle$ and a context operation L such that

$$\{a \mid L\Gamma \vdash a : A\} \cong \{\hat{a} \mid \Gamma \vdash \hat{a} : \langle R \mid A \rangle\}. \quad (5.5)$$

See Birkedal et al. [Bir+20] or section 5.2 for a formal definition.

Just as with DRAs, the MTT formation and introduction rules for modal types effectively *transpose* types and terms across this adjunction:

$$\frac{\Gamma, \mathbf{a}_\mu \vdash A \text{ type}_\ell @ p}{\Gamma \vdash \langle \mu \mid A \rangle \text{ type}_\ell @ q} \quad \frac{\Gamma, \mathbf{a}_\mu \vdash a : A @ p}{\Gamma \vdash \text{mod}_\mu a : \langle \mu \mid A \rangle @ q}$$

It remains to show how to eliminate modal types. Previous work on Fitch-style calculi [Bir+20; GSB19a] has employed elimination rules which essentially invert the introduction rule above. Such rules *remove* one or more locks from the context during type-checking, and sometimes even trim a part of it. For example, a rule of this sort would be

$$\frac{\mathbf{a}_\mu \notin \Theta \quad \Gamma \vdash \hat{a} : \langle \mu \mid A \rangle @ q}{\Gamma, \mathbf{a}_\mu, \Theta \vdash \text{open}_\mu \hat{a} : A @ p}.$$

However, this kind of rule tends to be unruly, and requires delicate work to prove even basic results, such as the admissibility of substitution: see the technical report by Gratzer et al. [GSB19b] for a particularly laborious case. The results in *op. cit.* could not possibly reuse any of the work of Birkedal et al. [Bir+20], as a small change in the syntax leads to many subtle changes in the metatheory. Consequently, it seems unlikely that one could adapt this approach to a modality-agnostic setting like ours.

We will use a different technique, which is reminiscent of dual-context calculi [Kav17]. First, we will let the variable rule control the use of modal variables. Then, we will take a ‘modal cut’ rule, which will allow the substitution of modal terms for modal variables, to be our modal elimination rule.

Accessing a modal variable The behavior of modal types can often be clarified by asking a simple question: when can we use $x : \langle \mu \mid A \rangle$ to construct a term of type A ? In previous Fitch-style calculi we would use the modal elimination rule to reduce the goal to $\langle \mu \mid A \rangle$, and then – *had the modal elimination rule not eliminated x from the context* – we would simply use the variable. We may thus write down a term of type A using a variable $x : \langle \mu \mid A \rangle$ only when our context has the appropriate structure, and the final arbiter of that is the modal elimination rule.

MTT turns this idea on its head: rather than handing control over to the modal elimination rule, we delegate this decision to the variable rule itself. In order to ascertain whether we can use a variable in our calculus, the variable rule examines *the locks to the right of the variable*. The rule of thumb is this: we should always be able to access $\langle \mu \mid A \rangle$ behind $\mathbf{!}_\mu$. Carrying the $(\sqsubset, \mathbf{!}_\mu) \dashv \langle \mu \mid \sqsubset \rangle$ analogy further, we see that the simplest judgment that fits this, namely $\Gamma, x : \langle \mu \mid A \rangle, \mathbf{!}_\mu \vdash x : A @ p$, corresponds to the *counit*.

To correctly formulate the variable rule, we will require one more idea: following modal type theories based on *left division* [Pfe01; Abe06; Abe08; NVD17a; ND18a], every variable in the context will be annotated with a modality, $\mu \mid x : A$. Intuitively a variable $\mu \mid x : A$ is the same as a variable $\hat{x} : \langle \mu \mid A \rangle$ (which in turn is shorthand for $1 \mid \hat{x} : \langle \mu \mid A \rangle$), but the annotations are part of the structure of a context while $\langle \mu \mid A \rangle$ is a type. This small circumlocution will ensure that the variable rule respects substitution.

The most general form of the variable rule will be able to handle the interaction of modalities, so we present it in stages. A first ‘counit-like’ approximation is then

$$\frac{\mathbf{!} \notin \Delta \quad \Gamma, \mathbf{!}_\mu \vdash A \text{ type } @ p}{\Gamma, \mu \mid x : A, \mathbf{!}_\mu, \Delta \vdash x : A @ p}$$

The first premise expresses that no further locks occur in Δ .

Ticks Just as we assign names to variables so that we do not have to use numbers and count variables to know what we are talking about, it will also be useful in complicated settings to have names to refer to locks in the context. So when we extend a context with a lock, we will assign it a name m and write $\Gamma, \mathbf{!}_\mu^m$. Following [BGM17], we call these names *ticks*. For almost all purposes (including formalizing MTT as a GAT), ticks can be ignored, but we believe they are of great help when modal bookkeeping is being performed by a human. By consequence, since the arguments A and a of $\langle \mu \mid^m A \rangle$ and $\text{mod}_\mu^m a$ are checked with an additional lock in the context, these operators bind a tick.

Modal context extension The switch to modality-annotated declarations $\mu \mid x :^m A$ also requires us to revise the context extension rule. The revised version below closely follows the formation rule for $\langle \mu \mid^m A \rangle$: if A is a type in the locked context $\Gamma, \mathbf{!}_\mu^m$, then we may extend Γ to include a declaration $\mu \mid x :^m A$, so that x stands for a term of type A *under the modality* μ :

$$\frac{\Gamma, \mathbf{!}_\mu^m \vdash A \text{ type } @ p}{\Gamma, \mu \mid x :^m A \text{ ctx } @ q}$$

The elimination rule The difference between a modal type $\langle \mu \mid^m A \rangle$ and an annotated declaration $\mu \mid x :^m A$ in the context is navigated by the modal elimination rule. In brief, its role is to enable the substitution of a term of the former type for a variable with the latter declaration. The full rule is complex, so in this section we will only discuss the case of a single modality, $\mu : n \rightarrow m$. The rule for this μ is

$$\frac{\Gamma, \mathbf{lock}_\mu^m \vdash A \text{ type } @ p \quad \Gamma, 1 \mid \hat{x} : \langle \mu \mid^m A \rangle \vdash T \text{ type } @ q \quad \Gamma \vdash \hat{a} : \langle \mu \mid^m A \rangle @ q \quad \Gamma, \mu \mid x :^m A \vdash t : T[\text{mod}_\mu x / \hat{x}] @ q}{\Gamma \vdash \text{let } (\text{mod}_\mu x = \hat{a}) \text{ in } t : T[\hat{a} / \hat{x}] @ q} \quad (5.6)$$

Forgetting dependence for a moment, this rule is close to the dual context style [PD01; Kav17]: if we think of annotations as separating the context into multiple zones, then $\mu \mid x :^m A$ clearly belongs to the ‘modal’ part.

In the dependent case we also need a motive T , which depends on a variable of modal type, but under the identity modality 1. This premise is then fulfilled by \hat{a} in the conclusion. In a sense, this rule permits a form of *modal induction*: every variable $1 \mid \hat{x} : \langle \mu \mid^m A \rangle$ can be assumed to be of the form $\text{mod}_\mu x$ for some $\mu \mid x :^m A$. This kind of rule has appeared before in dependent modal type theory, mainly in the work of Shulman [Shu18].

In the type theory of Birkedal et al. [Bir+20] modalities are taken to be dependent right adjoints, with terms witnessing eq. (5.5). This isomorphism can encode eq. (5.6), but eq. (5.6) cannot encode eq. (5.5). As a result, modalities in MTT are weaker than DRAs. For this reason, we shall call them *weak DRAs*.

5.3.2.c Multiple Modalities

Thus far we have only considered a single modality. In this section we discuss the small changes that are needed to enable MTT to support multiple interacting modalities. The final version of the modal rules is given in figs. 5.5 and 5.6.

Multimodal locks We have so far only used the operation $(\sqcup, \mathbf{lock}_\mu)$ on contexts for the single modality $\mu : p \rightarrow q$. This operation should also work for any modality with the same rule LOCK, hence inducing an action of locks on contexts that is contravariant with respect to the mode. The only question, then, is how these locks should interact. This is where the mode theory comes in: locks should be *functorial*, so that $\nu : q \rightarrow r$, $\mu : p \rightarrow q$, and $\Gamma \text{ ctx } @ r$ imply $(\Gamma, \mathbf{lock}_\mu^m, \mathbf{lock}_\nu^n) = (\Gamma, \mathbf{lock}_{\mu \circ \nu}^{mn}) \text{ ctx } @ p$. This brings us in a pesky situation regarding the ticks, but we save ourselves by taking the liberty to use strings like mn as ticks

Prerequisites: DTT @ p for each $p \in \mathcal{M}$ (fig. 3.2), with all the desired types (section 3.2.4).

Locked context:

Note: \downarrow_m^m is shorthand for $1\downarrow_m^m$.

$$\begin{array}{c}
 \text{LOCK} \\
 \frac{\Gamma \text{ctx} @ q \quad \mu : p \rightarrow q}{\Gamma, \mathbf{\mu}_\mu^m \text{ctx} @ p} \\
 \text{where } \Gamma = \Gamma, \mathbf{\mu}_1^* \quad (\text{LOCK:ID}) \\
 \Gamma, \mathbf{\mu}_\nu, \mathbf{\mu}_\mu^m = \Gamma, \mathbf{\mu}_{\nu \circ \mu}^{nm} \quad (\text{LOCK:COMP}) \\
 \\
 \text{LOCK:FMAP} \\
 \frac{\sigma : \Gamma \rightarrow \Delta @ q \quad \mu, \nu : p \rightarrow q \quad \alpha : \mu \Rightarrow \nu}{(\sigma, \alpha \downarrow_n^m) : (\Gamma, \mathbf{\mu}_\nu^m) \rightarrow (\Delta, \mathbf{\mu}_\mu^m) @ p} \\
 \text{where } (\sigma, \alpha \downarrow_n^m) \circ (\tau, \beta \downarrow_o^m) = (\sigma \circ \tau, (\beta \circ \alpha) \downarrow_o^m) \quad (\text{LOCK:FMAP:COMP}) \\
 \text{id} = (\text{id}, \downarrow_m^m) \quad (\text{LOCK:FMAP:ID}) \\
 (\sigma, \alpha' \downarrow_{n'}^m, \alpha \downarrow_n^m) = (\sigma, (\alpha' * \alpha) \downarrow_{n'}^m) \quad (\text{LOCK:COMP:FMAP}) \\
 \sigma = (\sigma, 1_1 \downarrow_o^*) \quad (\text{LOCK:ID:FMAP})
 \end{array}$$

Modal context extension:

We consider the non-modal rule CTX-EXT and its introduction, elimination and computation rules as a special case of CTX-MODEXT for $p = q$ and $\mu = 1$.

$$\begin{array}{c}
 \text{CTX-MODEXT} \\
 \frac{\Gamma \text{ctx} @ q \quad \mu : p \rightarrow q \quad \Gamma, \mathbf{\mu}_\mu^m \vdash T \text{type} @ p}{\Gamma, \mu \vdash x : {}^m T \text{ctx} @ q} \\
 \\
 \text{CTX-MODEXT:VAR} \\
 \frac{\Gamma, \mathbf{\mu}_\mu^m \vdash T \text{type} @ p \quad \alpha : \mu \Rightarrow \text{locks}(\Delta)}{\Gamma, \mu \vdash x : {}^m T, \Delta \vdash x \alpha \downarrow_m : T[\alpha \downarrow_{\text{ticks}(\Delta)}^m] @ p} \\
 \text{where } (x \alpha \downarrow_{\text{ticks}(\Delta)})[\sigma, t/x \downarrow_m] = t[\alpha \downarrow_{\text{ticks}(\Delta)}^m] \quad (\text{CTX-MODEXT:VAR:BETA}) \\
 \text{where } (x \alpha \downarrow_{\text{ticks}(\Delta)})[\beta \downarrow_{\text{ticks}(\Theta)}^{\text{ticks}(\Delta)}] = x(\beta \circ \alpha) \downarrow_{\text{ticks}(\Theta)} \quad (\text{CTX-MODEXT:VAR:2CELL}) \\
 \\
 \text{CTX-MODEXT:WKN} \\
 \frac{\Gamma \text{ctx} @ q \quad \mu : p \rightarrow q \quad \Gamma, \mathbf{\mu}_\mu^m \vdash T \text{type} @ p}{\pi_x : (\Gamma, \mu \vdash x : {}^m T) \rightarrow \Gamma @ q} \\
 \text{where } \pi_x \circ (\sigma, t/x \downarrow_m) = \sigma \quad (\text{CTX-MODEXT:WKN:BETA}) \\
 \\
 \text{CTX-MODEXT:INTRO} \\
 \frac{\sigma : \Gamma \rightarrow \Delta @ q \quad \mu : p \rightarrow q \quad \Gamma, \mathbf{\mu}_\mu^m \vdash t : T[\sigma] @ p}{(\sigma, t/x \downarrow_m) : \Gamma \rightarrow (\Delta, \mu \vdash x : {}^m T) @ q} \\
 \text{where } \tau = (\pi_x \circ \sigma, (x \downarrow_m)[\tau]/x \downarrow_m) \quad (\text{CTX-MODEXT:ETA}) \\
 (\sigma, t/x \downarrow_m) \circ \rho = (\sigma \circ \rho, t[\rho, 1\downarrow_m^m]/x \downarrow_m)
 \end{array}$$

Locks in a telescope:

$$\begin{array}{lll}
 \text{locks}(\cdot) = 1 & \text{locks}(\Delta, \mathbf{\mu}_\mu^m) = \text{locks}(\Delta) \circ \mu & \text{locks}(\Delta, \mu \vdash x : {}^m T) = \text{locks}(\Delta) \\
 \text{ticks}(\cdot) = \bullet & \text{ticks}(\Delta, \mathbf{\mu}_\mu^m) = \text{ticks}(\Delta) \mathfrak{m} & \text{ticks}(\Delta, \mu \vdash x : {}^m T) = \text{ticks}(\Delta)
 \end{array}$$

Figure 5.5: Structural rules of MTT.

Prerequisites: Structural rules of MTT (fig. 5.5).

$$\begin{array}{c}
 \text{WDRA} \\
 \frac{\mu : p \rightarrow q \quad \Gamma, \mathbf{\blacklozenge}_\mu^m \vdash A \text{ type}_\ell @ p}{\Gamma \vdash \langle \mu \mid^m A \rangle \text{ type}_\ell @ q} \\
 \\
 \text{WDRA:INTRO} \\
 \frac{\mu : p \rightarrow q \quad \Gamma, \mathbf{\blacklozenge}_\mu^m \vdash a : A @ p}{\Gamma \vdash \text{mod}_\mu^m a : \langle \mu \mid^m A \rangle @ q} \\
 \\
 \text{WDRA:ELIM} \\
 \frac{\begin{array}{l} \mu : p \rightarrow q \quad \nu : q \rightarrow r \\ \Gamma, \mathbf{\blacklozenge}_\nu^n \vdash \hat{a} : \langle \mu \mid^m A \rangle @ q \\ \Gamma, \nu \mid \hat{x} :^n \langle \mu \mid^m A \rangle \vdash T \text{ type } @ r \\ \Gamma, \nu \circ \mu \mid x :^m A \vdash t : T[\text{mod}_\mu^m x \downarrow_{nm} / \hat{x} \downarrow_n] @ r \end{array}}{\Gamma \vdash \text{let}_\nu^n (\text{mod}_\mu^m (x \downarrow_{nm}) = \hat{a}) \text{ in } t : T[\hat{a} / \hat{x} \downarrow_n] @ r} \\
 \text{where } \text{let}_\nu^n (\text{mod}_\mu^m (x \downarrow_{nm}) = \hat{a}) \text{ in } t = t[a/x \downarrow_{nm}] \quad (\text{WDRA:BETA})
 \end{array}$$

Figure 5.6: Typing rules for the modal type (weak DRA).

(recall that ticks are solely there for the sake of readability). We additionally ask that the identity modality $1 : p \rightarrow p$ at each mode has a trivial, invisible action on contexts, i.e. $\Gamma = (\Gamma, \mathbf{\blacklozenge}_1^\bullet) \text{ ctx } @ p$ (using the empty string \bullet as a tick).

These two actions, which are encoded by `LOCK:COMP` and `LOCK:ID`, ensure that $\mathbf{\blacklozenge}$ is a contravariant functor on \mathcal{M} , mapping each mode p to the category of contexts $\Gamma \text{ ctx } @ p$. The contravariance originates from the fact that \mathcal{M} is a specification of the behaviour of the modalities $\langle \mu \mid \sqcup \rangle$, so that their left-adjoint-like counterparts $(\sqcup, \mathbf{\blacklozenge}_\mu)$ act with the opposite variance.

The full variable rule We have seen that $\mathbf{\blacklozenge}$ induces a functor from \mathcal{M} to categories of contexts, but we have not yet used the 2-cells of \mathcal{M} . In short, a 2-cell $\alpha : \mu \Rightarrow \nu$ contravariantly induces a substitution from $(\Gamma, \mathbf{\blacklozenge}_\nu^m)$ to $(\Gamma, \mathbf{\blacklozenge}_\mu^m)$ (`LOCK:FMAP`), which we discuss this further in section 5.3.4.

In section 3.2, we formulated the variable rule `CTX-EXT:VAR` without weakening included. This means that the expression $x[\pi_y]$ cannot be reduced to a substitution-free expression. Similarly, here, we could state the variable rule in its most bare form:

$$\begin{array}{c}
 \text{CTX-MODEXT:VAR:BARE} \\
 \frac{\Gamma, \mathbf{\blacklozenge}_\mu^m \vdash T \text{ type } @ p}{\Gamma, \mu \mid x :^m T, \mathbf{\blacklozenge}_\mu^m \vdash x \downarrow_m : T @ p}
 \end{array}$$

However, for MTT, we prefer to give a variable rule which already incorporates a combination of weakenings and 2-cell substitutions, because it clarifies how the variable rule is checked in practice and also simplifies a discussion of the interaction of 2-cells and variables. The above rule `CTX-MODEXT:VAR:BARE` requires that the lock and the variable annotation are an exact match. We relax this requirement by allowing for a mediating 2-cell:

$$\frac{\Gamma, \mathbf{a}_{\mu}^m \vdash T \text{ type } @ p \quad \alpha : \mu \rightarrow \nu}{\Gamma, \mu \mid x :^m T, \mathbf{a}_{\nu}^n \vdash x \alpha \downarrow_n : T @ p} \quad (5.7)$$

The annotation $\alpha \downarrow_n$ on x is now part of the syntax: each variable must be annotated with a 2-cell, though we will still write x to mean $x \downarrow_{\circ}$. The final form of the variable rule, which appears as `CTX-MODEXT:VAR` in fig. 5.5, is only a slight generalization which allows the variable to occur at positions other than the very front of the context. In fact, `CTX-MODEXT:VAR` can be reduced to the one in eq. (5.7) by using weakening to remove variables to the right of x , and then invoking functoriality to fuse all the locks to the right of x into a single one with modality locks(Δ).

The full elimination rule Recall that the elimination rule for a single modality (eq. (5.6)) allowed us to plug a term of type $\langle \mu \mid^m A \rangle$ for an assumption $\mu \mid x :^m A$. Some additional generality is needed to cover the case where the motive $\nu \mid \hat{x} :^n \langle \mu \mid^m A \rangle \vdash T \text{ type } @ r$ depends on x under a modality $\nu \neq 1$. This is where the composition of modalities in \mathcal{M} comes in handy: our new rule will use it to absorb ν by replacing the assumption $\nu \mid \hat{x} :^n \langle \mu \mid^m A \rangle$ with $\nu \circ \mu \mid x :^{nm} A$. The new rule, `WDRA:ELIM`, is given in fig. 5.5. The simpler rule may be recovered by setting $\nu = 1$.

Modal Π -types In the technical report we have supplemented MTT with a primitive *modal dependent product* type, $(\mu \mid x :^m A) \rightarrow B$, which bundles together $\langle \mu \mid _ \rangle$ and the ordinary Π -type. If we ignore η -equality, $(\mu \mid x :^m A) \rightarrow B$ can be defined as $(\hat{x} : \langle \mu \mid^m A \rangle) \rightarrow \text{let } (\text{mod}_{\mu}^m(x \downarrow_m) = \hat{x}) \text{ in } B$. This modal Π -type is convenient for programming but it is not essential, so we defer further discussion to the technical report [Gra+20a].

Definitional equality in MTT A perennial problem in type theory is that of deciding where the boundary between derivable and *definitional* equalities should lie. We have followed standard practices regarding definitional equalities for dependent products, sums, etc. The situation is somewhat more complicated regarding modal types. On the one hand, we have expected β -rule `WDRA:BETA`. On the other hand, we do not include any definitional η -rules: as the eliminator is

Prerequisites: Structural rules of MTT (fig. 5.5).

Generalizes: Π -types in DTT if $\mu = 1$ (fig. 3.3).

$$\begin{array}{c}
 \text{MODPI} \\
 \frac{\Gamma, \mathbf{\mu}_\mu^m \vdash A \text{ type}_\ell @ p \quad \mu : p \rightarrow q}{\Gamma, \mu \upharpoonright x :^m A \vdash B \text{ type}_\ell @ q} \\
 \hline
 \Gamma \vdash (\mu \upharpoonright x :^m A) \rightarrow B \text{ type}_\ell @ q \\
 \\
 \text{MODPI:INTRO} \\
 \frac{\Gamma, \mu \upharpoonright x :^m A \vdash b : B @ q \quad \mu : p \rightarrow q}{\Gamma \vdash \lambda(\mu \upharpoonright x).b : (\mu \upharpoonright x :^m A) \rightarrow B @ q} \\
 \text{where } \lambda(\mu \upharpoonright x).f \cdot^m x \downarrow_m = f \quad (\text{MODPI:ETA}) \\
 \\
 \text{MODPI:ELIM} \\
 \frac{\Gamma \vdash f : (\mu \upharpoonright x :^m A) \rightarrow B @ q \quad \Gamma, \mathbf{\mu}_\mu^m \vdash a : A @ p \quad \mu : p \rightarrow q}{\Gamma \vdash f \cdot^m a : B[a/x] @ q} \\
 \text{where } (\lambda(\mu \upharpoonright x).b) \cdot^m a = b[a/x \downarrow_m] \quad (\text{MODPI:BETA})
 \end{array}$$

Figure 5.7: Typing rules for Π -types in MTT.

a *positive* pattern-matching construct, the proper η -rule would need *commuting conversions*, which would enormously complicate the metatheory.²

5.3.3 Programming with Modalities

In this section we show how MTT can be used to program and reason with modalities. We develop a toolkit of modal combinators, which we then use in section 5.3.3.b to show how MTT can be effortlessly used to present an idempotent comonad.

5.3.3.a Modal Combinators

We first show how each 2-cell $\alpha : \mu \Rightarrow \nu$ with $\mu, \nu : p \rightarrow q$ induces a natural transformation $\langle \mu \mid \sqcup \rangle \rightarrow \langle \nu \mid \sqcup \rangle$. Given $\Gamma, \mathbf{\mu}_\mu^m \vdash A \text{ type} @ q$, we define

$$\begin{aligned}
 \text{coe}[\mu : \nu \Rightarrow \alpha] &: \langle \mu \mid^m A \rangle \rightarrow \langle \nu \mid^n A[\alpha \downarrow_m^n] \rangle \\
 \text{coe}[\mu : \nu \Rightarrow \alpha] \hat{x} &:= \text{let } (\text{mod}_\mu^m (x \downarrow_m) = \hat{x}) \text{ in } \text{mod}_\nu^n (x \alpha \downarrow_m).
 \end{aligned}$$

²But see proposition 7.3.3.

With this operation, we have completed the correspondence from section 5.3.1: objects of \mathcal{M} correspond to modes, morphisms to modalities, and 2-cells to coercions.

We can also show that the assignment $\mu \mapsto \langle \mu \mid \sqsubset \rangle$ is, in some sense, *functorial*. Unlike the action of locks, this functoriality is not definitional, but only a type-theoretic *equivalence* [Uni13, §4]. Fixing $\Gamma, \mathbf{a}_{\nu}^n, \mathbf{a}_{\mu}^m \vdash A \text{ type } @ p$, let

$$\mathbf{comp}_{\nu, \mu} : \langle \nu \mid^n \langle \mu \mid^m A \rangle \rangle \rightarrow \langle \nu \circ \mu \mid^{nm} A \rangle$$

$$\mathbf{comp}_{\nu, \mu} \hat{x} = \mathbf{let} (\mathbf{mod}_{\nu}^n (\hat{x} \downarrow_n) = \hat{x}) \mathbf{in} \mathbf{let}_{\nu}^n (\mathbf{mod}_{\mu}^m (x \downarrow_{nm}) = \hat{x} \downarrow_n) \mathbf{in} \mathbf{mod}_{\nu \circ \mu}^{nm} (x \downarrow_{nm})$$

$$\mathbf{comp}_{\nu, \mu}^{-1} : \langle \nu \circ \mu \mid^{nm} A \rangle \rightarrow \langle \nu \mid^n \langle \mu \mid^m A \rangle \rangle$$

$$\mathbf{comp}_{\nu, \mu}^{-1} \hat{x} = \mathbf{let} (\mathbf{mod}_{\nu \circ \mu}^n (x \downarrow_{\circ}) = \hat{x}) \mathbf{in} \mathbf{mod}_{\nu}^n (\mathbf{mod}_{\mu}^m (x \downarrow_{nm})).$$

Even in this small example the context equations that involve locks are essential: for $\langle \nu \circ \mu \mid^{nm} A \rangle$ to be a valid type we need that $(\Gamma, \mathbf{a}_{\nu}^n, \mathbf{a}_{\mu}^m) = (\Gamma, \mathbf{a}_{\nu \circ \mu}^{nm})$, which is ensured by `LOCK:COMP`. Additionally, observe that $\mathbf{comp}_{\nu, \mu}$ relies crucially on the multimodal elimination rule `WDRA:ELIM`: we must pattern-match on \hat{x} , which is under ν in the context.

These combinators are only propositionally inverse. In one direction, the proof is

$$_ : (\hat{x} : \langle \nu \mid^n \langle \mu \mid^m A \rangle \rangle) \rightarrow \hat{x} \equiv \mathbf{comp}_{\nu, \mu}^{-1} (\mathbf{comp}_{\nu, \mu} \hat{x})$$

$$_ \hat{x} = \mathbf{let} (\mathbf{mod}_{\nu}^n (\hat{x} \downarrow_n) = \hat{x}) \mathbf{in} \mathbf{let}_{\nu}^n (\mathbf{mod}_{\mu}^m (x \downarrow_{nm}) = \hat{x} \downarrow_n) \mathbf{in} \mathbf{refl}_{\mathbf{mod}_{\nu}^n (\mathbf{mod}_{\mu}^m (x \downarrow_{nm}))}.$$

This is a typical example of reasoning about modalities: we use the modal elimination rule to induct on a modally-typed term. This reduces it to a term of the form `mod`, and the result follows definitionally. It is equally easy to construct an equivalence $\langle 1 \mid^{\circ} A \rangle \simeq A$.

As a final example, we will show that each modal type satisfies *axiom* K ,³ a central axiom of Kripke-style modal logics. This combinator will be immediately recognizable to functional programmers as the term that shows that $\langle \mu \mid - \rangle$ is an *applicative functor* [MP08].

$$\sqsubset \otimes_{\mu} \sqsubset : \langle \mu \mid^m A \rightarrow B \rangle \rightarrow \langle \mu \mid^m A \rangle \rightarrow \langle \mu \mid^m B \rangle$$

$$\hat{f} \otimes_{\mu} \hat{a} = \mathbf{let} (\mathbf{mod}_{\mu}^m (f \downarrow_m) = \hat{f}) \mathbf{in} \mathbf{let} (\mathbf{mod}_{\mu}^m (a \downarrow_m) = \hat{a}) \mathbf{in} \mathbf{mod}_{\mu}^m (f \downarrow_m (a \downarrow_m)).$$

We can also define a stronger combinator, which corresponds to a dependent form of the Kripke axiom [Bir+20], and which generalizes \otimes_{μ} to dependent products $(x : A) \rightarrow B(x)$.

³The modal one, that is, not uniqueness of identity proofs.

5.3.3.b Idempotent Comonads in MTT

A great deal of prior work in modal type theory has focused on *comonads* [PD01; dPR15; Shu18; GSB19a], and in particular *idempotent* comonads. Shulman [Shu18, Theorem 4.1] has shown that such modalities necessitate changes to the judgmental structure, as the only idempotent comonads that are internally definable are of the form $\sqcup \times U$ for some proposition U . In this section we present a mode theory for idempotent comonads, and prove that the resulting type theory internally satisfies the expected equations using just the combinators of the previous section.

We define the mode theory \mathcal{M}_{ic} to consist of a single mode p , and a single non-trivial morphism $\mu : p \rightarrow p$. We will enforce idempotence by setting $\mu \circ \mu = \mu$. Finally, we include a unique non-trivial 2-cell $\varepsilon : \mu \Rightarrow 1$. We force uniqueness of this 2-cell by imposing equations such as $1_\mu \star \varepsilon = \varepsilon \star 1_\mu = 1$. The resulting mode theory is a 2-category, albeit a very simple one: it is in fact only a *poset-enriched* category.

We can show that $\langle \mu \mid \sqcup \rangle$ is a comonad by defining the expected operations using the combinators of section 5.3.3.a:

$$\begin{aligned} \text{dup}_A : \langle \mu \mid^m A \rangle &\rightarrow \langle \mu \mid^m \langle \mu \mid^n A[\downarrow_{mn}^m] \rangle \rangle & \text{extract}_A : \langle \mu \mid^m A \rangle &\rightarrow A[\varepsilon \downarrow_\bullet^m] \\ \text{dup}_A &= \text{comp}_{\mu, \mu} & \text{extract}_A &= \text{coe}[\mu : 1 \Rightarrow \varepsilon] \end{aligned}$$

Note that the substitution \downarrow_{mn}^m is the identity after desugaring (i.e. erasing) ticks.

We must also show that dup_A and extract_A satisfy the comonad laws, but that automatically follows from general facts pertaining to coe and comp .⁴ This is indicative of the benefits of using MTT: every general result about it also applies to this instance, including the canonicity theorem of section 5.3.5.

5.3.4 The Substitution Calculus of MTT

Until this point we have presented a curated, high-level view of MTT, and we have avoided any discussion of its metatheory. Yet, these syntactic aspects can be quite complex, and have historically proven to be sticking points for modal type theory. While these details are not necessary for the casual reader, it is essential to validate that MTT is syntactically well-behaved, enjoying e.g. a substitution principle.

⁴In particular, our modal combinators satisfy a variant of the *interchange law* of a 2-category.

We have opted for a modern approach in the analysis of MTT by presenting it as a *generalized algebraic theory* (GAT) [Car86; KKA19]. While this simplifies the study of its semantics (see section 5.3.5), it can also be used to study the syntax. For example, the formulation of MTT as a GAT naturally leads us to include *explicit substitutions* [Mar92; Gra13] in the syntax. Thus, substitution in MTT is not a metatheoretic operation on raw terms, but a piece of the syntax. This presentation helps us carefully state the equations that govern substitutions and their interaction with type formers. We consequently obtain an elegant *substitution calculus*, which can often be quite complex for modal type theories. We only discuss the modal aspects of substitution here; the full calculus is discussed in the technical report [Gra+20a].

Modal substitutions In addition to the usual rules, MTT features substitutions corresponding to the 1- and 2-cells of the mode theory. First, recall that for each modality $\mu : p \rightarrow q$ we have the operation \mathbf{A}_μ on contexts. We make this operation, as an action on pairs of a context and a modality, bifunctorial (LOCK:FMAP). At the same time, we ensure that $\mathbf{A} : \mathcal{M}^{\text{coop}} \rightarrow \text{Cat}$ is a strict 2-functor (LOCK:ID:FMAP, LOCK:COMP:FMAP); see fig. 5.5.

While it is no longer necessary to prove that substitution is *admissible*, we would like to show that explicit substitutions can be pushed inside terms, and ultimately eliminated on closed terms. The proof of canonicity (theorem 5.3.5) implicitly contains such an algorithm, but it is overkill: a simple argument directly proves that all explicit substitutions can be propagated down to variables.

Pushing substitutions under modalities In order for the aforementioned algorithm to work, we must specify how substitutions commute with the modal connectives of MTT. Unlike previous work [GSB19b], the necessary equations are straightforward:

$$\begin{aligned} \langle \mu \mid^m A \rangle [\sigma] &= \langle \mu \mid^m A[\sigma, \downarrow_m^m] \rangle, \\ (\text{mod}_\mu^m a) [\sigma] &= \text{mod}_\mu^m (a[\sigma, \downarrow_m^m]). \end{aligned}$$

This simplicity is not coincidental. Previous modal type theories included rules that, in one way or another, *trimmed* the context during type checking: some removed variables [Pra65; PD01; Shu18], while others erased context formers, e.g. locks [Bir+20; GSB19a]. In either case, it was necessary to show that the trimming operation, which we may write as $\|\Gamma\|$, is functorial: $\Gamma \vdash \delta : \Delta$ should imply $\|\Gamma\| \vdash \|\delta\| : \|\Delta\|$. Unfortunately, the proof of this fact is almost always very complicated. Some type theories avoid it by ‘forcing’ substitution to be

admissible using delayed substitutions [BdP00; LSR17], but this causes serious complications to the equational theory.

MTT circumvents this by avoiding any context trimming. As a result, we need neither delayed substitutions nor a complex proof of admissibility.

5.3.5 The Semantics of MTT

As mentioned in section 5.3.4, we have structured MTT as a GAT. As a result, MTT *automatically* induces a category of models and (strict) homomorphisms between them [Car86; KKA19]. However, this notion of model follows the syntax quite closely. In order to work with it more effectively we factor it into pieces, using the more familiar definition of *categories with families* (CwFs) [Dyb96].⁵ We will then use this notion of model to present a *semantic* proof of canonicity via gluing [Shu14; AK16; Coq18; KHS19].

Like MTT itself, the definition of model is parametrized by a mode theory, so we fix a mode theory \mathcal{M} .

Mode-local structure Recall that MTT is divided into several modes, each of which is closed under the standard connectives of DTT. Accordingly, a model of MTT requires a CwF $(\text{Ctx}_p, \text{Ty}_p, \text{Tm}_p)$ for each mode $p \in \mathcal{M}$. Each CwF is required to be a model of DTT with Σ -, Π - and identity types, and a Coquand-style universe. This part of the definition is entirely standard, and can be found in the literature (see chapter 3) [Dyb96; Hof97; Awo18]. The novel portion of an MTT model describes the relations between CwFs induced by the 1- and 2-cells of \mathcal{M} .

Locks Recall that for $\Gamma \text{ctx} @ q$ and $p : q \rightarrow$ we have a context $\Gamma, \mathbf{a}_\mu^m \text{ctx} @ p$, and that this construction extends functorially to substitutions. Hence, we will require for each modality $p : q \rightarrow$ a functor $\llbracket \mathbf{a}_\mu \rrbracket : \text{Ctx}_q \rightarrow \text{Ctx}_p$. Similarly, each $\alpha : \mu \Rightarrow \nu$ induces a natural transformation from $(\Gamma, \mathbf{a}_\nu^m)$ to $(\Gamma, \mathbf{a}_\mu^m)$. Accordingly, a model should come with a natural transformation $\llbracket \alpha \downarrow \rrbracket : \llbracket \mathbf{a}_\nu \rrbracket \rightarrow \llbracket \mathbf{a}_\mu \rrbracket$. Moreover, the equalities of fig. 5.5 require that the assignments $\mu \mapsto (\sqsubset, \mathbf{a}_\mu)$ and $\alpha \mapsto (\sqsubset, \alpha \downarrow)$ be strictly 2-functorial. Thus, this part of the model can be succinctly summarized by requiring a 2-functor $\text{Ctx}_- : \mathcal{M}^{\text{coop}} \rightarrow \text{Cat}$. The contravariance accounts for the fact μ corresponds to $\langle \mu \mid - \rangle$, but that the functor $\llbracket \mathbf{a}_\mu \rrbracket \text{ models } (\sqsubset, \mathbf{a}_\mu)$, which acts with the opposite variance.

⁵In the technical report [Gra+20a] we have used a more categorical presentation of CwFs, known as *natural models* [Awo18]. However, in the interest of clarity we state our results in terms of CwFs here.

Modal comprehension structure Variable declarations in MTT are annotated with a modality, and the context extension rule CTX-MODEXT involves locks. Thus, our CwFs should be equipped with more structure than mere context extension to support it.

Recall from definition 3.2.3 that, in an ordinary CwF \mathcal{C} , given a context $\Gamma \in \mathcal{C}$ and a type $A \in \text{Ty}(\Gamma)$ we have a context $\Gamma.A$ along with a substitution $\pi : \Gamma.A \rightarrow \Gamma$, and a term $\xi \in \text{Tm}(\Gamma.A, A[\pi])$.

To model MTT we need a modal comprehension operation, which for each context $\Gamma \in \text{Ctx}_q$, modality $\mu : p \rightarrow q$, and type $A \in \text{Ty}_p(\llbracket \blacksquare_\mu \rrbracket(\Gamma))$ yields

- a context $\Gamma.(\mu \mid A) \in \text{Ctx}_q$,
- a substitution $\pi : \Gamma.(\mu \mid A) \rightarrow \Gamma$, and
- a term $\xi \in \text{Tm}_p(\llbracket \blacksquare_\mu \rrbracket(\Gamma.(\mu \mid A)), A[\llbracket \blacksquare_\mu \rrbracket(\pi)])$

where $\Gamma.(\mu \mid A)$ is universal in an appropriate sense.

Intuitively, ξ corresponds to $\text{CTX-MODEXT:VAR:BARE}$. As mentioned before, this suffices to model the full variable rule CTX-MODEXT:VAR , as π , $\llbracket \alpha \downarrow \rrbracket$, and ξ can be used to define it from $\text{CTX-MODEXT:VAR:BARE}$.

Modal types The interpretation of the modal type $\langle \mu \mid _ \rangle$ for a modality $\mu : p \rightarrow q$ requires operations for the formation, introduction, and elimination rules. Just as with the other connectives, these are a direct translation of the rules WDRA , WDRA:INTRO , and WDRA:ELIM to the language of CwFs. For example, for every $\Gamma \in \text{Ctx}_q$, $A \in \text{Ty}_p(\llbracket \blacksquare_\mu \rrbracket(\Gamma))$, and $a \in \text{Tm}_p(\llbracket \blacksquare_\mu \rrbracket(\Gamma), A)$, we require $\text{mod}_\mu a \in \text{Tm}_q(\Gamma, \langle \mu \mid A \rangle)$.

Conclusions This discussion leads to the following definition.

Definition 5.3.1. A model of MTT is a 2-functor $\text{Ctx}_\square : \mathcal{M}^{\text{coop}} \rightarrow \text{Cat}$, equipped with the following structure:

- for each $p \in \mathcal{M}$, a CwF $(\text{Ctx}_p, \text{Ty}_p, \text{Tm}_p)$ that is closed under Π -, Σ - and identity types as well as U ,
- a modal comprehension structure for \mathcal{M} on these CwFs, and
- for each modality $\mu : p \rightarrow q$, a modal type structure $(\langle \mu \mid _ \rangle, \text{mod}_\mu, \text{open})$ modelling the weak DRA.

Definition 5.3.2. A morphism between models $F : \mathcal{C}_\square \rightarrow \mathcal{D}_\square$ is a strict 2-natural transformation such that each $F_p : \mathcal{C}_p \rightarrow \mathcal{D}_p$ is part of a strict CwF morphism $[\text{CCD17}]$ which strictly preserves modal comprehension and types.

We observed in section 5.3.2.c that modalities in MTT are weaker than DRAs [Bir+20].⁶ Since DRAs are often easier to construct, we make this relation formal.

Theorem 5.3.3. A 2-functor $\text{Ctx}_{\sqsubseteq} : \mathcal{M}^{\text{coop}} \rightarrow \text{Cat}$ satisfying the following two conditions induces a model of MTT:

1. for each $p \in \mathcal{M}$, there is a CwF $(\text{Ctx}_p, \text{Ty}_p, \text{Tm}_p)$ that is closed under Pi -, Σ - and identity types as well as \mathbb{U} ,
2. for each $\mu : p \rightarrow q$, the functor $\llbracket \mu \rrbracket : \text{Ctx}_q \rightarrow \text{Ctx}_p$ has a DRA.

In practice virtually all the models of MTT that we consider will be constructed by applying theorem 5.3.3. We can also use it to immediately prove consistency:

Corollary 5.3.4. There is no closed term of type $\text{true} \equiv_{\text{Bool}} \text{false}$.

Proof. By theorem 5.3.3, any model \mathcal{C} of DTT is a valid model of MTT: send each mode to \mathcal{C} , and each modality to the identity. Therefore, a closed term of type $\text{true} \equiv_{\text{Bool}} \text{false}$ in MTT would also be a term of the same type in DTT. We may therefore reduce the consistency of MTT to that of a model of DTT, and in particular the set-theoretic one. \square

5.3.5.a Canonicity

We can now use MTT models to prove *canonicity* via gluing. Canonicity is an important metatheoretic result: it establishes the computational adequacy of MTT by ensuring that every *closed* term already is in or is equal to a *canonical form* – a value. Canonicity is traditionally established through a *logical relation* [Tai67; Mar75]. However, this method becomes very complicated when we have universes, as their presence makes the definition by induction on types impossible. It is instead necessary to construct a (large) relation on types, which associates a pair of types with a PER; the logical relation on terms is then subordinated to this relation on types [All87; Ang19]. This technique requires significant effort, and involves many proofs by simultaneous induction.

This approach can be simplified by replacing proof-*irrelevant* logical relations by a proof-relevant gluing construction [MS92]. This leads to the construction of a model in which (a) types are paired with proof-relevant predicates and (b) terms are equivalence classes of syntactic terms, along with a (type-determined) proof of their canonicity. The proof-relevance is crucial in the case of the universe,

⁶While Birkedal et al. [Bir+20] only consider endofunctors, there is no obstacle to extending the definition of a DRA to different categories, see section 5.2.

which contains not just the canonicity data for $A : \mathcal{U}$ but also the predicate for $\text{El } A$.

In order to simplify the construction of the glued model, we add an additional definitional equality to MTT, namely $(\cdot, \mathbf{a}_\mu) = () \text{ ctx}$. This equation is satisfied by all the concrete examples described in section 5.3.6.⁷ Semantically, it states that the functors $\llbracket \mathbf{a}_\mu \rrbracket$ strictly preserve the chosen terminal objects. Without this assumption we would have to establish canonicity not just for terms in the empty context, but for terms in a *locked empty context*, i.e. of the form (\mathbf{a}_μ) . This would semantically correspond to gluing along the nerve of the inclusion of locked empty contexts into the categories of contexts. This situation is comparable to that of proving canonicity for cubical type theories, where it is necessary to consider terms with open dimension variables [AHH18; Hub19]. However, the attendant glued model is complex, so we restrict this discussion to this simpler and more common case.

The full details of the glued model can be found in the technical report. Once we construct it, the initiality of syntax [Car86; KKA19] provides a witness of canonicity for every term.

Theorem 5.3.5 (Canonicity). If we extend MTT⁸ with the judgemental equality $(\cdot, \mathbf{a}_\mu) = () \text{ ctx}$, then for every closed term $\cdot \vdash a : A @ p$, the following conditions hold:

- If $A = \text{Bool}$, then $\cdot \vdash a = \bar{b} : A @ p$ where $\bar{b} \in \{\text{true}, \text{false}\}$.
- If $A = (b_0 \equiv_B b_1)$ then $\cdot \vdash b_0 = b_1 : B @ p$ and $\cdot \vdash a = \text{refl}_{b_0} : b_0 \equiv_B b_1 @ p$.
- If $A = \langle \nu \mid^n B \rangle$ then there is a term $\cdot \vdash b : B @ o$ such that $\cdot \vdash a = \text{mod}_\mu^m b : \langle \nu \mid^n B \rangle @ p$.

5.3.6 Applying MTT

We will now show concretely how MTT can be used in specific modal situations by varying the mode theory. We will focus on two different examples: *guarded recursion* [Nak00; Clo+16; Biz+16], which captures productive recursive definitions through a combination of modalities, and *adjoint modalities* [Ree09; LS16; LSR17; Shu18; Zwa19], where two modalities form an adjunction internal to the type theory. In both cases we will show how to reconstruct examples from *op. cit.* in MTT. The case of guarded recursion is particularly noteworthy, as the specialization of MTT to the appropriate mode theory leads to a new syntax which is considerably simpler than previous work.

⁷But not by the model of the mode theory used in chapter 7.

⁸without special equality rules such as PI:FUNEXT and ID:UIP.

5.3.6.a Guarded Recursion

The key idea of guarded recursion [Nak00] is to use the *later modality* (\blacktriangleright) to mark data which may only be used after some progress has been made, thereby enforcing productivity at the level of types. Concretely, the later modality is equipped with three basic operations:

$$\begin{aligned} \text{next} &: A \rightarrow \blacktriangleright A \\ \otimes &: \blacktriangleright(A \rightarrow B) \rightarrow \blacktriangleright A \rightarrow \blacktriangleright B \\ \text{l\"ob} &: (\blacktriangleright A \rightarrow A) \rightarrow A \end{aligned}$$

The first two operators make \blacktriangleright into an applicative functor [MP08] while the third, which is known as Löb induction, encodes guarded recursion: it enables us to define a term recursively, provided the recursion is provably productive.

The perennial example is, of course, the guarded stream type $\text{Str}_A \cong A \times \blacktriangleright \text{Str}_A$. This recursive type requires that the head of the stream is immediately available, but the tail may only be accessed after some productive work has taken place. This allows us to e.g. construct an infinite stream of ones:

$$\text{inf_stream_of_ones} := \text{l\"ob}(\lambda s. \text{cons}(1, s))$$

However, Str_A does not behave like a coinductive type: we may only define *causal* operations on streams, which excludes e.g. *tail*. In order to regain coinductive behaviour, Clouston et al. [Clo+16] introduced a second modality, \square ('always'), an idempotent comonad for which

$$\square \blacktriangleright A \simeq \square A. \quad (5.8)$$

Combining this modality with \blacktriangleright has proved rather tricky: previous work has used *delayed substitutions* [Biz+16], or has replaced \square with *clock quantification* [AM13; Møg14; BM15; BGM17]. The former poses serious implementation issues, and – while more flexible – the latter does not enjoy the conceptual simplicity of a single modality. In contrast, MTT enables us to effortlessly combine the two modalities and satisfy eq. (5.8).

To encode guarded recursion inside MTT, we must

1. choose a mode theory which induces an applicative functor \blacktriangleright and a comonad \square satisfying eq. (5.8),
2. construct the *intended model* of MTT with this mode theory, i.e. a model where these modalities are interpreted in the standard way [Bir+12], and
3. include Löb induction as an axiom.

To begin, we define $\mathcal{M}_{\mathbf{g}}$ to be the mode theory generated by

$$\begin{array}{ccc}
 \ell \curvearrowright t & \begin{array}{c} \xrightarrow{\gamma} \\ \xleftarrow{\delta} \end{array} & s \\
 & & \delta \circ \gamma \leq 1 \quad 1 = \gamma \circ \delta \\
 & & 1 \leq \ell \quad \gamma = \gamma \circ \ell
 \end{array} \tag{5.9}$$

We require that $\mathcal{M}_{\mathbf{g}}$ is poset-enriched, i.e. that there is at most one 2-cell between a pair of modalities, (μ, ν) , which we denote $\mu \leq \nu$ when it exists. As $\mathcal{M}_{\mathbf{g}}$ is not a full 2-category, we do not need to state any coherence equations between 2-cells.

Unlike prior guarded type theories, $\mathcal{M}_{\mathbf{g}}$ has *two modes*. We will think of elements of s as being *constant types and terms*, while types in t may *vary over time*. The reason for enforcing this division will become apparent in theorem 5.3.8, but for now observe that we can construct an idempotent comonad $b := \delta \circ \gamma$.

Lemma 5.3.6. $\langle b \mid \sqcup \rangle$ is an idempotent comonad and $\langle \ell \mid \sqcup \rangle$ is an applicative functor.

Proof. Follows from the combinators in section 5.3.3. \square

Next, eq. (5.8), which was hard to force in previous type theories, is provable: as $\gamma \circ \ell = \gamma$, the combinator $\text{comp}_{b,\ell}$ from section 5.3.3.a has the appropriate type:

$$\text{comp}_{b,\ell} : \langle b \mid^b \langle \ell \mid^t A \rangle \rangle \simeq \langle b \circ \ell \mid^{b \circ \ell} A \rangle = \langle b \mid^{b \circ \ell} A \rangle.$$

In order to construct the intended model, recall that the standard interpretation of guarded type theory uses the *topos of trees*, $\text{Psh}(\omega)$: see Birkedal et al. [Bir+12] for a thorough discussion. Crucially, it is easy to see that $\square = \Delta \circ \sqcup$, where

$$\begin{array}{ll}
 \sqcup : \text{Psh}(\omega) \rightarrow \text{Set} & \Delta : \text{Set} \rightarrow \text{Psh}(\omega) \\
 \sqcup \Gamma := \text{Hom}(\top, \Gamma) & (i \Rightarrow \Delta S) := S
 \end{array}$$

As both Set and $\text{Psh}(\omega)$ are models of DDT (see chapter 4), we may use theorem 5.3.3 to construct the intended model.

Theorem 5.3.7. There exists a model of MTT with mode theory $\mathcal{M}_{\mathbf{g}}$ where $\langle b \mid \sqcup \rangle$ is interpreted as \square and $\langle \ell \mid \sqcup \rangle$ as \blacktriangleright .

Proof. We choose the 2-functor which sends $s \mapsto \text{Set}$ and $t \mapsto \text{Psh}(\omega)$. Moreover, we define $\llbracket \blacktriangleleft_{\ell} \rrbracket$, $\llbracket \blacktriangleleft_{\delta} \rrbracket$, and $\llbracket \blacktriangleleft_{\gamma} \rrbracket$ to be the left adjoints of \blacktriangleright , Δ , and \sqcup respectively [Bir+20]. \square

From this point onwards we will write $\blacktriangleright^{\mathfrak{l}} := \langle \ell \mid^{\mathfrak{l}} \sqcup \rangle$ (making \mathfrak{l} truly a tick in the sense of Bahr et al. [BGM17]), $\Delta^{\mathfrak{v}} := \langle \delta \mid^{\mathfrak{v}} \sqcup \rangle$, and $\square^{\mathfrak{b}} := \langle b \mid^{\mathfrak{b}} \sqcup \rangle$.

The only thing that remains is to add Löb induction. This is a *modality-specific* operation that cannot be expressed in the mode theory, so we must add it as an axiom:

$$\begin{array}{l}
 \text{LOEB} \\
 \Gamma \vdash A \text{ type } @ t \\
 \Gamma, \ell \mid x :^{\mathfrak{l}} A[(1 \leq \ell) \downarrow_{\mathfrak{l}}^*] \vdash a : A @ t \\
 \hline
 \Gamma \vdash \text{lob}(x.a) : A @ t \\
 \text{where } \text{lob}(x.a) = a[\text{lob}(x.a)[(1 \leq \ell) \downarrow_{\mathfrak{l}}^*] / x \downarrow_{\mathfrak{l}}] \quad (\text{LOEB:BETA})
 \end{array}$$

Unfortunately, any axiom disrupts the metatheory of MTT so canonicity no longer applies. However, adding it to the type theory is sound, as the model supports it. At this point we may as well assume *equality reflection* (ID:REFLECTION) [Jac99], as is commonplace in previous guarded type theories [Biz+16]. This is stronger than necessary (function extensionality would suffice), but it simplifies proofs and makes comparison to previous work more direct.

Programming with Guarded MTT We can now use MTT to program with and reason about guarded recursion. For instance, we can define coinductive streams:

$$\begin{array}{l}
 \text{Str} : \mathbf{U} \rightarrow \mathbf{U} @ s \\
 \text{Str}(A) := \sqcup^{\mathfrak{u}}(\text{lob}(S.\Delta^{\mathfrak{v}}(A[\downarrow_{\mathfrak{u}\mathfrak{v}}^*]) \times \blacktriangleright^{\mathfrak{l}}(S(1 \leq \ell) \downarrow_{\mathfrak{l}})))
 \end{array}$$

Unlike prior guarded type theories, we have defined this stream operator not in mode t , which represents $\text{Psh}(\omega)$, but in mode s , which represents Set . Accordingly, this definition does not use \square . It first uses Δ to convert A to a t -type, and then \sqcup to move the recursive definition back to s . This alleviates some bookkeeping: in previous work [Biz+16] the stream type was actually coinductive only if A was a constant type (i.e. $A \simeq \square A$). Accordingly, theorems about streams had to pass around proofs that the elements of the stream are constant. In our case, defining Str at mode s ensures that the elements of the stream are automatically constant. Hence, $\text{Str}(A)$ is equivalent to the familiar definition, but it is no longer necessary to carry through proofs of constancy. Therefore, for *any* $A : \mathbf{U} @ s$ we have

Theorem 5.3.8. $\text{Str}(A)$ is the final coalgebra for $S \mapsto A \times S$ in mode s .

We can also program with $\text{Str}(A)$ by more directly appealing to the underlying guarded structure. For instance, we can define a ‘zip with’ function. Let

$\text{Str}'(\circlearrowleft A) = \text{l\"ob}(S.\Delta^\circlearrowleft(A) \times \blacktriangleright^{\downarrow}(S(1 \leq \ell)\downarrow_i))$ and write zs_h and zs_t for $\text{fst } zs$ and $\text{snd } zs$ respectively:

$$\begin{aligned} \text{zipWith}' &: \Delta^\circlearrowleft(A \rightarrow B \rightarrow C) \rightarrow \text{Str}'(\circlearrowleft A) \rightarrow \text{Str}'(\circlearrowleft B) \rightarrow \text{Str}'(\circlearrowleft C) \\ \text{zipWith}' &:= \lambda \hat{f}.\text{l\"ob}(r.\lambda xs.\lambda ys.(\hat{f} \otimes_\delta xs_h \otimes_\delta ys_h, \text{mod}_\ell^{\downarrow}(r \downarrow_i) \otimes_\ell xs_t \otimes_\ell ys_t)) \\ \text{zipWith} &: (A \rightarrow B \rightarrow C) \rightarrow \text{Str}(A) \rightarrow \text{Str}(B) \rightarrow \text{Str}(C) \\ \text{zipWith} &:= \lambda f.\lambda \widehat{xs}.\lambda \widehat{ys}.\text{mod}_\gamma^{\uparrow}(\text{zipWith}'(\text{mod}_\delta^\circlearrowleft f \downarrow_{\text{u}\circlearrowleft})) \otimes_\gamma \widehat{xs} \otimes_\gamma \widehat{ys} \end{aligned}$$

where \otimes_μ is defined in section 5.3.3.a.

We can also use dependent types to reason about guarded recursive programs. For example,

Theorem 5.3.9. If f is commutative then $\text{zipWith } f$ is commutative. That is, given $A, B : \mathbf{U}$ and $f : A \rightarrow A \rightarrow B$ there is a term of the following type:

$$\begin{aligned} &((x, y : A) \rightarrow f \ x \ y \equiv_B f \ y \ x) \\ &\rightarrow (xs, ys : \text{Str}(A)) \rightarrow \text{zipWith } f \ xs \ ys \equiv_{\text{Str}(B)} \text{zipWith } f \ ys \ xs \end{aligned}$$

All things considered, instantiating MTT with \mathcal{M}_g yields a highly expressive guarded dependent type theory with coinductive types. Unlike prior systems, e.g. Bahr et al.'s [BGM17], we do not need clock variables or syntactic checks of constancy. Moreover, the syntax is more robust than previous work that combines \square and \blacktriangleright [Clo+16; Biz+16], as there is no need for delayed substitutions. Unfortunately, the addition of the L\"ob axiom means theorem 5.3.5 cannot be directly applied, but the syntax remains sound and tractable.

5.3.6.b Internal Adjunctions

Up to this point we have only considered mode theories which are poset-enriched: there is at most one 2-cell between any pair of modalities. This has worked well for describing strict structures (section 5.3.3.b), as well as some specific settings (section 5.3.6.a). However, we would like to use MTT to reason about less strict categorical models. In this section we will show that we can readily use MTT to reason about a pair $\nu \dashv \mu$ of adjoint modalities.

Adjoint modalities are common in modal type theory, much in the same way that adjunctions are ubiquitous in mathematics [Ree09; LS16; LSR17; Lic+18; Shu18]. For example, the adjunction $\delta \dashv \gamma$ played an important role in the previous section. However, that particular case is unusually well-behaved, as it arises from a Galois connection. In contrast, the behavior of general adjoint modalities is much more subtle. We will show that by instantiating MTT with

a particular mode theory we can internally prove many properties of adjoint modalities that have previously been established only in special cases.

To begin, we pick the *walking adjunction* [SS86] for our mode theory, i.e. the 2-category \mathcal{M}_{adj} generated by:

$$\begin{array}{ccc}
 p & \begin{array}{c} \xrightarrow{\rho} \\ \xleftarrow{\kappa} \end{array} & q \\
 \eta : 1 \Rightarrow \rho \circ \kappa & 1_\rho = (1_\rho \star \varepsilon) \circ (\eta \star 1_\rho) & \\
 \varepsilon : \kappa \circ \rho \Rightarrow 1 & 1_\kappa = (\varepsilon \star 1_\kappa) \circ (1_\kappa \star \eta) & (5.10)
 \end{array}$$

This mode theory is the *classifying 2-category* for internal adjunctions: every 2-functor $\mathcal{M}_{\text{adj}}^{\text{coop}} \simeq \mathcal{M}_{\text{adj}} \rightarrow \text{Cat}$ determines a pair of adjoint functors, and vice versa. Consequently, substitutions $\Delta \rightarrow (\Gamma, \mathbf{\Delta}_\rho)$ are in bijection with substitutions $(\Delta, \mathbf{\Delta}_\kappa) \rightarrow \Gamma$. However, this is not enough on its own: we must also show that $\langle \kappa \mid \sqcup \rangle$ and $\langle \rho \mid \sqcup \rangle$ form an adjunction *inside* MTT.

Recovering the adjunction in MTT We can construct the unit and counit internally:

$$\begin{aligned}
 \text{unit} &: A \rightarrow \langle \rho \mid^\top \langle \kappa \mid^\natural A[\eta \downarrow_{\top \natural}] \rangle \rangle \\
 \text{unit} &:= \lambda x. \text{mod}_\rho^\top (\text{mod}_\kappa^\natural (x \eta \downarrow_{\top \natural})) \\
 \text{counit} &: \langle \kappa \mid^\natural \langle \rho \mid^\top B \rangle \rangle \rightarrow B[\varepsilon \downarrow_{\bullet \natural}^\top] \\
 \text{counit} &:= \lambda \hat{y}. \text{let} (\text{mod}_\kappa^\natural (\hat{y} \downarrow_{\top \natural}) = \hat{y}) \text{ in } \text{let}_\kappa^\natural (\text{mod}_\rho^\top (y \downarrow_{\top \natural}) = \hat{y} \downarrow_{\top \natural}) \text{ in } y \varepsilon \downarrow_{\bullet \natural}
 \end{aligned}$$

In order to account for dependence we must adjust the type by a 2-cell. For example, in the definition of `unit` we assume $\Gamma \vdash A \text{ type } @ q$, so $\langle \rho \mid^\top \langle \kappa \mid^\natural A \rangle \rangle$ is ill-typed. We can, however, obtain a version of A that is typable in the context $(\Gamma, \mathbf{\Delta}_\rho^\top, \mathbf{\Delta}_\kappa^\natural)$ by applying the substitution $\eta \downarrow_{\top \natural}^\circ$ to it, as in `CTX-MODEXT:VAR`.

We can prove that these two operations form an adjunction by showing they satisfy the triangle identities, e.g.

$$\begin{aligned}
 _ &: (\hat{x} : \langle \kappa \mid^\natural A \rangle) \rightarrow \hat{x} \equiv_{\langle \kappa \mid^\natural A \rangle} \text{counit}((\text{mod}_\kappa^\natural \text{unit}) \otimes_\kappa \hat{x}) \\
 _ &:= \lambda \hat{x}. \text{let} (\text{mod}_\kappa^\natural (x \downarrow_{\top \natural}) = \hat{x}) \text{ in } \text{refl}_{\text{mod}_\kappa^\natural x \downarrow_{\top \natural}}
 \end{aligned}$$

This proof relies on the fact that the modalities κ and ρ satisfy the triangle identities themselves in \mathcal{M}_{adj} .

The existence of the unit and counit is enough to internally determine an adjunction. We might want to use an alternative description, e.g. to manipulate a natural bijection of Hom-sets, $\text{Hom}(LA, B) \cong \text{Hom}(A, RB)$.

Unfortunately, this isomorphism cannot be recovered internally. First, notice that $\langle \kappa \mid A \rangle \rightarrow B$ and $A \rightarrow \langle \rho \mid B \rangle$ are types in different modes – p and q respectively – so $(\langle \kappa \mid A \rangle \rightarrow B) \simeq (A \rightarrow \langle \rho \mid B \rangle)$ is ill-typed. Second, even if $p = q$ so that κ and ρ are endomodalities and this equivalence is well-typed, an internal equivalence is a stronger condition than a bijection of hom-sets: it is equivalent to an isomorphism of exponential objects $B^L \simeq (RB)^A$.

Prior work [Lic+18] addressed this by introducing a third modality \square , such that terms of $\square A$ represent *global* elements of A , and then requiring transposition only for functions under \square . Global elements of B^A are in bijection with $\text{Hom}(A, B)$, so the postulated equivalence corresponds to the expected bijection. We can rephrase this argument in MTT. Suppose that $p = q$, and that $\text{Hom}(p, p)$ is equipped with an initial object, i.e. a modality $\perp : p \rightarrow p$ and a unique 2-cell $\llbracket \perp : \perp \Rightarrow \mu \rrbracket$ for all μ . Then,

Theorem 5.3.10. The following equivalence is definable in MTT:

$$\langle \perp \mid \langle \kappa \mid A[\llbracket \downarrow_{\perp}^{\perp} \rrbracket] \rangle \rightarrow B \rangle \simeq \langle \perp \mid A \rightarrow \langle \rho \mid B[\llbracket \downarrow_{\perp}^{\perp} \rrbracket] \rangle \rangle.$$

In fact, because modalities in MTT preserve finite products (a consequence of \otimes_{ν}), an alternative phrasing of transposition is possible.

Theorem 5.3.11. The following equivalence is definable in MTT:

$$\langle \rho \mid \langle \kappa \mid A[\eta \downarrow_{\perp}^{\circ}] \rangle \rightarrow B \rangle \simeq A \rightarrow \langle \rho \mid B \rangle.$$

Modal induction for the left adjoint Having internalized $\kappa \dashv \rho$, many of the classical results about adjunctions can be replayed inside MTT. For instance, by carrying out a proof that left adjoints preserve colimits internally to MTT, we recover *modal* or *crisp induction principles* for κ [LS16; Shu18]. We can then show e.g. that $\langle \kappa \mid \text{Bool} \rangle \simeq \text{Bool}$. However, in order to construct this equivalence it will be convenient to formulate a general induction principle for $\langle \kappa \mid \text{Bool} \rangle$.

Supposing that $\Gamma, \blacksquare_{\kappa}, \blacksquare_{\rho} \vdash T : \langle \kappa \mid \text{Bool} \rangle \rightarrow \mathbf{U} @ p$, we can define a term

$$\begin{aligned} \text{if}^{\kappa} : \langle \kappa \circ \rho \mid \text{true} \rangle &\rightarrow \langle \kappa \circ \rho \mid \text{false} \rangle \\ &\rightarrow (\hat{b} : \langle \kappa \mid \text{Bool} \rangle) \rightarrow C[\varepsilon \downarrow_{\circ}^{\perp}](\hat{b}). \end{aligned}$$

This is a version of the conditional that operates on $\langle \kappa \mid \text{Bool} \rangle$ rather than Bool . Using this modal induction principle, we can show

Theorem 5.3.12. $\langle \kappa \mid \text{Bool} \rangle \simeq \text{Bool}$

Similarly, we can prove that κ preserves identity types:

Theorem 5.3.13. $\langle \kappa \mid^{\mathbb{E}} a \equiv_A b \rangle \simeq (\text{mod}_{\kappa}^{\mathbb{E}} a \equiv_{\langle \kappa \mid^{\mathbb{E}} A \rangle} \text{mod}_{\kappa}^{\mathbb{E}} b)$.

This instantiation of MTT with \mathcal{M}_{adj} yields a systematic treatment of an internal transposition axiom [Lic+18], and is sufficiently expressive to derive crisp induction principles [Shu18]. In both cases we can use MTT instead of a handcrafted modal type theory. Moreover, as we have not added any new axioms to deal with internal adjunctions, our canonicity result applies.

5.3.6.c Further Examples

In addition to the examples described above, we have applied MTT to a wide variety of other situations, including

- parametricity, via degrees of relatedness [ND18a] (see chapter 9),
- synchronous and guarded programming with warps [Gua18],
- finer grained notions of realizability and local maps of categories of assemblies [Bir00].

While interesting, we cannot discuss the details of these applications here for want of space. We invite the interested reader to consult the accompanying technical report.

5.3.7 Related Work

MTT is related to many prior modal type theories. In particular, its formulation draws on three important techniques: split contexts, left division, and the Fitch style.

Split-context type theories [PD01; NPP08; dPR15; Kav17; Shu18; Pie+19; Zwa19] divide the context into different *zones*, one for each modality, which are then manipulated by modal connectives. This has proven to be an effective approach for a number of modalities, and sometimes even scales to full dependent type theories [dPR15; Shu18; Zwa19]. However, the structure of contexts becomes very complex as the number of modalities increases.

In order to manage this complexity, some modal type theories employ *left-division*: each variable declaration in the context is annotated with a modality, and a *left-division operation*, which is a left adjoint to post-composition of modalities, is used to state the introduction rules [Pfe01; Abe06; Abe08; AS12; NVD17a; ND18a]. Left-division calculi handle multiple modalities and support

full dependent types, but many important modal situations cannot be equipped with a left-division structure.

Another technique stipulates that modalities are essentially right adjoints, with the corresponding left adjoints being constructors on contexts. These *Fitch-style* type theories [BGM17; Clo18; Bir+20; GSB19a; BGM19] are relatively simple, which has made them convenient for programming applications [GSB19a; BGM19]. Nevertheless, scaling this approach to a multimodal setting has proven difficult. In particular, extending the elimination rule to a multimodal setting remains an open problem.

MTT synthesizes these approaches by including both Fitch-style locks and left-division-style annotations in its judgmental structure. The combination of these devices circumvents many previously encountered difficulties. For example, this combination obviates the need for a left division operation, as MTT uses a Fitch-style introduction rule.⁹ On the other hand, the left-division-style elimination rule of MTT smoothly accommodates multiple interacting modalities.

Most prior work on modal type theory has focused on incorporating a specific collection of modalities. The sole exception is the *LSR framework* of Licata, Shulman, and Riley [LSR17]. The LSR framework supports an arbitrary collection of substructural modalities over simple types, and there is ongoing work on a dependently-typed system. The price to pay for this expressivity is practicality: the modal connectives require *delayed substitutions* [BdP00; Biz+16], which complicate the equational theory, and make pen-and-paper calculations cumbersome. The relationship between the modalities of MTT and those of the LSR framework is not clear. The introduction rule WDRA:INTRO mirrors the introduction rule for U types. This is to be expected, as U types behave like right adjoints. On the other hand, the elimination rule in eq. (5.6) does not match the rule for U types, but instead is closer to the elimination rule for F types. In fact, this is a necessary compromise to avoid the introduction of delayed substitutions. In *op. cit.* the elimination rule for U types and the introduction rule for F types both require annotation with a substitution to bring the context into a specific shape. MTT avoids this by mixing these two styles of presentation.

5.3.8 Conclusion

We introduced and studied MTT, a dependent type theory parametrized by a mode theory that describes interacting modalities. We have demonstrated

⁹Should a left division operation be available, then the lock can be desugared to a left division.

that MTT may be used to reason about several important modal settings, and proven basic metatheorems about its syntax, including canonicity.

In the future we plan to further develop the metatheory of MTT. In addition to extending our canonicity result to remove the restriction that locks preserve the empty context, we hope to prove that MTT enjoys normalization, and hence that type-checking is decidable (provided the mode theory is). Both of these theorems can be proven by gluing arguments similar to that discussed in section 5.3.5.a by gluing along the appropriate nerves. The latter result would pave the way to a practical implementation of a multimodal proof assistant.

Presently MTT only supports modalities which behave like right adjoints. While this covers a wide class of examples, many modalities are instead left adjoints. We hope to extend MTT to allow left adjoints to act on types instead of merely contexts while retaining its practical syntax. Similarly, we also hope to extend our analysis to some class of *modality-specific* operations, e.g. Löb induction. These operations cannot be captured by a mode theory, and so can only be added *axiomatically* to MTT (as was done in section 5.3.6.a), thus invalidating some of our metatheorems. However, such operations play an important role in many applications, and should be accounted for in a systematic way.

Chapter 6

Presheaf Type Theory

Recall from chapter 4 that every presheaf category is a CwF. In this chapter, we consider what special things we can do in type theory *if* we model it in presheaf CwFs.

Section 6.3 discusses four type formers that can be modelled in any presheaf category:

- The **Glue-type**, which takes as input a type $\Gamma \vdash A$ type and a *partial* slice $f : T \rightarrow A$; partial meaning that the slice is only defined when a certain proposition φ holds. It then gives the final total (as opposed to partial) slice over A that extends (T, f) .

This type was introduced by Cohen, Coquand, Huber, and Mörtberg [Coh+17] in order to give computational content to the univalence axiom of HoTT. There, in order to guarantee Kan fibrancy (composition of paths) of the **Glue-type**, f was required to be an equivalence.

In ParamDTT [NVD17a], we used the **Glue-type** as an internal parametricity operator. As we did not need fibrancy, we removed the condition that f be an equivalence. In the technical report of ParamDTT [Nuy17], I think I was first to interpret the **Glue-type** in an arbitrary presheaf category, but this was a straightforward generalization.

We include Orton and A. M. Pitts’s implementation of the **Glue-type** by strictifying a pullback [OP18; Ort18].

- We introduced the **Weld-type** in ParamDTT [NVD17a], also as an internal parametricity operator. It is dual to **Glue** and gives the initial completion of a partial coslice $f : A \rightarrow T$ under A . We dualize Orton and A. M. Pitts’s

implementation of the `Glue`-type and implement `Weld` by strictifying a pushout.

- The `Strict`-type [OP18; Ort18] is used to perform said strictifications; it gives the essentially unique completion of a partial ‘isoslice’ $i : T \cong A$ over A .
- The pushout needed to implement `Weld`, is not definable and needs to be added explicitly as a (moderately novel) type former.

The four type formers above all take *partial* objects as input, which are only defined when a proposition φ holds. Thus, we need to extend DTT with propositions. Moreover, in an intensional setting, the idea is that `Glue`, `Weld` and `Strict` extend their inputs *definitionally*, which is only useful if it is at least sometimes decidable whether φ is true. In cubical HoTT, Cohen et al. [Coh+17] give a theory of decidable propositions (called *faces*) depending only on interval variables, thus defining a subpresheaf of an n -cube. This theory does not take proposition *variables* into account.

In **sections 6.1 and 6.2**, we give typing rules for dealing with propositions in DTT modelled in an arbitrary presheaf category, and we do consider proposition variables, as we believe that this is very desirable in practice.¹ These sections do not sparkle with novelty, but we do generalize (and in the latter case formalize) the treatments in cubical HoTT [Coh+17] and ParamDTT [NVD17a] and bring some structure by translating less straightforward intensional concepts to more straightforward extensional concepts before interpreting those in the presheaf model. We also sketch a type-checking algorithm, arguing that indeed it is often possible to reduce true propositions to \top .

In **section 6.4** on presheaf modalities, we prove that certain functors involving presheaf categories are automatically strict or weak CwF morphisms.

In **section 6.5** on presheaf MTT, we discuss what happens when we want to use the content of sections 6.1 to 6.3 in MTT.

Those who have consulted the technical reports on ParamDTT [Nuy17] and RelDTT [Nuy18a], may expect that this chapter would also contain theorems asserting that certain presheaf modalities preserve certain type formers, which were used there as a lemma in order to model modal induction principles. However, these preservation theorems often only hold for internally left adjoint modalities, and then we argue in the MTT technical report [Gra+20a] that they can be proven internally.

¹Cubical Agda [VMA19] has proposition variables by reusing the interval as the type of propositions. The connections \vee and \wedge and the endpoints 0 and 1 serve as logical operators, the identity serves as $\sqcup \doteq_{\mathbb{I}} 1$ and the negation as $\sqcup \doteq_{\mathbb{I}} 0$.

For readers familiar with the presheaf model of DTT, we recommend a look at notation 2.3.2 and a quick skim of chapter 4 to acquaint oneself with the peculiar notations used in this thesis. Readers unfamiliar with the presheaf model, can either omit the proofs on semantics in the current chapter or learn about it in chapter 4 or of course in the original work [Hof97; HS97].

6.1 Propositions: Syntax and Presheaf Semantics

In this section, we extend DTT with a judgement form and typing rules for working with propositions. Semantically, these are essentially types whose elements are all equal. Such types are also considered internally in many type theories, including HoTT [Uni13], and proof assistants, including Coq [Coq; Coq14] and recently also Agda [Nor09].

However, syntactically, these propositions will not serve quite the same purpose as in the aforementioned systems. Rather, they will take the role of faces in cubical homotopy type theory [Coh+17]. The idea is that these propositions will actually reduce to \top (logical truth) when they are provable. This allows us to consider terms and types extending other terms and types. For example, if a type A exists when the proposition φ holds, then we can consider a type B_φ such that B_\top reduces to A . This is then effectively a type that extends A . Note that this application does not require that propositions reduce to \perp when they are not provable.

We give an intensional and an extensional variant of the typing rules for propositions (marking inference rules with I resp. E when they belong to only one variant), each meant to extend DTT with the corresponding identity type, and we will gradually prove the following results:

Theorem 6.1.1. All typing rules of the extensional variant are sound in any presheaf CwF over a small base category.

Theorem 6.1.2. All judgement forms of the intensional variant are definable by translation to other judgements in the extensional variant. After doing so (in an appropriate way), all typing rules of the intensional variant are derivable in the extensional variant, and therefore also sound in any presheaf CwF.

Note that we no longer aim for completeness, now that we have moved to presheaf CwFs.

The extensional version will not be so different from a typical treatment of propositions as in Coq; the main unusual characteristic will be that logical operators have computation rules, e.g. $\varphi \wedge \top = \varphi$. In the intensional

version, however, we will need to avoid ending up with *neutral* propositions (i.e. propositions whose computation is stuck on a variable). Having a proposition that is *simply* a variable is ok, as we can simply decree that the variable equals \top when the proposition is assumed to be true in the context. However, we cannot have propositions of the form $f a$ where f is a variable, because it is problematic from an algorithmic viewpoint to equate $f a$ to \top .

The treatment of propositions for this purpose is different in almost all accounts [see e.g. Coh+17; VMA19; NVD17a]. To our knowledge, none of these handles proposition *variables* in a principled way, and all of these are designed with a specific base category in mind. In this sense, the existing approaches do not meet our needs. For our purposes in the remainder of this thesis, we needed *some* fixed treatment, and we decided to try and give an intensional treatment that is maximally ready to be implemented, leading to a treatment that is probably novel in the sense that no prior treatment is identical. However, we do not consider the implementability of our typing rules to be a core contribution of this thesis and do not attempt to solidify it in a formal theorem.

The main purpose of the extensional treatment is that we can simplify the semantics of the intensional system by factoring them through the extensional one. Similar factorizations, using some form of extensional type theory as the internal language of a topos, have been used by Birkedal et al. [Bir+19] for guarded cubical HoTT and later² by Orton and A. M. Pitts [OP18] for cubical HoTT.

6.1.1 Structural Rules

In fig. 6.1, we extend DTT with proposition and assertion judgements. The proposition judgement $(\Gamma \vdash \varphi \text{ prop})$ states that φ is a proposition, and the assertion judgement $(\Gamma \vdash \varphi)$ states that φ holds. This means that an assertion judgement by itself does not classify a GAT term, which is against the definition of a GAT. However, this can again be solved by a simple trick: we could have a judgement $\Gamma \vdash p : \varphi$ which expresses that p is proof of φ , and then have a typing rule that equates all proofs of the same proposition.

Propositions can be substituted and when a proposition holds, this fact is preserved under substitution.

A context can be restricted to the situation where φ holds (one could think of this as adding a variable $p : \varphi$), and this behaves quite similar to weakening. An object (type, term, proposition, ...) which has a proposition in the context, is called **partial**; the antonym of partial is **total**.

²in terms of first arXiv preprint

Prerequisites: DTT (fig. 3.2).

Judgement forms:

$$\begin{array}{c}
 \text{JUD:PROP} \\
 \frac{\Gamma \text{ ctx}}{(\Gamma \vdash \varphi \text{ prop}) \text{ jud}} \\
 \\
 \text{JUD:ASSERT} \\
 \frac{\Gamma \vdash \varphi \text{ prop}}{(\Gamma \vdash \varphi) \text{ jud}} \\
 \\
 \text{PROP:SUB} \\
 \frac{\Gamma \vdash \varphi \text{ prop} \quad \sigma : \Delta \rightarrow \Gamma}{\Delta \vdash \varphi[\sigma] \text{ prop}} \\
 \text{where } \varphi[\text{id}] = \varphi \quad \varphi[\sigma \circ \tau] = \varphi[\sigma][\tau] \\
 \\
 \text{ASSERT:SUB} \\
 \frac{\Gamma \vdash \varphi \quad \sigma : \Delta \rightarrow \Gamma}{\Delta \vdash \varphi[\sigma]}
 \end{array}$$

Context restriction:

$$\begin{array}{c}
 \text{CTX-RESTR} \\
 \frac{\Gamma \text{ ctx} \quad \Gamma \vdash \varphi \text{ prop}}{(\Gamma, \varphi) \text{ ctx}} \\
 \\
 \text{CTX-RESTR:INTRO} \\
 \frac{\sigma : \Delta \rightarrow \Gamma \quad \Delta \vdash \varphi[\sigma]}{(\sigma, \text{ok}) : \Delta \rightarrow (\Gamma, \varphi)} \\
 \text{where } (\sigma, \text{ok}) \circ \rho = (\sigma \circ \rho, \text{ok}) \\
 \tau = (\pi \circ \tau, \text{ok}) \quad (\text{CTX-RESTR:ETA}) \\
 \\
 \text{CTX-RESTR:WKN} \\
 \frac{\Gamma \text{ ctx} \quad \Gamma \vdash \varphi \text{ prop}}{\pi : (\Gamma, \varphi) \rightarrow \Gamma} \\
 \text{where } \pi \circ (\sigma, \text{ok}) = \sigma \quad (\text{CTX-RESTR:WKN:BETA}) \\
 \\
 \text{CTX-RESTR:VAR} \\
 \frac{\Gamma \text{ ctx} \quad \Gamma \vdash \varphi \text{ prop}}{\Gamma, \varphi \vdash \varphi}
 \end{array}$$

Figure 6.1: Structural rules for propositions.

Proof of theorem 6.1.1. Semantically, a proposition $\Gamma \vdash \varphi \text{ prop}$ is a subpresheaf $\Gamma.\varphi$ of Γ . Substitution is defined by pullback (preimage under σ). The judgement $\Gamma \vdash \varphi$ is interpreted by $\Gamma.\varphi = \Gamma$. The restricted context is of course interpreted by $\Gamma.\varphi$, making the other operations straightforward to model. \square

Notation 6.1.3. For $\gamma \in (W \Rightarrow \Gamma)$, we write $W \triangleright \varphi[\gamma]$ for the statement $\gamma \in (W \Rightarrow \Gamma.\varphi)$, mimicking the assertion judgement.

6.1.2 Universe of Propositions

In the extensional system, just like there is a universe U_ℓ classifying types of level ℓ (fig. 3.10), we will have a universe **Prop** classifying propositions. We will

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1).

Extensional variant:

$$\begin{array}{c}
 \text{PROPUNI} \\
 \frac{\Gamma \text{ ctx}}{\Gamma \vdash \text{Prop type}_0} \text{E}
 \end{array}
 \qquad
 \begin{array}{c}
 \text{PROPUNI:INTRO} \\
 \frac{\Gamma \vdash \varphi \text{ prop}}{\Gamma \vdash \ulcorner \varphi \urcorner : \text{Prop}} \text{E} \\
 \text{where } \theta = \ulcorner \text{El } \theta \urcorner \\
 (\text{PROPUNI:ETA})
 \end{array}
 \qquad
 \begin{array}{c}
 \text{PROPUNI:ELIM} \\
 \frac{\Gamma \vdash \theta : \text{Prop}}{\Gamma \vdash \text{El } \theta \text{ prop}} \text{E} \\
 \text{where } \text{El } \ulcorner \varphi \urcorner = \varphi \\
 (\text{PROPUNI:BETA})
 \end{array}$$

Intensional variant:

Well-behaved propositions:

$$\begin{array}{c}
 \frac{\Gamma \text{ ctx}}{(\Gamma \vdash \theta :: \text{Prop}) \text{ jud}} \text{I}
 \end{array}
 \qquad
 \begin{array}{c}
 \frac{\Gamma \vdash \varphi \text{ prop}}{\Gamma \vdash \ulcorner \varphi \urcorner :: \text{Prop}} \text{I} \\
 \text{where } \ulcorner \text{El } \theta \urcorner = \theta
 \end{array}
 \qquad
 \begin{array}{c}
 \frac{\Gamma \vdash \theta :: \text{Prop}}{\Gamma \vdash \text{El } \theta \text{ prop}} \text{I} \\
 \text{where } \text{El } \ulcorner \varphi \urcorner = \varphi
 \end{array}$$

$$\frac{\Gamma \vdash \theta :: \text{Prop} \quad \sigma : \Delta \rightarrow \Gamma}{\Delta \vdash \theta[\sigma] :: \text{Prop}}$$

where $\theta[\text{id}] = \theta$ $\theta[\sigma \circ \tau] = \theta[\sigma][\tau]$

Context extension with a proposition:

$$\frac{\Gamma \text{ ctx}}{(\Gamma, \chi :: \text{Prop}) \text{ ctx}} \quad (\text{other rules analogous to fig. 3.2})$$

Π -types over Prop:

$$\frac{\Gamma, \chi :: \text{Prop} \vdash T \text{ type}_\ell}{\Gamma \vdash (\chi :: \text{Prop}) \rightarrow T \text{ type}_\ell} \quad (\text{other rules analogous to fig. 3.3})$$

Figure 6.2: Typing rules for the universe of propositions.

generally omit the encoding and decoding operations.

In the intensional system, we want to avoid obtaining propositions that are eliminations stuck on a variable. Therefore, following Cohen et al.'s treatment of interval variables [Coh+17], we perform a hack: we make **Prop** only a **pseudotype**: it does not appear in a type judgement (so that we cannot apply arbitrary type formers to it) but instead inhabiting **Prop** is a judgement form in itself. We do allow **Prop** to appear in the domain of a Π -type. So functions can take propositions as input, but not as output. Similarly, we cannot have $\text{Prop} \times B$, so we avoid propositions of the form $\text{fst } z$ where z is a variable.

Proof of theorem 6.1.1. In order to define the semantics of \mathbf{Prop} , we need to define $(W \triangleright \mathbf{Prop}[\gamma])$. By lemma 3.2.4, this is isomorphic to $(\mathbf{y}W \vdash \mathbf{Prop}[\gamma]) = (\mathbf{y}W \vdash \mathbf{Prop})$, which by the typing rules is clearly isomorphic to $(\mathbf{y}W \vdash \mathbf{prop})$, i.e. the set of subpresheaves of $\mathbf{y}W$. Unsurprisingly, this reveals that $\mathbf{Prop} \in \mathbf{Ty}_0((\))$ is actually the type counterpart (in the sense of definition 3.2.8) of the subobject classifier $\mathbf{Prop} \in \mathbf{Psh}(\mathcal{W})$ (section 2.3.6). We then interpret \mathbf{El} and $\lceil _ \rceil$ as the encoding and decoding of subpresheaves as/from morphisms to the subobject classifier. \square

Proof of theorem 6.1.2. We define $\Gamma \vdash \theta :: \mathbf{Prop}$ simply as $\Gamma \vdash \theta : \mathbf{Prop}$. Then all other rules follow. \square

6.1.3 Logical Operators

The typing rules discussed in this section are listed in fig. 6.3.

Logical truth Logical truth always holds. In the extensional system, we postulate that any proposition that holds in a context, is in that context equal to \top . In the intensional system, we will prove this by induction, and we need only postulate it in the base case.

Proposition 6.1.4. In the intensional system, every proposition that holds, is equal to \top .

Logical conjunction The logical conjunction computes to one operand if the other is \top . We need no introduction rules, because if φ and χ both hold, then they are both equal to \top , so that $\varphi \wedge \chi = \top \wedge \top = \top$. We do need elimination rules.

Logical falsehood Logical falsehood of course has no introduction rule. It is eliminated in the same way as **Empty** (fig. 3.7). However, the motive T gets no assumption that \perp holds. Indeed, such an assumption is useless: since $\Gamma \vdash \perp$, we know that $\perp = \top$, and it is useless to assume \top .

We equip logical falsehood with an η -rule which not only applies to terms but to inhabitants of arbitrary judgement forms. This effectively asserts that everything is equal as soon as \perp holds.

In the extensional system, the η -rule does not really add anything. Indeed, there, thanks to the existence of universes and the non-existence of a special

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1).

Logical truth:

$$\frac{\text{TRUTH}}{\Gamma \text{ ctx}} \quad \frac{\text{TRUTH:INTRO}}{\Gamma \vdash \text{ ctx}}$$

$$\frac{}{\Gamma \vdash \top \text{ prop}} \quad \frac{}{\Gamma \vdash \perp}$$

Uniqueness of truth:

E: Always holds. I: Holds as a rule for variables; then admissible for all propositions (proposition 6.1.4).

$$\frac{\text{UT}}{\Gamma \vdash \varphi}$$

$$\frac{}{\Gamma \vdash \varphi = \top \text{ prop}}$$

Logical conjunction:

$$\frac{\text{AND}}{\Gamma \vdash \varphi, \chi \text{ prop}} \quad \frac{\text{AND:ELIM:FST}}{\Gamma \vdash \varphi \wedge \chi} \quad \frac{\text{AND:ELIM:SND}}{\Gamma \vdash \varphi \wedge \chi}$$

$$\frac{}{\Gamma \vdash \varphi \wedge \chi \text{ prop}} \quad \frac{}{\Gamma \vdash \varphi}$$

where $\varphi \wedge \top = \varphi$ (AND:RED:FST) $\quad \Gamma \vdash \chi$

$\top \wedge \chi = \chi$ (AND:RED:SND)

Logical falsehood:

$$\frac{\text{FALSEHOOD}}{\Gamma \text{ ctx}} \quad \frac{\text{FALSEHOOD:ELIM}}{\Gamma \vdash T \text{ type}} \quad \frac{}{\Gamma \vdash \perp}$$

$$\frac{}{\Gamma \vdash \perp \text{ prop}} \quad \frac{\text{FALSEHOOD:ETA}}{\Gamma \vdash j_1, j_2 : J} \quad \frac{}{\Gamma \vdash \perp}$$

$$\frac{}{\Gamma \vdash j_1 = j_2 : J}$$

Logical disjunction:

$$\frac{\text{OR}}{\Gamma \vdash \varphi, \chi \text{ prop}} \quad \frac{}{\Gamma \vdash \varphi \vee \chi \text{ prop}}$$

where $\varphi \vee \top = \top$ (OR:RED:INL) $\quad \Gamma \vdash \chi$

$\top \vee \chi = \top$ (OR:RED:INR)

$$\frac{\text{OR:ELIM}}{\Gamma, \varphi \vdash t_{\text{inl}} : T \quad \Gamma \vdash T \text{ type}} \quad \frac{\text{OR:ETA}}{\Gamma \vdash j_1, j_2 : J}$$

$$\frac{}{\Gamma, \chi \vdash t_{\text{inr}} : T \quad \Gamma \vdash \varphi \vee \chi} \quad \frac{}{\Gamma, \varphi \vdash j_1 = j_2 : J}$$

$$\frac{}{\Gamma, \varphi \wedge \chi \vdash t_{\text{inl}} = t_{\text{inr}} : T} \quad \frac{}{\Gamma, \chi \vdash j_1 = j_2 : J}$$

$$\frac{}{\Gamma \vdash \{\varphi ? t_{\text{inl}} \mid \chi ? t_{\text{inr}}\} : T} \quad \frac{}{\Gamma \vdash \varphi \vee \chi}$$

where $\{\top ? t_{\text{inl}} \mid \chi ? t_{\text{inr}}\} = t_{\text{inl}}$ (OR:INL:BETA) $\quad \Gamma \vdash j_1 = j_2 : J$

$\{\varphi ? t_{\text{inl}} \mid \top ? t_{\text{inr}}\} = t_{\text{inr}}$ (OR:INR:BETA)

Figure 6.3: Typing rules for logical operators and proof elimination.

Prop judgement, every judgemental equality is equivalent to a judgemental equality for terms. And the η -rule for terms holds judgementally since it is provable propositionally.

Logical disjunction Logical disjunction computes to \top if one operand is \top . Hence, again, we need no introduction rules. We eliminate by pattern matching. The eliminator is similar to that of the coproduct type (fig. 3.6), but requires that the clauses for both patterns match as we do not distinguish between evidence of $\varphi \vee \chi$ that proves the left vs. the right operand. Again, it is superfluous to let the motive assume $\varphi \vee \chi$. Again, the η -rule holds for all judgement forms.

Fulfilling our obligations

Proof of theorem 6.1.1. \top We (must) interpret \top as the full subpresheaf $\Gamma.\top := \Gamma \subseteq \Gamma$. Clearly then, \top holds. Uniqueness of proofs is semantically obvious because both judgements mean the same thing.

\wedge We define $\Gamma.(\varphi \wedge \chi) := \Gamma.\varphi \cap \Gamma.\chi$. This clearly satisfies the computation rules for \top , as well as the elimination rules for \wedge .

\perp We define $\Gamma.\perp$ as the empty presheaf. Then $\Gamma \vdash \perp$ means that Γ is empty so we can soundly say whatever we want in context Γ .

\vee We define $\Gamma.(\varphi \vee \chi) := \Gamma.\varphi \cup \Gamma.\chi$. This clearly satisfies the computation rules for \top .

For the eliminator, call the resulting term t . The premise $\Gamma \vdash \varphi \vee \chi$ means that $\Gamma = \Gamma.\varphi \cup \Gamma.\chi$. Then every cell $\gamma : W \Rightarrow \Gamma$ is a cell of either $\Gamma.\varphi$ or $\Gamma.\chi$. In the former case, we define $t[\gamma]$ as $t_{\text{inl}}[\gamma, \text{tt}]$; in the latter as $t_{\text{inr}}[\gamma, \text{tt}]$. In the overlap of both cases, we have ensured that both definitions match, so that the resulting term automatically respects restriction. \square

Proof of theorem 6.1.2. We did not add any intensional-specific typing rules. \square

Proof of proposition 6.1.4. We prove that any proposition $\Gamma \vdash \varphi \text{ prop}$ which holds ($\Gamma \vdash \varphi$ is true ($\Gamma \vdash \varphi = \top \text{ prop}$)).

\top Trivial.

\wedge Assume $\varphi \wedge \chi$ holds. Then φ and χ hold, so $\varphi \wedge \chi = \top \wedge \top = \top$.

\perp Assume \perp holds. Then by the η -rule for \perp -elimination, everything is equal.

\vee Assume $\varphi \vee \chi$ holds. Then by the η -rule for \vee -elimination, we only need to prove $\varphi \vee \chi = \top$ assuming that either φ or χ holds. But by the induction hypothesis, this operand is then equal to \top , so that $\varphi \vee \chi = \top$. \square

6.1.4 Equality Proposition

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1), logical truth (fig. 6.3).

$$\begin{array}{c}
 \text{EQ} \\
 \Gamma \vdash A \text{ type}_\ell \\
 \Gamma \vdash a, b : A \\
 \hline
 \Gamma \vdash a \doteq_A b \text{ prop} \\
 \text{where } (a \doteq_A a) = \top \quad (\text{EQ:RED})
 \end{array}
 \qquad
 \begin{array}{c}
 \text{EQ:REFLECTION} \\
 \Gamma \vdash a \doteq_A b \\
 \hline
 \Gamma \vdash a = b : A \quad \text{E}
 \end{array}$$

$$\begin{array}{c}
 \text{EQ:J} \\
 \Gamma \vdash a : A \\
 \Gamma, y : A, a \doteq_A y \vdash T \text{ type} \quad \Gamma \vdash b : A \\
 \Gamma \vdash t_{\text{refl}} : T[a/y, \text{refl}_a/w] \quad \Gamma \vdash a \doteq_A b \\
 \hline
 \Gamma \vdash \{b := a ? t_{\text{refl}}\} : T[b/y] \\
 \text{where } \{a := a ? t_{\text{refl}}\} = t_{\text{refl}} \quad (\text{EQ:BETA}) \\
 \text{To be supplemented later with an } \eta\text{-rule.}
 \end{array}
 \quad \text{I}$$

Figure 6.4: Typing rules for the equality proposition.

We introduce an equality proposition which computes to \top if both hands are judgementally equal. Hence, we do not need an introduction rule for reflexivity.

The eliminators are essentially the same as for the intensional/extensional identity type (fig. 3.8). However, unlike the eliminators for falsehood and disjunction, the use of the eliminator (J-rule) for the intensional identity proposition requires a choice of motive T . Hence, we cannot simply postulate an η -rule for arbitrary judgement forms as we did earlier, and therefore we cannot yet prove proposition 6.1.4.

The computation rules EQ:RED and EQ:BETA are impractical from an implementation perspective. Although we aim for judgemental equality to be decidable, it is still a judgement and deciding its derivability can be a fair amount of work. Moreover, in a language with implicit arguments, its

derivability can depend on the value of certain metavariables. It is therefore inconvenient if the rewriting algorithm has to wait for such judgement to be decided.

We will only resolve the aforementioned issues when we have more knowledge of the specific application at hand. Typically, we will only allow the use of the equality proposition for a specific pseudotype (e.g. the interval \mathbb{I} in cubical type theory) which, like **Prop**, can only appear in the domain of a Π -type and nowhere else in type formers, in order to avoid having to deal with terms that are stuck on eliminating a variable.

We do not postulate function extensionality for the identity proposition. One reason is that it follows from function extensionality for the identity type (using proof types, see section 6.2). More importantly, we do not intend to use the identity proposition on types as advanced as the Π -type; rather, we intend to reserve it for types such as the interval \mathbb{I} .

Proof of theorem 6.1.1. We define

$$(W \Rightarrow \Gamma.(a \doteq_A b)) := \{\gamma : W \Rightarrow \Gamma \mid a[\gamma] = b[\gamma]\}.$$

Thus, $\Gamma \vdash a \doteq_A b$ asserts that $a[\gamma] = b[\gamma]$ for *all* cells $\gamma : W \Rightarrow \Gamma$, i.e. $a = b$. \square

Proof of theorem 6.1.2. In the extensional system, we can define $\text{refl}_a := \text{tt}$ and $t := t_{\text{refl}}$. \square

6.1.5 System Notation

We adopt notational conventions, officially purely syntactic sugar, that are similar to the ‘system’ notation introduced by Cohen et al. [Coh+17].

Notation 6.1.5. While deciding inhabitation for any given proposition in a context, may be an undecidable problem, it is often easily resolved by a human reader. For this reason, we will prefer to use some more lightweight notations in which we omit proof variables and proof terms:

- Just like we may informally use pattern matching notation for matching deeper than one level, we will do so for this notation, e.g. $\{\varphi ? t_1 \mid \chi ? t_2 \mid \psi ? t_3\}$ can mean

$$\{\varphi ? t_1 \mid \chi \vee \psi ? \{\chi ? t_2 \mid \psi ? t_3\}\},$$

and $\{i := 0 ? t_0 \mid i := 1 ? t_1\}$ means

$$\{0 \doteq_{\mathbb{I}} i ? \{i := 0 ? t_0\} \mid 1 \doteq_{\mathbb{I}} i ? \{i := 1 ? t_1\}.\}$$

Of course remark 3.2.18 also applies.

6.1.6 Sketch of a Type Checking Algorithm

Without claiming a mathematical theorem, we motivate our choices in the intensional system by sketching how a type-checking algorithm might work. The idea is that, when type-checking a term $\Gamma \vdash t : T$, we force most or all propositions assumed by Γ . The ‘or’ appears because the actions taken will be slightly different for checking judgements with evidence (e.g. term, type and proposition judgements) and checking judgements without evidence (assertion and equality judgements). The actions needed to force φ are determined by induction:

- \wedge In order to force $\varphi \wedge \chi$, force φ and force χ .
- \top In order to force \top , do nothing.
- \vee When checking a judgement with evidence, we do not force disjunctions. When checking a judgement without evidence, in order to force $\varphi \vee \chi$, we simply check the judgement at hand twice in different extended contexts, namely (Γ, φ) and (Γ, χ) . That is, the judgement must hold when we force φ and it must hold when we force χ .
- \perp When checking a judgement with evidence, we do not force \perp . When checking a judgement without evidence, in order to force \perp , we simply do not check the judgement at all. That is, the judgement need not hold.
- φ In order to force a proposition variable, we just substitute it with \top . This is justified: the proposition is asserted and is a variable, so even in the intensional system we know that it is equal to \top anyway.
- \doteq The way to force equalities depends on the specific application. Typically, assumptions of the form $x \doteq_A a$ where x is a variable, will be forced by substituting a/x , as is justified by the J-rule for the equality proposition. Obvious falsities such as $0 \doteq_{\mathbb{I}} 1$ in a cubical type theory may be treated as \perp .

An assertion judgement $\Gamma \vdash \varphi$ is approved if φ reduces to \top after forcing. Other judgements are checked the same way as usual, but after forcing.

Example 6.1.6 (Cubical type theory). In presheaf type theory for cartesian cubical sets (example 2.3.10), we will typically have a closed **interval** type \mathbb{I} modelled by $\mathbf{y}(i : \mathbb{I})$. Similar to **Prop**, in the intensional system we will make this a pseudotype whose only terms are variables, 0 and 1. We then only allow \doteq to be used on the type \mathbb{I} , which makes it easy to force $i \doteq_{\mathbb{I}} j$: if either hand

is a variable, we substitute it for the other. If both are constants, then we force it as if it were \top (if the constants are equal) or \perp (if they are not).

For CCHM cubical sets (example 2.3.12), Cohen, Coquand, Huber, and Mörtberg [Coh+17] have a similar but more complicated procédé, where terms of the interval pseudotype are elements of the free de Morgan algebra over the available interval variables.

6.2 Types for Propositions

6.2.1 Proof Types

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1), logical truth (fig. 6.3).

$$\begin{array}{ccc}
 \text{PROOF} & \text{PROOF:INTRO} & \text{PROOF:ELIM} \\
 \frac{\Gamma \vdash \varphi \text{ prop}}{\Gamma \vdash [\varphi] \text{ type}_0} \text{E} & \frac{\Gamma \vdash \varphi}{\Gamma \vdash \text{tt} : [\varphi]} \text{E} & \frac{\Gamma \vdash p : [\varphi]}{\Gamma \vdash \varphi} \text{E} \\
 & \text{where } p = \text{tt} \quad (\text{PROOF:ETA}) &
 \end{array}$$

Figure 6.5: Typing rules for proof types.

In the extensional system, we will add a proof type constructor (fig. 6.5) which promotes propositions to types. This type constructor will allow us to implement many interesting types from the intensional system. The introduction rule inhabits $[\varphi]$ when φ holds; the elimination rule asserts φ when $[\varphi]$ is inhabited. The η -rule states that any element of $[\varphi]$ equals tt .

Proof of theorem 6.1.1. Given $\gamma : W \Rightarrow \Gamma$

$$[\varphi][\gamma] = \{\text{tt} \mid \gamma \in (W \Rightarrow \Gamma.\varphi)\}.$$

If $\Gamma \vdash \varphi$, then we know that $\Gamma.\varphi = \Gamma$, so we can define $\text{tt}[\gamma] = \text{tt}$. Conversely, if $\Gamma \vdash p : [\varphi]$, then $p[\varphi] \in \{\text{tt} \mid \gamma \in (W \Rightarrow \Gamma.\varphi)\}$ clearly equals tt , asserting the η -rule, and also asserts that $\gamma \in (W \Rightarrow \Gamma.\varphi)$. Since this holds for all $\gamma : W \Rightarrow \Gamma$, we conclude that $\Gamma.\varphi = \Gamma$, i.e. φ holds. \square

Remark 6.2.1. We could alternatively implement $[\varphi] := (\varphi \equiv_{\text{Prop}} \top)$.

6.2.2 Extension Types

The elimination rule for the disjunction has an equality judgement in its premises. In the extensional DTT this is unproblematic, but in intensional DTT we can expect situations where the terms we want to use are propositionally but not judgementally equal. In order to deal with such situations, we can extend intensional DTT with a type $A \text{ext}\{\varphi ? a\}$ of terms of type A which are judgementally equal to a when $\varphi = \top$.

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1), logical truth (fig. 6.3), proof types (fig. 6.5).

$$\begin{array}{c}
 \text{EXTENSION} \\
 \frac{\Gamma \vdash A \text{type}_\ell \quad \Gamma \vdash \varphi \text{prop} \quad \Gamma, \varphi \vdash a : A}{\Gamma \vdash A \text{ext}\{\varphi ? a\} \text{type}_\ell} \text{I} \\
 \\
 \begin{array}{cc}
 \text{EXTENSION:INTRO} & \text{EXTENSION:ELIM} \\
 \frac{\Gamma \vdash a : A \quad \Gamma \vdash \varphi \text{prop}}{\Gamma \vdash \text{cut}_\varphi a : A \text{ext}\{\varphi ? a\}} \text{I} & \frac{\Gamma \vdash e : A \text{ext}\{\varphi ? a\}}{\Gamma \vdash \text{paste}\{\varphi ? a | e\} : A} \text{I} \\
 \text{where } e = \text{cut}_\varphi(\text{paste}\{\varphi ? a | e\}) & \text{where } \text{paste}\{\top ? a | e\} = a \\
 (\text{EXTENSION:ETA}) & (\text{EXTENSION:STRICT}) \\
 & \text{paste}\{\varphi ? a | \text{cut}_\varphi a\} = a \\
 & (\text{EXTENSION:BETA})
 \end{array}
 \end{array}$$

Figure 6.6: Typing rules for extension types.

Figure 6.6 lists the typing rules for this extension type.³ The formation rule takes a type A , a proposition φ and a partial term a defined when φ holds. The introduction rule takes only a term $a : A$. By weakening that term to context (Γ, φ) , we get a partial term to be used in the type. Of course, since everything respects judgemental equality, we also have $\text{cut}_\varphi a : A \text{ext}\{\varphi ? b\}$ if $\Gamma, \varphi \vdash a = b : A$. The elimination rule promotes an element of $A \text{ext}\{\varphi ? a\}$ to an element of A that definitionally extends a on φ .

Proof of theorem 6.1.2. In the extensional system, we can define $A \text{ext}\{\varphi ? a\} = (x : A) \times ((p : [\varphi]) \rightarrow x \equiv_A a)$. The introduction and elimination rules are straightforward to implement and the required equations follow. \square

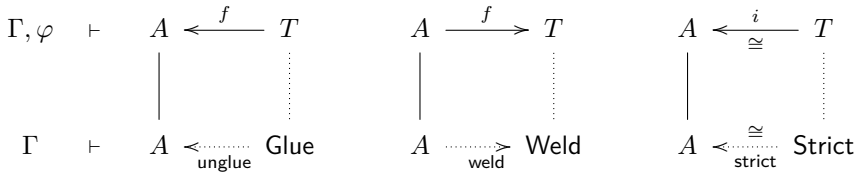
³Others [e.g. RS17] have used the term ‘extension type’ for what is essentially a quantification over an extension type, e.g. a construct such as $(i : \mathbb{I}) \rightarrow A \text{ext}\{i := 0 ? a_0 | i := 1 ? a_1\}$.

The idea is now that, if we have terms $\Gamma, \varphi \vdash t_{\text{inl}} : T$ and $\Gamma, \chi \vdash t_{\text{inr}} : T$ and we know that if $\varphi \wedge \chi$ holds, then $t_{\text{inl}} \equiv_T t_{\text{inr}}$ propositionally, then we get $\text{cut}_{\chi} t_{\text{inl}} : T \text{ext}\{\chi ? t_{\text{inl}}\}$, which we can convert to $e : T \text{ext}\{\chi ? t_{\text{inr}}\}$ using the J-rule, and then we get $\Gamma, \varphi \vdash \text{paste}\{\chi ? t_{\text{inr}} \mid e\} : T$ which is propositionally equal to t_{inl} and can be used as a clause of the disjunction eliminator, yielding

$$\Gamma, \varphi \vee \chi \vdash \{\varphi ? \text{paste}\{\chi ? t_{\text{inr}} \mid e\} \mid \chi ? t_{\text{inr}}\} : T.$$

6.3 Presheaf Types: Syntax and Presheaf Semantics

In this section, we extend dependent type theory with three type formers that are sound in every presheaf category: **Glue** [Coh+17; NVD17a], **Weld** [NVD17a] and **Strict** [OP18; Ort18]. Given a total type A and a partial slice $(T, f : T \rightarrow A)$ / coslice $(T, f : A \rightarrow T)$ / isoslice $(T, i : A \cong T)$, the type formers **Glue**/**Weld**/**Strict** (resp.) give us a universal extension of that partial slice/coslice/isoslice, which by the universal property is unique up to isomorphism. In the diagrams below, the bottom row shows a general situation which reduces to the top row when φ holds:



It is clear that, at least in the extensional system where we need not worry about η -rules, **Strict** will be definable internally from either **Glue** or **Weld**. Conversely, Orton and Pitts [OP18; Ort18] show that **Glue** is definable from **Strict**, by strictifying an internally definable pullback. Their construction can be dualized, but the required pushout is not definable internally. It is, however, sound in any presheaf category, so we provide a type former for it in section 6.3.3.

6.3.1 Strictification

The strictness axiom is one out of 9 axioms that Orton and Pitts [OP18; Ort18] rely on to build what we call a meta-internal⁴ model of cubical homotopy type

⁴By a meta-internal approach, we mean an approach that models a type system internally in another type system, as opposed to an auto-internal approach, that implements the special features of a type system in the system itself.

theory. It is sound in every presheaf category. Orton and Pitts present it as an axiom (a postulated invariant of a type expressing a certain theorem), but we present it (entirely equivalently) as a type former `Strict`. The strictness type is a bit obnoxious when it comes to β - and η -rules, so we only give it in the extensional system. The typing rules are given densely in fig. 6.7.

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1), logical truth (fig. 6.3).

$$\begin{array}{c}
 \Gamma \vdash A \text{ type}_\ell \quad \Gamma, \varphi \vdash T \text{ type}_\ell \\
 \Gamma \vdash \varphi \text{ prop} \quad \Gamma, \varphi \vdash i : T \cong A \\
 \hline
 \Gamma \vdash \text{Strict}\{A \cong (\varphi ? T, i)\} \text{ type}_\ell \quad (\text{STRICT}) \\
 \text{where } \text{Strict}\{A \cong (\top ? T, i)\} = T \quad (\text{STRICT:STRICT}) \\
 \Gamma \vdash \text{strict}\{\varphi ? i\} : \text{Strict}\{A \cong (\varphi ? T, i)\} \cong A \quad (\text{STRICT:ISO}) \\
 \text{where } \Gamma \vdash \text{strict}\{\top ? i\} = i \quad (\text{STRICT:ISO:STRICT})
 \end{array} \text{E}$$

Figure 6.7: Typing rules for the strictness type.

Given a total type A and a partial type T that is isomorphic to A , we get an extension $\text{Strict}\{A \cong (\varphi ? T, i)\}$ of T and an extension of the isomorphism. The presentation is dense in the sense that we would normally unpack the rule `STRICT:ISO` into four rules `STRICT:INTRO`, `STRICT:ELIM`, `STRICT:BETA` and `STRICT:ETA` corresponding to the four components of the isomorphism, and similar for `STRICT:ISO:STRICT`.

Proof of theorem 6.1.1. An analysis of the semantics of the type of isomorphisms, reveals that the information encoded in i is exactly an isomorphism of presheaves $A[\pi] \cong T$ (the weakening is over $\pi : \Gamma.\varphi \subseteq \Gamma$), where both presheaves are over \mathcal{W}/Γ . Given $\gamma : W \Rightarrow \Gamma$, we define

$$(W \triangleright \text{Strict}\{A \cong (\varphi ? T, i)\}[\gamma]) := \begin{cases} (W \triangleright T[\gamma]) & \text{if } \gamma \text{ is a cell of } \Gamma.\varphi, \\ (W \triangleright A[\gamma]) & \text{otherwise.} \end{cases}$$

Restriction along $\varphi : (V, \gamma \circ \varphi) \rightarrow (W, \gamma) : \mathcal{W}/\Gamma$ where $\gamma \circ \varphi$ is a cell of $\Gamma.\varphi$ but γ is not, is defined by applying the isomorphism.

Clearly, the resulting presheaf over \mathcal{W}/Γ extends T and is isomorphic to A by an isomorphism extending the one given by i . \square

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1), logical truth (fig. 6.3).

$$\begin{array}{c}
 \Gamma \vdash A \text{ type}_\ell \quad \Gamma, \varphi \vdash T \text{ type}_\ell \\
 \Gamma \vdash \varphi \text{ prop} \quad \Gamma, \varphi \vdash f : T \rightarrow A \\
 \hline
 \Gamma \vdash \text{Glue}\{A \leftarrow (\varphi ? T, f)\} \text{ type}_\ell \quad (\text{GLUE}) \\
 \text{where } \text{Glue}\{A \leftarrow (\top ? T, f)\} = T \quad (\text{GLUE:STRICT}) \\
 \Gamma \vdash \text{unglue}\{\varphi ? f\} : \text{Glue}\{A \leftarrow (\varphi ? T, f)\} \rightarrow A \quad (\text{GLUE:ELIM}) \\
 \text{where } \Gamma \vdash \text{unglue}\{\top ? f\} = f \quad (\text{GLUE:ELIM:STRICT}) \\
 \Gamma \vdash \text{unglue}\{\varphi ? f\}(\text{glue}\{a \leftarrow (\varphi ? t)\}) = a \quad (\text{GLUE:BETA}) \\
 \\
 \text{GLUE:INTRO} \\
 \Gamma \vdash a : A \quad \Gamma, \varphi \vdash t : T \quad \Gamma, \varphi \vdash ft = a : A \\
 \hline
 \Gamma \vdash \text{glue}\{a \leftarrow (\varphi ? t)\} : \text{Glue}\{A \leftarrow (\varphi ? T, f)\} \\
 \text{where } \text{glue}\{a \leftarrow (\top ? t)\} = t : T \quad (\text{GLUE:STRICT:BETA}) \\
 g = \text{glue}\{\text{unglue}\{\varphi ? f\}(g) \leftarrow (\varphi ? g)\} \quad (\text{GLUE:ETA})
 \end{array}$$

Figure 6.8: Typing rules for the Glue type.

6.3.2 Glueing: The Final Slice Extension Operation

If we replace every isomorphism sign (\cong) in the typing rules for Strict, we get excellent formation and elimination rules for the Glue type (fig. 6.8).⁵ However, we are still in need of a good introduction rule. In order to find out what that rule should look like, observe that in the extensional system we can obtain the final extension of the coslice (T, f) (up to isomorphism) by taking a pullback:

$$\begin{array}{ccc}
 \text{Glue}\{A \leftarrow (\varphi ? T, f)\} & \xrightarrow{\text{unglue}} & A \\
 \text{const} \downarrow \lrcorner & & \text{const} \downarrow \\
 (p : [\varphi]) \rightarrow T & \xrightarrow{f \circ \lrcorner} & [\varphi] \rightarrow A.
 \end{array}$$

Indeed, if $\varphi = \top$, then the vertical arrow $\text{const} : A \cong ([\top] \rightarrow A)$ becomes an isomorphism, so that the pullback will be isomorphic to $(p : [\top]) \rightarrow T$, which is isomorphic to T . This shows that the pullback extends T . Moreover, any slice (G, u) extending (T, f) will factor through the pullback, since we have $u : G \rightarrow A$ and $\text{const} : G \rightarrow ((p : [\varphi]) \rightarrow T)$ compatibly.

⁵Again, merely for brevity, we are giving a rule for $\text{unglue}\{\varphi ? f\}$ as a function instead of the fully applied $\text{unglue}\{\varphi ? f\}g$.

This shows that, despite the fact that there is only a single eliminator `unglue`, there are actually two ways to eliminate an element g of the `Glue` type: we can apply `unglue`, or we can assume φ and obtain $g : T$. Thus, a constructor by copattern matching will provide data for these two constructors which match up at $[\varphi] \rightarrow A$. This is exactly the input of the `glue` constructor.

Proof of theorem 6.1.2. [OP18; Ort18] From the above, it is immediately clear that we can define the `Glue`-type up to isomorphism as

$$\text{Glue}\{A \leftarrow (\varphi ? T, f)\} \cong (a : A) \times (t : (p : [\varphi]) \rightarrow T) \times ((p : [\varphi]) \rightarrow (f t \equiv_A a)).$$

If φ holds, the right hand side is isomorphic to T , so we can use `Strict` to obtain a type isomorphic to the above that strictly extends T . The constructor and eliminator are trivial to implement. \square

6.3.3 Pushout Type

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1), logical truth (fig. 6.3).

<p style="margin: 0;">PUSHOUT</p> $\frac{\Gamma \vdash A, B \text{ type}_\ell \quad \Gamma, y : B \vdash \varphi \text{ prop} \quad \Gamma, y : B, \varphi \vdash a : A}{\Gamma \vdash A \uplus (y : B \uparrow \{\varphi ? a\}) \text{ type}_\ell}$	<p style="margin: 0;">PUSHOUT:INL</p> $\frac{\Gamma \vdash a : A}{\Gamma \vdash \text{inl } a : A \uplus (y : B \uparrow \{\varphi ? a\})}$
<p style="margin: 0;">PUSHOUT:INR</p> $\frac{\Gamma \vdash b : B}{\Gamma \vdash \text{inr } b : A \uplus (y : B \uparrow \{\varphi ? a\})}$ <p style="margin: 0;">where $\Gamma, \varphi[b/y] \vdash \text{inr } b = \text{inl } a[b/y]$ (PUSHOUT:TIP)</p>	
<p style="margin: 0;">PUSHOUT:ELIM</p> $\frac{\Gamma, z : A \uplus (y : B \uparrow \{\varphi ? a\}) \vdash T \text{ type} \quad \Gamma, x : A \vdash t_{\text{inl}} : T[\text{inl } x/z] \quad \Gamma, y : B \vdash t_{\text{inr}} : T[\text{inr } y/z] \quad \Gamma, y : B, \varphi \vdash t_{\text{inr}} = t_{\text{inl}}[a[y/y]/x] : T[\text{inr } y/z]}{\Gamma \vdash c : A \uplus (y : B \uparrow \{\varphi ? a\}) \rightarrow T[c/z]}$ <p style="margin: 0;">where $t[\text{inl } a_1/c] = t_{\text{inl}}[a_1/x]$ (PUSHOUT:INL:BETA)</p> <p style="margin: 0;">$t[\text{inr } b/c] = t_{\text{inr}}[b/y]$ (PUSHOUT:INR:BETA)</p>	

Figure 6.9: Typing rules for the pushout type.

In order to dualize the above reasoning for the `Weld` type, we need to be able to take the dual pushout internally. We add a pushout type former for taking a class of pushouts that is only slightly larger than what we need:⁶ given the inputs of the `PUSHOUT` rule (fig. 6.9), we want to take the pushout of the following cospan constructed in the extensional system:

$$\begin{array}{ccc}
 (y : B) \times [\varphi] & \xrightarrow{\lambda y. \lambda p. a} & A \\
 \text{fst} \downarrow & & \downarrow \text{inl} \\
 B & \xrightarrow{\text{inr}} & A \uplus (y : B \uparrow \{\varphi ? a\}).
 \end{array}$$

Clearly the pushout type should get two constructors `inl` and `inr` which match up when φ holds. The eliminator needs to handle these two constructors in a compatible way.

Proof of theorem 6.1.1. All presheaf categories have pushouts (proposition 2.3.6), so this holds in particular for $\mathbf{Psh}(\mathcal{W}/\Gamma)$. The only non-trivial remaining thing is that pushouts admit a dependent eliminator. This however follows from the fact that terms of type T are just substitutions to $\Gamma.P.T$ (where P is the pushout type) whose weakening is predetermined. \square

6.3.4 Welding: The Initial Coslice Extension Operation

Again, if we replace the isomorphism signs in the typing rules for `Strict` with arrows, now pointing the other way, we get formation and introduction rules for `Weld`. In order to eliminate, we observe that in the extensional system, the initial coslice over (T, f) can be obtained as a pushout:

$$\begin{array}{ccc}
 [\varphi] \times A & \xrightarrow{\text{id} \times f} & (p : [\varphi]) \times T \\
 \text{snd} \downarrow & & \downarrow \text{snd} \\
 A & \xrightarrow{\text{weld}} & \text{Weld}\{A \uparrow (\varphi ? Tf)\}.
 \end{array}$$

by reasoning dual to that in section 6.3.2. This shows that there are two ways to construct an element of the `Weld` type: either by using the `weld` constructor or

⁶We could of course add a general pushout type former, but the one introduced here works even in the intensional system satisfying the definitional equalities that `Weld` needs to inherit. This can become relevant if we actually want to apply this construction in a programming language using an intensional version of `Strict`.

Prerequisites: DTT (fig. 3.2), propositions (fig. 6.1), logical truth (fig. 6.3).

$$\begin{array}{c}
 \Gamma \vdash A \text{ type}_\ell \quad \Gamma, \varphi \vdash T \text{ type}_\ell \\
 \Gamma \vdash \varphi \text{ prop} \quad \Gamma, \varphi \vdash f : A \rightarrow T \\
 \hline
 \Gamma \vdash \text{Weld}\{A \rightarrow (\varphi ? T, f)\} \text{ type}_\ell \quad (\text{WELD}) \\
 \text{where } \text{Weld}\{A \rightarrow (\top ? T, f)\} = T \quad (\text{WELD:STRICT}) \\
 \Gamma \vdash \text{weld}\{\varphi ? f\} : A \rightarrow \text{Weld}\{A \rightarrow (\varphi ? T, f)\} \quad (\text{WELD:INTRO}) \\
 \text{where } \Gamma \vdash \text{weld}\{\top ? f\} = f \quad (\text{WELD:INTRO:STRICT}) \\
 \\
 \text{WELD:ELIM} \\
 \Gamma, z : \text{Weld}\{A \rightarrow (\varphi ? T, f)\} \vdash S \text{ type} \\
 \Gamma, x : A \vdash s_{\text{weld}} : S[\text{weld}\{\varphi ? f\} x/z] \\
 \Gamma, \varphi, z : T \vdash s_\top : S \\
 \Gamma, \varphi, x : A \vdash s_{\text{weld}} = s_\top[f x/z] : S[f x/z] \\
 \Gamma \vdash w : \text{Weld}\{A \rightarrow (\varphi ? T, f)\} \\
 \hline
 \Gamma \vdash s := \text{case } w \text{ of } \{\text{weld}\{\top ? f\} x \mapsto s_{\text{weld}} \mid \varphi ? z \mapsto s_\top\} : S[w/z] \quad \text{I} \\
 \text{where } s[\text{weld}\{\varphi ? f\} a/w] = s_{\text{weld}}[a/x] \quad (\text{WELD:BETA}) \\
 s[\top/\varphi] = s_\top[w/z] \quad (\text{WELD:STRICT:BETA})
 \end{array}$$

Figure 6.10: Typing rules for the Weld type.

by asserting φ and taking an element of T . Hence, we give a pattern matching eliminator that requires the user to handle both cases compatibly.

Proof of theorem 6.1.2. We can define the Weld-type up to isomorphism by the following pushout:

$$\text{Weld}\{A \rightarrow (\varphi ? T f)\} := ((p : [\varphi]) \times T) \uplus (x : A \uparrow \{\varphi ? (\text{tt}, f x)\}).$$

When φ holds, the `inr` constructor becomes redundant and the right hand side becomes isomorphic to T , so we can use `Strict` to obtain a type isomorphic to the above that strictly extends T . The constructor and eliminator are straightforward to implement. \square

6.4 Presheaf Modalities

In this section, we prove that certain functors involving presheaf categories are automatically strict or weak CwF morphisms.

6.4.1 Right Adjoint Presheaf Functors

Theorem 6.4.1. If a functor $R : \mathcal{C} \rightarrow \text{Psh}(\mathcal{W})$ from an arbitrary CwF to a presheaf CwF has a left adjoint L , then it is automatically a weak CwF morphism.

Proof. Given $\Gamma \vdash T \text{ type } @ \mathcal{C}$, we need to define $R\Gamma \vdash RT \text{ type } @ \text{Psh}(\mathcal{W})$. So pick $\delta : W \Rightarrow R\Gamma$ and write $\gamma = \mathbf{A}^{-1}(\delta) : LyW \rightarrow \Gamma$, so $\delta = \mathbf{A}(\gamma)$. Then we need to define $(W \triangleright (RT)[\mathbf{A}(\gamma)])$. By theorem 4.1.4, this will be isomorphic to $(yW \vdash (RT)[\mathbf{A}(\gamma)])$, which by proposition 5.1.9 will be isomorphic to $(LyW \vdash T[\gamma])$. So we *must* (up to isomorphism) define

$$(W \triangleright (RT)[\mathbf{A}(\gamma)]) := \{\mathbf{A}(t) \mid LyW \vdash t : T[\gamma]\}.$$

The symbol \mathbf{A} is a label (notation 2.0.1). Restriction is of course given by $\mathbf{A}(t)\langle\varphi\rangle := \mathbf{A}(t[Ly\varphi])$.

We have to show that the action of R on types defined above is natural, i.e. that $(RT)[R\sigma] = R(T[\sigma])$. We have:

$$\begin{aligned} (W \triangleright (RT)[R\sigma][\mathbf{A}(\gamma)]) &= (W \triangleright (RT)[R\sigma \circ \mathbf{A}(\gamma)]) \\ &= (W \triangleright (RT)[\mathbf{A}(\sigma \circ \gamma)]) \\ &= (LyW \vdash T[\sigma \circ \gamma]) \\ &= (LyW \vdash T[\sigma][\gamma]) \\ &= (W \triangleright (R(T[\sigma]))[\mathbf{A}(\gamma)]). \end{aligned}$$

Given $\Gamma \vdash t : T$, we must define $R\Gamma \vdash Rt : RT$. Again, we have no choice. Indeed, pick $\delta = \mathbf{A}(\gamma) : W \Rightarrow R\Gamma$. By corollary 5.1.12, we will have

$$(Rt)[\mathbf{A}(\gamma)] = (Rt)[\mathbf{A}(\gamma)][\text{id}] = \mathbf{A}(t[\gamma])[\text{id}] = \mathbf{A}(t[\gamma]).$$

So we should define $(Rt)[\mathbf{A}(\gamma)] := \mathbf{A}(t[\gamma])$. This is compatible with restriction:

$$\begin{aligned} (Rt)[\mathbf{A}(\gamma)]\langle\varphi\rangle &= \mathbf{A}(t[\gamma])\langle\varphi\rangle = \mathbf{A}(t[\gamma][Ly\varphi]) \\ &= (Rt)[\mathbf{A}(\gamma \circ Ly\varphi)] = (Rt)[\mathbf{A}(\gamma) \circ \varphi]. \end{aligned}$$

Again, we have to show that the action of R on terms is natural, i.e. that $(Rt)[R\sigma] = R(t[\sigma])$, but the reasoning is similar to the reasoning for types.

Being a right adjoint, R preserves the terminal object (the empty context) up to isomorphism.

For context extension, we have:

$$\begin{aligned}
(W \Rightarrow R(\Gamma.T)) &\cong (LyW \rightarrow \Gamma.T) \\
&\cong (\sigma : LyW \rightarrow \Gamma) \times (LyW \vdash T[\sigma]) \\
&= (\delta : W \Rightarrow R\Gamma) \times (W \triangleright (RT)[\delta]) \\
&= (W \Rightarrow R\Gamma.RT),
\end{aligned}$$

so context extension is preserved up to isomorphism. \square

Corollary 6.4.2. The right lifting⁷ (theorem 2.3.32) $F_* : \mathbf{Psh}(\mathcal{V}) \rightarrow \mathbf{Psh}(\mathcal{W})$ of a functor $F : \mathcal{V} \rightarrow \mathcal{W}$ is a weak CwF morphism. \square

We will write $(W \triangleright (F_*T)[\mathbf{B}_F(\gamma)]) := \{\mathbf{B}_F(t) \mid F^*\mathbf{y}W \vdash t : T[\gamma]\}$.

6.4.2 Central Liftings of Base Functors

Theorem 6.4.3. The central lifting (theorem 2.3.32) $F^* : \mathbf{Psh}(\mathcal{W}) \rightarrow \mathbf{Psh}(\mathcal{V})$ of a functor $F : \mathcal{V} \rightarrow \mathcal{W}$ is a strict CwF morphism.

Proof. In the proof of theorem 6.4.1, we saw that up to isomorphism, we never had a choice. So the only thing we can do now is redefine the action on types in a way isomorphic to the one used before, and hope that the resulting CwF morphism will be strict. To this end, we use the property that $F_! \circ \mathbf{y} \cong \mathbf{y} \circ F$, and define

$$\begin{aligned}
(V \triangleright (F^*T)[\mathbf{A}_F(\gamma)]) &:= \{\mathbf{A}_F(t) \mid FV \triangleright t : T[\gamma]\}, \\
(F^*t)[\mathbf{A}_F(\gamma)] &:= \mathbf{A}_F(t[\gamma]).
\end{aligned}$$

We then have

$$\begin{aligned}
(V \Rightarrow F^*(\Gamma.T)) &= (FV \Rightarrow \Gamma.T) \\
&= (\gamma : FV \Rightarrow \Gamma) \times (FV \triangleright T[\gamma]) \\
&= (\mathbf{A}_F(\gamma) : V \Rightarrow F^*\Gamma) \times (V \triangleright (F^*T)[\mathbf{A}_F(\gamma)])
\end{aligned}$$

⁷For central liftings, we have a stronger theorem 6.4.3.

$$= (V \Rightarrow F^* \Gamma . F^* T),$$

and similar for the empty context. \square

6.5 Presheaf MTT

Prerequisites: Structural rules for propositions fig. 6.1, MTT fig. 5.5.

Modal context restriction:

Generalizes CTX-RESTR and corresponding rules.

CTX-MODRESTR

$$\frac{\begin{array}{l} \Gamma \text{ctx} @ q \\ \Gamma, \mathbf{\mu}_{\mu}^m \vdash \varphi \text{prop} @ p \\ \mu : p \rightarrow q \end{array}}{\Gamma, \mu^{im} \varphi \text{ctx} @ q}$$

CTX-MODRESTR:INTRO

$$\frac{\begin{array}{l} \sigma : \Delta \rightarrow \Gamma @ q \quad \mu : p \rightarrow q \\ \Delta, \mathbf{\mu}_{\mu}^m \vdash \varphi[\downarrow_{\mu}^m] @ p \end{array}}{(\sigma, \text{ok}) : \Delta \rightarrow (\Gamma, \mu^{im} \varphi) @ q} \\ \text{where } (\sigma, \text{ok}) \circ \rho = (\sigma \circ \rho, \text{ok}) \\ \tau = (\pi \circ \tau, \text{ok}) \quad (\text{CTX-MODRESTR:ETA})$$

CTX-MODRESTR:WKN

$$\frac{\begin{array}{l} \Gamma \text{ctx} @ q \\ \Gamma, \mathbf{\mu}_{\mu}^m \vdash \varphi \text{prop} @ p \end{array}}{\pi : (\Gamma, \mu^{im} \varphi) \rightarrow \Gamma @ q} \\ \text{where } \pi \circ (\sigma, \text{ok}) = \sigma \\ (\text{CTX-MODRESTR:WKN:BETA})$$

CTX-MODRESTR:VAR

$$\frac{\begin{array}{l} \Gamma \text{ctx} @ q \\ \Gamma, \mathbf{\mu}_{\mu}^m \vdash \varphi \text{prop} @ p \\ \alpha : \mu \Rightarrow \text{locks}(\Theta) \end{array}}{\Gamma, \mu^{im} \varphi, \Theta \vdash \varphi[\alpha \downarrow_{\text{ticks}(\Theta)}^m] @ p}$$

DRA for propositions:

PROPDRA

$$\frac{\begin{array}{l} \mu : p \rightarrow q \\ \Gamma \text{ctx} @ q \\ \Gamma, \mathbf{\mu}_{\mu}^m \vdash \varphi \text{prop} @ p \end{array}}{\Gamma \vdash \langle \mu \mid \varphi \rangle \text{prop} @ q} \\ \text{where } \langle \mu \mid \top \rangle = \top \\ (\text{PROPDRA:RED})$$

PROPDRA:INTRO

$$\frac{\begin{array}{l} \mu : p \rightarrow q \\ \Gamma \text{ctx} @ q \\ \Gamma, \mathbf{\mu}_{\mu}^m \vdash \varphi @ p \end{array}}{\Gamma \vdash \langle \mu \mid \varphi \rangle @ q}$$

PROPDRA:ELIM

$$\frac{\begin{array}{l} \mu : p \rightarrow q \\ \Gamma \text{ctx} @ q \\ \Gamma \vdash \langle \mu \mid \varphi \rangle @ q \end{array}}{\Gamma, \mathbf{\mu}_{\mu}^m \vdash \varphi @ p}$$

Figure 6.11: Modal typing rules for presheaf type theory

In this section, we briefly consider some additional typing rules that are of interest when using MTT modelled in presheaf categories. Figure 6.11 lists typing rules for modal context restriction, which generalizes context restriction

(fig. 6.1) in the same way that modal context extension (fig. 5.5) generalizes ordinary context extension (fig. 3.2).

On top of that, fig. 6.11 lists typing rules for applying a DRA to a proposition. We choose a DRA, rather than an MTT-style weak DRA, because most presheaf modalities arise as central or right liftings anyway and are therefore DRAs. Of course every DRA is also a weak DRA. These rules are the obvious proof-irrelevant counterpart of those in fig. 5.4. Note that PROPDRA:INTRO is actually unnecessary as it follows from PROPDRA:RED and UT .

Theorem 6.5.1. The typing rules in fig. 6.11 are sound in any model of MTT which interprets modes as presheaf categories and modalities as DRAs.

Proof. We first model the DRA. Write $L = \llbracket \mathbf{!}_\mu \rrbracket$. We have

$$\begin{aligned} (W \triangleright \langle \mu \mid \varphi \rangle[\gamma]) &\Leftrightarrow (\mathbf{y}W \vdash \langle \mu \mid \varphi \rangle[\gamma]) \\ &= (\mathbf{y}W \vdash \langle \mu \mid \varphi[L\gamma] \rangle) \Leftrightarrow (L\mathbf{y}W \vdash \varphi[L\gamma]) \end{aligned}$$

so we (must) define

$$(W \triangleright \langle \mu \mid \varphi \rangle[\gamma]) : \Leftrightarrow (L\mathbf{y}W \vdash \varphi[L\gamma]).$$

It follows that $[\langle \mu \mid \varphi \rangle] \cong \langle \mu \mid [\varphi] \rangle$ so that the introduction and elimination rules clearly hold.

Next, we model $(\Gamma, \mu \mid \varphi)$ as $(\Gamma, \langle \mu \mid \varphi \rangle)$. Then CTX-MODRESTR and CTX-MODRESTR:WKN are clearly sound, whereas CTX-MODRESTR:VAR is derived by using, in this order, CTX-RESTR:VAR , PROPDRA:ELIM , LOCK:FMAP over α , and then strict functoriality of locks and weakening to turn $\mathbf{!}_{\text{locks}(\Theta)}$ into Θ . \square

6.5.1 Extending the Type Checking Algorithm

Regarding the algorithm sketched in section 6.1.6:

- Of course, we force $\langle \mu \mid \varphi \rangle$ by turning it into a modal assumption $\mu \mid \varphi$.
- The forcing approaches for equality, conjunction, truth and DRAs concern isomorphisms of contexts which are preserved by locks and modalities. As such, these forcing approaches can be executed unproblematically under a lock/modality.
- Left adjoints, such as locks, preserve colimits, so that the forcing approaches for disjunction and falsehood can be executed unproblematically under a lock or under a modality that has a right adjoint.

- We do not force disjunctions and falsehoods under a modality that does not have a right adjoint. For example, it does not make sense to force \perp under the final modality sending every type to the unit type, or to force $(i \dot{=} 0) \vee (i \dot{=} 1)$ under a codiscrete modality.

Chapter 7

Transpension: The Right Adjoint to the Π -type

Preamble This chapter contains an extended version of a presently unpublished paper with Dominique Devriese [ND20], and has an associated technical report [Nuy20] which is not included in this thesis for reasons of space. Section 7.5, which introduces an informal and friendlier notation, was added on the occasion of this thesis. A section introducing MTT (section 7.3) was largely removed, as we can refer to section 5.3.

Personal contributions Dominique Devriese incited this research by attempting to use `Glue` and `Weld` to prove parametricity of an internal System F universe in ParamDTT [NVD17a] in order to benchmark their expressivity, and obtaining convincing non-positive results. Other than that, his contribution is primarily in the area of writing. Apart from this input, the work described in this chapter is entirely my own.

Abstract Presheaf models of dependent type theory have been successfully applied to model HoTT, parametricity, and directed, guarded and nominal type theory. There has been considerable interest in internalizing aspects of these presheaf models, either to make the resulting language more expressive, or in order to carry out further reasoning internally, allowing greater abstraction and sometimes automated verification. While the constructions of presheaf models largely follow a common pattern, approaches towards internalization do not. Throughout the literature, various internal presheaf operators (\surd , Φ/extent ,

Ψ/Gel , Glue , Weld , mill , the strictness axiom and locally fresh names) can be found and little is known about their relative expressiveness. Moreover, some of these require that variables whose type is a shape (representable presheaf) be used affinely.

We propose a novel type former, the transpension type, which is right adjoint to universal quantification over a shape. Its structure resembles a dependent version of the suspension type in HoTT . We give general typing rules and a presheaf semantics in terms of base category functors dubbed multipliers. Structural rules for shape variables and certain aspects of the transpension type depend on characteristics of the multiplier. We demonstrate how the transpension type and the strictness axiom can be combined to implement all and improve some of the aforementioned internalization operators (without formal claim in the case of locally fresh names).

7.1 Introduction and Related Work

7.1.1 The Power of Presheaves

Presheaf semantics (see chapter 4) [Hof97; HS97] are an excellent tool for modelling relational preservation properties of (dependent) type theory. They have been applied to parametricity (which is about preservation of relations) [AGJ14; BCM15; ND18a; NVD17a], univalent type theory (preservation of equivalences) [BCH14; CMS20; Coh+17; Hub16; KLV12; Ort18; OP18], directed type theory (preservation of morphisms), guarded type theory (preservation of the stage of advancement of computation) [BM18] and even combinations thereof [Bir+19; CH20; RS17; WL20].¹ The presheaf models cited almost all follow a common pattern: First one chooses a suitable base category \mathcal{W} . The presheaf category over \mathcal{W} is automatically a model of dependent type theory with important basic type formers [Hof97] as well as a tower of universes [HS97]. Next, one identifies a suitable notion of fibrancy (see section 2.4 and chapter 8) and replaces or supplements the existing type judgement $\Gamma \vdash T$ type with one that classifies fibrant types:

HoTT For homotopy type theory (HoTT , [Uni13]), one considers Kan fibrant types, i.e. presheaves in which edges can be composed and inverted as in an ∞ -groupoid. The precise definition may differ in different treatments.

Parametricity For parametric type theory, one considers discrete types [AGJ14; CH20; ND18a; NVD17a]: essentially those that satisfy Reynolds’

¹We omit models that are not explicitly structured as presheaf models [AHH18; LH11; Nor19].

identity extension property [Rey83] which states that homogeneously related objects are equal.

Directed In directed type theory, one may want to consider Segal, covariant, discrete and Rezk types [RS17] and possibly also Conduché types [Gir64; Nuy18b].

Guarded In guarded type theory, one considers clock-irrelevant types [BM18]: types A such that any non-dependent function $\odot \rightarrow A$ from the clock type, is constant.

Nominal Nominal type theory [Che12; PMD15] can be modelled in the Schanuel topos [Pit14].²

To the extent possible, one subsequently proves that the relevant notions of fibrancy are closed under basic type formers, so that we can restrict to fibrant types and still carry out most of the familiar type-theoretic reasoning and programming. Special care is required for the universe: it is generally straightforward to adapt the standard Hofmann-Streicher universe to classify only fibrant types, but the universe of fibrant types is in general not automatically fibrant itself. In earlier work on parametricity with Vezzosi [NVD17a; ND18a], we made the universe discrete by modifying its presheaf structure and introduced a parametric modality in order to use that universe. In contrast, Atkey et al. [AGJ14] and Cavallo and Harper [CH20] simply accept that their universes of discrete types are not discrete. In guarded type theory, Bizjak et al. [Biz+16] let the universe depend on a collection of in-scope clock variables lest the clock-indexed later modality $\triangleright : \forall(\kappa : \odot). \mathbf{U}_\Delta \rightarrow \mathbf{U}_\Delta$ (where $\kappa \in \Delta$) be non-dependent and therefore constant (not clock-indexed) by clock-irrelevance of $\mathbf{U}_\Delta \rightarrow \mathbf{U}_\Delta$ [BM18].

7.1.2 Internalizing the Power of Presheaves

Purely metatheoretic results about type theory certainly have their value. Parametricity, for instance, has originated and proven its value as a metatheoretic technique for reasoning about programs. However, with dependent type theory being not only a programming language but also a logic, it is preferable to formulate results about it within the type system, rather than outside it.

²This is the full subcategory of the category of nullary affine cubical sets $\mathbf{Psh}({}^0\mathbf{Cube}_\square)$ consisting of presheaves sending pushouts (intersections of dimensions) in ${}^0\mathbf{Cube}_\square$ to pullbacks in \mathbf{Set} , i.e. presheaves such that if a cell γ is degenerate in dimension i ($\gamma = \gamma_1 \circ (i/\odot)$) and degenerate in dimension j ($\gamma = \gamma_2 \circ (j/\odot)$), then it is uniquely simultaneously degenerate in both dimensions ($\gamma_1 = \gamma_3 \circ (j/\odot)$ and $\gamma_2 = \gamma_3 \circ (i/\odot)$).

Enlarging the end user’s toolbox One motivation for internalizing metatheorems is to enlarge the toolbox of the end user of the proof assistant. If this is the only goal, then we can prove the desired results in the model on pen and paper and then internalize them ad hoc with an axiom with or without computation rules.

HoTT Book HoTT [Uni13] simply postulates the univalence axiom without computational behaviour, as justified e.g. by the model of Kan-fibrant simplicial sets [KLV12].

CCHM cubical type theory [Coh+17] provides the **Glue** type, which comes with introduction, elimination, β - and η -rules and which turns the univalence axiom into a theorem with computational behaviour. It also contains CCHM-Kan-fibrancy of all types as an axiom, in the form of the CCHM-Kan composition operator, with decreed computational behaviour that is defined by induction on the type.

Parametricity Bernardy, Coquand and Moulin [BCM15; Mou16] (henceforth: Moulin et al.) internalize their (unary, but generalizable to k -ary) cubical set model of parametricity using two combinators Φ and Ψ [Mou16], a.k.a. **extent** and **Gel** [CH20]. Φ internalizes the presheaf structure of the function type, and Ψ that of the universe.

The combinator Φ and at first sight also Ψ require that the cubical set model lacks diagonals. Indeed, to construct a value over the primitive interval, Φ and Ψ each take one argument for every endpoint and one argument for the edge as a whole. Nested use of these combinators, e.g. to create a square, will take $(k + 1)^2$ arguments for k^2 vertices, $2k$ sides and 1 square as a whole but none for specifying the diagonal. For this reason, Moulin et al.’s type system enforces a form of *affine* use of interval variables.

In earlier work with Vezzosi [NVD17a], we have internalized parametricity instead using the **Glue** type [Coh+17] and its dual **Weld**. Later on, we added a primitive **mill** [ND18b] for swapping **Weld** and $\Pi(i : \mathbb{I})$. These operations are sound in presheaves over any base category where we can multiply with \mathbb{I} , and therefore strictly less expressive than Φ which is not. Discreteness of all types was internalized as a non-computing *path degeneracy* axiom.

Directed Weaver and Licata [WL20] use a bicubical set model to show that directed HoTT [RS17] can be soundly extended with a directed univalence axiom.

Guarded In guarded type theory [BM18], one axiomatizes Löb induction and clock-irrelevance.

Nominal One version of nominal type theory [PMD15] provides the locally fresh name abstraction $\nu(i : \mathbb{I})$ which introduces a name but requires a body that is fresh for the name (i.e. we do not get to use it) and can be used anywhere. This would be rather useless, were it not that we are allowed to *abstract* over the fresh name (see section 7.8).

Internalizing fibrancy proofs Another motivation to internalize aspects of presheaf categories, is for building parts of the model inside the type theory, thus abstracting away certain categorical details such as the very definition of presheaves, and for some type systems enabling automatic verification of proofs. Given the common pattern in models described in the previous section, it is particularly attractive to try and define fibrancy and prove results about it internally.

In the context of HoTT, Orton and Pitts [Ort18; OP18] study CCHM-Kan-fibrancy [Coh+17] in a type theory satisfying a set of axioms, of which all but one serve to characterize the interval and the notion of cofibration. One axiom, *strictness*, provides a type former `Strict` for strictifying partial isomorphisms, which exists in every presheaf category. In order to prove fibrancy of the universe, Licata et al. postulate an “amazing right adjoint” $\mathbb{I}\sqrt{\sqsubset}$ to the non-dependent path functor $\mathbb{I} \rightarrow \sqsubset$ [Lic+18; Ort18], which indeed exists in presheaves over cartesian base categories if \mathbb{I} is representable. Since $\mathbb{I}\sqrt{\sqsubset}$ and its related axioms are global operations (only applicable to closed terms, unless you want to open Pandora’s box as we do in the current paper), they keep everything sound by introducing a judgemental comonadic *global* modality \flat .

Orton et al.’s formalization [Lic+18; Ort18; OP18] is only what we call *meta-internal*: the argument is internalized to *some* type theory which still only serves as a metatheory of the type system of interest. Ideally, we would define and prove fibrancy of types *within* the type theory of interest, which we call *auto-internal*. Such treatments exist of discrete types in parametricity [CH20], and discrete, Segal and Rezk types in directed type theory [RS17], but not yet for covariant or CCHM-Kan-fibrant types due to the need to consider paths in the context $\mathbb{I} \rightarrow \Gamma$.

7.1.3 The Transpension Type

What is striking about the previous section is that, while most authors have been able to solve their own problems, a common approach is completely absent. We have encountered Φ and Ψ [Mou16], the amazing right adjoint $\sqrt{\quad}$ [Lic+18], Glue [Coh+17; NVD17a], Weld [NVD17a], mill [ND18b] and the strictness axiom

[OP18]. We have also seen that Φ and Ψ presently require an affine base category, and that \surd presently requires the global modality \flat .

The goal of the current paper is to develop a smaller collection of internal primitives that impose few restrictions on the choice of base category and allow the internal construction of the aforementioned operators when sound. To this end, we introduce the **transpension** type former $\check{\exists}i : \text{Ty}(\Gamma) \rightarrow \text{Ty}(\Gamma, i : \mathbb{I})$ which in cartesian settings is right adjoint to $\Pi(i : \mathbb{I}) : \text{Ty}(\Gamma, i : \mathbb{I}) \rightarrow \text{Ty}(\Gamma)$ and is therefore not a quantifier binding i , but a coquantifier that *depends* on it. Using the transpension and **Strict**, we can construct Φ (when sound), Ψ , \surd and **Glue**. Given a type former for certain pushouts, we can also construct **Weld**.

The transpension coquantifier $\check{\exists}(u : \mathbb{U}) : \text{Ty}(\Gamma) \rightarrow \text{Ty}(\Gamma, u : \mathbb{U})$ is part of a sequence of adjoints $\Sigma u \dashv \Omega u \dashv \Pi u \dashv \check{\exists} u$, preceded by the Σ -type, weakening and the Π -type. Adjointness of the first three is provable from the structural rules of type theory. However, it is not immediately clear how to add typing rules for a further adjoint. Birkedal et al. [Bir+20] explain how to add a single modality that has a left adjoint in the semantics. If we want to have two or more adjoint modalities internally, then we can use a multimodal type system such as MTT [Gra+20b; Gra+20a]. Each modality in MTT needs a semantic left adjoint, so we can only internalize Ωu , Πu and $\check{\exists} u$. A drawback which we accept, is that Ωu and Πu become modalities which are a bit more awkward to deal with than ordinary weakening and Π -types.

7.1.4 Contributions

Our central contribution is to reduce the plethora of internal presheaf operators in the literature to only a few operations.

- To this end, we introduce the **transpension type** $\check{\exists}(u : \mathbb{U})$, right adjoint to $\Pi(u : \mathbb{U})$, with typing rules built on *extensional* MTT [Gra+20b; Gra+20a]. We explain how it is reminiscent of the suspension type from HoTT [Uni13].
- More generally, the transpension type can be right adjoint to any quantifier-like operation $\forall(u : \mathbb{U})$ which need neither respect the exchange rule, nor weakening or contraction. In this setting, we also introduce the **fresh weakening** coquantifier $\exists(u : \mathbb{U})$, which is left adjoint to $\forall(u : \mathbb{U})$ and therefore coincides with weakening $\Omega(u : \mathbb{U})$ in cartesian settings.
- We provide a categorical semantics for $\check{\exists}(u : \mathbb{U})$ in almost any presheaf category $\text{Psh}(\mathcal{W})$ over base category \mathcal{W} , for almost any representable object $\mathbb{U} = \mathbf{y}U$, $U \in \mathcal{W}$. To accommodate non-cartesian variables, our system is not parametrized by a representable object $\mathbb{U} = \mathbf{y}U$,

but by an arbitrary endofunctor $\sqsubset \times U$ on \mathcal{W} : the **multiplier**. We introduce **criteria** for characterizing the multiplier – viz. semi-cartesian, cartesian, cancellative, affine, connection-free and quantifiable – which we use as requirements for internal type theoretic features. We identify a complication dubbed **spookiness** in certain models (most notably in guarded type theory), and define dimensionally split morphisms (a generalization of split epimorphisms) in order to include spooky models. We exhibit relevant multipliers in base categories found in the literature (fig. 7.2).

- We show that **all general presheaf internalization operators** that we are aware of – viz. Φ/extent (when sound), Ψ/Gel [Mou16; BCM15], the amazing right adjoint \surd [Lic+18], *Glue* [Coh+17; NVD17a], *Weld* [NVD17a], *mill* [ND18b] and (with no formal claim) locally fresh names – can be **recovered** from just the transpension type, the strictness axiom and pushouts along $\text{snd} : \varphi \times A \rightarrow A$ where $\varphi : \text{Prop}$. In the process, some of these operators can be **improved**: We generalize Ψ to arbitrary multipliers, including cartesian ones and we justify $\mathbb{U} \surd \sqsubset$ without a global modality and get proper computation rules for it. Moreover, since our system provides an operation $\mathbb{A}_{\surd u}$ for quantifying over contexts, we take a step towards auto-internalizing Orton et al.’s work [Lic+18; Ort18; OP18]. When Φ is not sound (e.g. in cartesian or non-connection-free settings), we suggest the internal notion of **transpensive** types to retain some of its power. Finally, a form of higher dimensional pattern matching is enabled by exposing $\forall(u : \mathbb{U})$ internally as a left adjoint.
- In a technical report [Nuy20], we investigate how the modalities introduced in this paper commute with each other, and with prior modalities (i.e. those already present before adding the transpension type). We also consider 2-cells arising from multiplier morphisms.

While MTT [Gra+20b; Gra+20a] satisfies canonicity³, decidable type-checking will at least require a computational understanding of the mode theory, and of some new typing rules that we add to MTT. For this reason, we build on extensional MTT [Gra+20a], and defer decidability and canonicity to future work.

Overview of the chapter In section 7.2, we demonstrate in a simple setting how the transpension resembles the suspension from HoTT and how it allows for higher-dimensional pattern matching. In section 7.3, we list a few results from

³The current state of affairs is that MTT extended with a single typing rule satisfies canonicity. That rule is not compatible with the model used in this paper, but Gratzner et al. are working on a canonicity theorem for MTT proper.

MTT. In section 7.4, we define the mode theory on which we will instantiate MTT and highlight some important modalities. The instantiation of MTT is the type system we are after, but it is next to unreadable for humans, so in section 7.5, we define an informal notation that is *much* less precise but more readable. In section 7.6, we supplement our MTT instance with a few specialized typing rules. In section 7.7, we investigate the structure of the transpension type. In section 7.8, we explain how to recover known internal presheaf operators. We conclude in section 7.9.

7.2 A Naïve Transpension Type

In this section, for purposes of demonstration, we present simplified typing rules for the transpension. Using these, we will already be able to exhibit the transpension as similar to a dependent version of the suspension in HoTT [Uni13]. Moreover, in order to showcase how the transpension type allows us to internalize the presheaf structure of other types, we will demonstrate a technique which we call higher-dimensional pattern matching and which has already been demonstrated by A. Pitts [Pit14] in nominal type theory using locally fresh names.

Typing rules To do this, we first present, in fig. 7.1, typing rules for the transpension type in a very simple setting: a type system with affine shape variables $u : \mathbb{U}$. Variables to the left of u are understood to be fresh for u ; variables introduced after u may be substituted with terms depending on u . In particular, we have no contraction $(w/u, w/v) : (w : \mathbb{U}) \rightarrow (u, v : \mathbb{U})$, while exchange $(x : A, u : \mathbb{U}) \rightarrow (u : \mathbb{U}, x : A)$ only works in one direction. This is enforced by the special substitution rules for shape variables.

The elimination rule for $f u$ requires that the function f be fresh for u , i.e. that f depend only on variables to the left of u [BCM15; Mou16].

Additionally, the system contains a transpension type $\check{\lambda} u. A$ over \mathbb{U} , with more unusual rules. When checking the type $\check{\lambda}(u : \mathbb{U}). A$ in context $(\Gamma, u : \mathbb{U}, y : B)$, the part A will be checked in a modified context [BV17; ND19a], where $y : B$ (which potentially depends on u) will change type, becoming a function $y|_{u=\sqcup} : \forall u. B$ that can be applied to $v : \mathbb{U}$ yielding $y|_{u=v} : B[v/u]$. In other words, we get hold of the dependency of y on u . The *meridian* constructor $\text{merid } u a$ is checked in a similar way, and is modelled by transposition for the adjunction $\forall u \dashv \check{\lambda} u$. We remark that both $\check{\lambda} u. A$ and $\text{merid } u a$ depend on u , whereas A and a do not, so in a way the transpension lifts data to a higher dimension, turning points into \mathbb{U} -cells. The elimination rule takes data again to a lower dimension: it turns a

Affine shape variables:

$$\frac{\Gamma \text{ctx}}{\Gamma, u : \mathbb{U} \text{ctx}} \quad \frac{\sigma : \Gamma \rightarrow \Gamma'}{(\sigma, u/v) : (\Gamma, u : \mathbb{U}) \rightarrow (\Gamma', v : \mathbb{U})} \quad \frac{\sigma : \Gamma \rightarrow \Gamma'}{\sigma : (\Gamma, u : \mathbb{U}) \rightarrow \Gamma'}$$

Affine function type:

$$\frac{\Gamma, u : \mathbb{U} \vdash A \text{ type}}{\Gamma \vdash \forall u. A \text{ type}} \quad \frac{\Gamma, u : \mathbb{U} \vdash a : A}{\Gamma \vdash \lambda u. a : \forall u. A} \quad \frac{\Gamma \vdash f : \forall u. A}{\Gamma, u : \mathbb{U}, \delta : \Delta \vdash f u : A}$$

Telescope quantification:

$$\begin{aligned} \forall u. () &= () \\ \forall u. (\delta : \Delta, x : A) &= (\forall u. (\delta : \Delta)), x|_{u=\sqcup} : \forall u. (A[\delta|_{u=\sqcup}/\delta]) \\ \forall u. (\delta : \Delta, v : \mathbb{U}) &= (\forall u. (\delta : \Delta)), v : \mathbb{U} \end{aligned}$$

Transpension type:

$$\frac{\Gamma, u : \mathbb{U} \vdash (\delta : \Delta) \text{ telescope} \quad \Gamma, \forall u. (\delta : \Delta) \vdash A \text{ type}}{\Gamma, u : \mathbb{U}, \delta : \Delta \vdash \check{\exists}(u : \mathbb{U}). A \text{ type}} \quad \frac{\Gamma, u : \mathbb{U} \vdash t : \check{\exists} u. A}{\Gamma \vdash \text{unmerid}(u.t) : A}$$

where $\text{unmerid}(u.\text{merid } u a) = a$

$$\frac{\Gamma, \forall u. (\delta : \Delta) \vdash a : A}{\Gamma, u : \mathbb{U}, \delta : \Delta \vdash \text{merid } u a : \check{\exists}(u : \mathbb{U}). A}$$

where $\text{merid } u (\text{unmerid}(v.t[v/u, \delta|_{u=v}/\delta])) = t$

Figure 7.1: Selection of typing rules for a naïve transpension type.

dependent \mathbb{U} -cell in the transpension into a point in A . This is the co-unit of the adjunction. The β - and η -rules internalize the adjunction laws.

These typing rules are sound in certain presheaf categories (those where \mathbb{U} is cancellative and affine, see definition 7.4.1), but are unsatisfactory in several respects. First, we have no story for substitutions which exist in cubical type systems such as $(0/i) : \Gamma \rightarrow (\Gamma, i : \mathbb{I})$ [BCM15; BCH14; Coh+17] or $(j \wedge k/i) : (\Gamma, j, k : \mathbb{I}) \rightarrow (\Gamma, i : \mathbb{I})$ [Coh+17], as there is no formation rule for $\check{\exists} 0.A$ or $\check{\exists}(j \wedge k).A$. Secondly, in non-affine generalizations, the transpension is not stable under substitution of the variables preceding u [Nuy20]. In order to obtain a better behaved type system, in the rest of the paper we will rely on MTT, which we briefly summarize in section 7.3.

Poles We can still try to get a grasp on $\check{\mathbb{I}}0.A$, however. In general we have $T[0/i] \cong (\forall i.(i = 0) \rightarrow T)$. For $T = \check{\mathbb{I}}i.A$, the latter type is inhabited by $\lambda i.\lambda e.\text{merid } i (\zeta e|_{i=1})$, since $e|_{i=1}$ proves $1 = 0$. Moreover, using the η -rule, we can show that this is the only element.

Thus we see that the transpension type essentially consists of meridians $(i : \mathbb{I}) \rightarrow \check{\mathbb{I}}i.T$ for all $t : T$ which are all equal when $i = 0$ or $i = 1$. This makes the transpension type quite reminiscent of a dependent version of the suspension type from HoTT [Uni13], although the quantification of the context is obviously a distinction.

Higher-dimensional pattern matching Given two types $A, B : \mathbb{U}$, higher-dimensional pattern matching allows us to construct a function $\Gamma \vdash f : (\forall u.A \uplus B) \rightarrow (\forall u.A) \uplus (\forall u.B)$. This function expresses that any \mathbb{U} -shaped cell in the coproduct type $A \uplus B$ must be either a cell in A or a cell in B . In that sense, it exposes the presheaf structure of the coproduct type $A \uplus B$. We can define f as follows (and generalization to $A, B : \forall u.\mathbb{U}$ is straightforward):

$$f \hat{c} = \text{unmerid} \left(u.\text{case } \hat{c} u \text{ of } \left\{ \begin{array}{ll} \text{inl } a & \mapsto \text{merid } u (\text{inl } (\lambda v.a|_{u=v})) \\ \text{inr } b & \mapsto \text{merid } u (\text{inr } (\lambda v.b|_{u=v})) \end{array} \right\} \right)$$

The argument to $\text{unmerid}(u.\sqcup)$ should be of type $\check{\mathbb{I}}u.((\forall u.A) \uplus (\forall u.B))$. Interestingly, since we have u in scope, we can apply $\hat{c} : \forall u.(A \uplus B)$ to it, and pattern match to decide which case we are in. Both cases are analogous; in the first case, a variable $a : A$ is brought in scope, so we are in context $(\Gamma, \hat{c} : \forall u.A \uplus B, u : \mathbb{U}, a : A)$. We then use the constructor $\text{merid } u \sqcup$, which again removes u from scope and turns $a : A$ into a function $a|_{u=\sqcup} : \forall u.A$. Then we trivially finish the proof by writing $\text{inl } (\lambda v.a|_{u=v})$, where we have η -expanded $a|_{u=\sqcup}$ mainly for facilitating further narrative. In summary, between unmerid and merid , we had temporary access to a variable $u : \mathbb{U}$ which allowed us to pattern-match on $\hat{c}u$. More conceptually, we can say that the transpension allows us to temporarily work with \hat{c} as if it were a lower-dimensional value of type $A \uplus B$. We will see in section 7.8 how similar ideas can be used to implement other internalization operators. Interestingly, this construction of f using the transpension also comes with suitable computational behaviour. When we evaluate $f(\lambda u.\text{inl}(\hat{a}u))$, then $\hat{a}u$ is substituted for a . Next, $(\hat{a}u)|_{u=v}$ simplifies to $\hat{a}v$, so we can η -contract $\lambda v.\hat{a}v$, and β -reduce unmerid , which yields $\text{inl } \hat{a}$ as expected.

7.3 A Snippet of MTT

We highlight some results about MTT [Gra+20b; Gra+20a] (section 5.3) that are relevant in the current paper.

Proposition 7.3.1. We have $\langle \text{id} \mid^* A \rangle \cong A$ and $\langle \nu \circ \mu \mid^{\text{nm}} A \rangle \cong \langle \nu \mid^n \langle \mu \mid^m A \rangle \rangle$.

Proposition 7.3.2. For any 2-cell $\alpha : \mu \Rightarrow \nu$, we have $\langle \mu \mid^m A \rangle \rightarrow \langle \nu \mid^n A[\alpha \downarrow_n^m] \rangle$.

Proposition 7.3.3. If $\kappa \dashv \mu$ internal to the mode theory, there is a function $\text{prmod}_\mu : (\kappa \dashv^{\text{t}} \langle \mu \mid^m A \rangle) \rightarrow A[\varepsilon \downarrow_*^{\text{tm}}]$, satisfying a β - and (thanks to extensionality) an η -law. Combined with these rules, prmod_μ is equally expressive as the let-eliminator for $\langle \mu \mid \sqcup \rangle$.

Proposition 7.3.4. If $\kappa \dashv \mu$ internal to the mode theory, there is an isomorphism of contexts $\sigma = (x \eta \downarrow_{m\ell} / y \downarrow_\ell) : (\Gamma, x : A, \mathbf{\bullet}_\mu^m) \cong (\Gamma, \mathbf{\bullet}_\mu^m, \kappa \dashv y :^{\text{t}} A[\eta \downarrow_{m\ell}^*])$ with inverse $\sigma^{-1} = (\text{id}_\Gamma, \eta \downarrow_{m\ell}^*, y \downarrow_\ell / x \downarrow_*, \downarrow_{m\ell}^m) \circ (\text{id}_{(\Gamma, \mathbf{\bullet}_\mu^m, y)}, \varepsilon \downarrow_*^{\text{tm}'})$. Correspondingly, given B in the latter context, there is an isomorphism of types $((x : A) \rightarrow \langle \mu \mid^m B[\sigma] \rangle) \cong \langle \mu \mid^m (\kappa \dashv y :^{\text{t}} A[\eta \downarrow_{m\ell}^*]) \rightarrow B \rangle$.

7.4 A Mode Theory for Shape (Co)quantification

For space reasons, we assume that there are no prior modalities, i.e. that the type system to which we wish to add a transpension type is non-modal in the sense that it has a single mode and only the identity modality. Prior modalities are considered in the technical report [Nuy20]. We assume that this single prior mode is modelled by the presheaf category $\text{Psh}(\mathcal{W})$.

7.4.1 Shape Contexts

A first complication is that the modalities $\Omega u \dashv \Pi u \dashv \wp u$ all bind or depend on a variable, a phenomenon which is not supported by MTT. However, the following trick solves this problem. Assume we have in the prior system a context Ξ modelled by a presheaf over \mathcal{W} . Then the presheaves $\text{Psh}(\mathcal{W}/\Xi)$ over the category of elements of the presheaf Ξ are also a model of dependent type theory. Denoting the judgements of the latter system with a prefix $\Xi \mid$, it happens to be the case that judgements $\Xi \mid \Gamma \vdash J$ (i.e. $\Gamma \vdash J$ in $\text{Psh}(\mathcal{W}/\Xi)$) have precisely the same meaning as judgements $\Xi, \Gamma \vdash J$ in $\text{Psh}(\mathcal{W})$ (for a suitable but straightforward translation of J). Thus, we will group together all shape variables (variables for which we want a transpension type) in a **shape**

context Ξ in front of the typing context. Our judgements will then take the form $\Xi \mid \Gamma \vdash J$. This allows us to frame Ξ as the *mode* of the judgement.

7.4.2 Mode Theory

For simplicity, we take a highly general mode theory and will then only be able to say interesting things about specific modes, modalities and 2-cells. In practice, and especially in implementations, one will want to select a more syntactic subtheory right away.

As **modes**, we take the set of all small presheaves over \mathcal{W} , which we think of as **shape contexts**. The mode Ξ is modelled in $\mathbf{Psh}(\mathcal{W}/\Xi)$. As **modalities** $\mu : \Xi_1 \rightarrow \Xi_2$, we take all functors $\llbracket \blacklozenge_\mu \rrbracket : \mathbf{Psh}(\mathcal{W}/\Xi_2) \rightarrow \mathbf{Psh}(\mathcal{W}/\Xi_1)$ which have a right adjoint μ that is then automatically a weak CwF morphism [Nuy20] and gives rise to a DRA [Bir+20; Nuy18a]. As **2-cells** $\alpha : \mu \Rightarrow \nu$, we take all natural transformations.

7.4.3 Shapes and Multipliers

We now proceed by highlighting some interesting modes, modalities, and 2-cells. To begin with, we fix a collection of shapes. We associate to each shape \mathbb{U} a functor $\sqsubset \times U : \mathcal{W} \rightarrow \mathcal{W}$ which extends to a functor $\sqsubset \times \mathbf{y}U : \mathbf{Psh}(\mathcal{W}) \rightarrow \mathbf{Psh}(\mathcal{W})$.⁴ Internally, we will use shape variables to increase human readability and to reduce the heaviness of notation for shape substitutions, writing $(\Xi, u : \mathbb{U})$ for the shape context $\Xi \times \mathbf{y}U$. However, in a fully elaborate syntax these variables would be redundant.

Of course, if we model shape context extension with $u : \mathbb{U}$ by an *arbitrary* functor, then we will not be able to prove many results. Depending on the properties of the functor, the variable u will behave like an affine one or like a cartesian one (or perhaps neither) and the Φ -combinator will or will not be sound for \mathbb{U} . For this reason, we introduce some criteria that help us classify shapes:

Definition 7.4.1. Assume \mathcal{W} has a terminal object \top . A **multiplier** for an object U is a functor $\sqsubset \times U : \mathcal{W} \rightarrow \mathcal{W}$ such that $\top \times U \cong U$.⁵ This gives us a natural second projection $\pi_2 : (\sqsubset \times U) \rightarrow U$. We define the **fresh weakening functor** to the slice category as $\perp_U : \mathcal{W} \rightarrow \mathcal{W}/U : W \mapsto (W \times U, \pi_2)$. We say that a multiplier (as well as its shape) is:

⁴Both $\sqsubset \times U$ and $\sqsubset \times \mathbf{y}U$ are to be regarded as single-character symbols, i.e. \times in itself is meaningless.

⁵In the technical report [Nuy20], we generalize beyond endofunctors.

- **Semicartesian** if it is copointed, i.e. has a first projection $\pi_1 : (\sqcup \times U) \rightarrow \text{Id}$,
- **Cartesian** if it is naturally isomorphic to the cartesian product with U ,
- **Cancellative** if \perp_U is faithful,
- **Affine** if \perp_U is full,
- **Connection-free** if \perp_U is essentially surjective on objects (V, ψ) such that ψ is dimensionally split (definition 7.4.3),
- **Quantifiable** if \perp_U has a left adjoint $\exists_U : \mathcal{W}/U \rightarrow \mathcal{W}$.

Variables of shape \mathbb{U} admit weakening if and only if the multiplier is semicartesian, and exchange and contraction if (but not only if) it is cartesian. Weakening, exchange and contraction are all shape substitutions, which will be internalized as modalities.

Proposition 7.4.2. If a multiplier is affine and cartesian, then it is the identity functor. [Nuy20]

Definition 7.4.3. A morphism $\psi : V \rightarrow U$ is called **dimensionally split** (w.r.t. $\sqcup \times U$) if there is some W such that $\pi_2 : W \times U \rightarrow U$ factors over ψ . We define the **boundary** ∂U as the subpresheaf of the Yoneda-embedding $\mathbf{y}U$ consisting of those morphisms that are *not* dimensionally split, and we define $\sqcup \times \partial U$ by pullback. We also write $(\Xi, u : \partial \mathbb{U})$ for $\Xi \times \partial U$.

In most popular base categories, all morphisms to \top are split epi. Being dimensionally split is then equivalent to being split epi. We call a category **spooky** if some morphism to \top is not split epi. The notion of dimensionally split morphisms lets us consider the boundary and connection-freedom (a requirement for modelling Φ) also in spooky base categories, where the output of \perp_U may not be split epi.

Examples Let us look at some examples of multipliers. Their properties are listed in fig. 7.2. Most properties are easy to verify, so we omit the proofs.

Example 7.4.4 (Identity). The identity functor on an arbitrary category \mathcal{W} is an endomultiplier for \top . It is quantifiable, with $\exists_\top : \mathcal{W}/\top \rightarrow \mathcal{W} : (W, ()) \mapsto W$.

Example 7.4.5 (Cartesian product). Let \mathcal{W} be a category with finite products and $U \in \mathcal{W}$. Then $\sqcup \times U$ is an endomultiplier for U , which is affine if and only if U is non-terminal (proposition 7.4.2, example 7.4.4). It is quantifiable with $\exists_U : \mathcal{W}/U \rightarrow \mathcal{W} : (W, \psi) \mapsto W$. Hence, we have $\exists_U \perp_U = \sqcup \times U$.

Example	Base category	Multiplier	Spooky category	Wkn. (semicartesian)	Exchange	Contraction	Cartesian	Cancellative	Affine	Connection-free	Quantifiable
7.4.4	\mathcal{W}	Id	?	✓	✓	✓	✓	✓	✓	✓	✓
7.4.5	\mathcal{W}	$\sqcup \times U$?	✓	✓	✓	✓	?	✓	?	✓
7.4.6	${}^k\text{Cube}_{\square}$	$\sqcup * (i : \mathbb{I})$	$k = 0$	✓	✓	✗	✗	✓	✓	✓	✓
7.4.7	${}^k\text{Cube}$	$\sqcup \times (i : \mathbb{I})$	$k = 0$	✓	✓	✓	✓	✓	✗	✓	✓
7.4.8	CCHM	$\sqcup \times (i : \mathbb{I})$	✗	✓	✓	✓	✓	✓	✗	✗	✓
7.4.9	DCube_d	$\sqcup \times \langle d \rangle$	✗	✓	✓	✓	✓	✓	✗	✓	✓
7.4.10	Clock	$\sqcup \times (i : \odot_k)$	✓	✓	✓	✓	✓	✓	✓	✓	✓
7.4.11	TwCube	$\sqcup \times \mathbb{I}$	✗	✗	✗	✗	✗	✓	✓	✓	✓
7.4.12	n	$\min(\sqcup, i)$	✓	✓	✓	✓	✓	✓	✗	✓	✓
7.4.13	Simplex	$\sqcup \uplus < [0]$	✗	✓	✗	✓	✗	✓	✓	✗	✓

Figure 7.2: Some interesting multipliers and their properties.

Example 7.4.6 (Affine cubes). Let ${}^k\text{Cube}_{\square}$ be the category of affine k -ary cubes as used in [BCH14] (binary) or [BCM15] (unary) (example 2.3.11). This category is spooky if and only if $k = 0$. Consider the functor $\sqcup * (i : \mathbb{I}) : {}^k\text{Cube}_{\square} \rightarrow {}^k\text{Cube}_{\square} : W \mapsto (W, i : \mathbb{I})$, which is a multiplier for $(i : \mathbb{I})$. This functor is quantifiable with $\exists_{(i:\mathbb{I})}((W, j : \mathbb{I}), (j/i)) = W$ and $\exists_{(i:\mathbb{I})}(W, (\varepsilon/i)) = W$ for each of the k endpoints ε .

In the nullary case, ${}^0\text{Cube}_{\square}$ is the base category of the Schanuel topos, which is equivalent to the category of nominal sets [Pit13]. In that case, $\exists_{(i:\mathbb{I})}$ is not just left adjoint to $\exists_{(i:\mathbb{I})}$, but in fact an inverse and hence also right adjoint. This is in line with the fact that in nominal type theory [PMD15], there is a single name quantifier which can be read as either existential or universal quantification.

Example 7.4.7 (Cartesian cubes). We also consider $\sqcup \times (i : \mathbb{I})$ in the category ${}^k\text{Cube}$ of cartesian k -ary cubes (example 2.3.10), which is an instance of example 7.4.5.

Example 7.4.8 (CCHM cubes). Idem for CCHM (example 2.3.12). Connection-freedom is now violated by $(j \vee k/i), (j \wedge k/i) : (j : \mathbb{I}, k : \mathbb{I}) \rightarrow (i : \mathbb{I})$.

Example 7.4.9 (Depth d cubes). We consider $\sqcup \times (i : \langle k \rangle)$ in the category DCube_d of depth d cubical sets (example 2.3.13), which is an instance of example 7.4.5.

Example 7.4.10 (Clocks). Idem for $\sqcup \times (i : \odot_k)$ in the category Clock (example 2.3.16).

Example 7.4.11 (Twisted cubes). Pinyo and Kraus’s category of twisted cubes TwCube [PK19] can be described as a subcategory of the category of linear orders, and indeed a subcategory of the category of simplices Simplex . On these categories, we can define a functor $\sqcup \times \mathbb{I}$ such that $W \times \mathbb{I} = W^{\text{op}} \uplus_{<} W$, where we consider elements from the left smaller than those from the right. Now TwCube is the subcategory of Simplex whose objects are generated by \top and $\sqcup \times \mathbb{I}$ (note that every object then also has an opposite since $\top^{\text{op}} = \top$ and $(V \times \mathbb{I})^{\text{op}} \cong V \times \mathbb{I}$), and whose morphisms are given by

- $(\varphi, 0) : \text{Hom}_{\text{TwCube}}(V, W \times \mathbb{I})$ for all $\varphi : \text{Hom}_{\text{TwCube}}(V, W^{\text{op}})$,
- $(\varphi, 1) : \text{Hom}_{\text{TwCube}}(V, W \times \mathbb{I})$ for all $\varphi : \text{Hom}_{\text{TwCube}}(V, W)$,
- $\varphi \times \mathbb{I} : \text{Hom}_{\text{TwCube}}(V \times \mathbb{I}, W \times \mathbb{I})$ for all $\varphi : \text{Hom}_{\text{TwCube}}(V, W)$,
- $() : \text{Hom}_{\text{TwCube}}(V, \top)$.

Note that this collection automatically contains all identities, composites, and opposites. Isomorphism to Pinyo and Kraus’s category of twisted cubes can be seen from their ternary representation [PK19, def. 34]. We now consider the multiplier $\sqcup \times \mathbb{I} : \text{TwCube} \rightarrow \text{TwCube}$, which Pinyo and Kraus call the twisted prism functor. It is quantifiable with

$$\exists_{\mathbb{I}} : \begin{cases} (W, ((), 0)) & \mapsto W^{\text{op}} \\ (W, ((), 1)) & \mapsto W \\ (W \times \mathbb{I}, () \times \mathbb{I}) & \mapsto W, \end{cases} \tag{7.1}$$

with the obvious action on morphisms.

Example 7.4.12 (Finite ordinals). In the base category ω of the topos of trees (example 2.3.14), a cartesian product is given by $i \times j = \min(i, j)$. However, this category lacks a terminal object. Instead, on the subcategory n , which is endowed with the same cartesian product, we consider the multiplier $\sqcup \times i$, which is again an instance of example 7.4.5.

Example 7.4.13 (Simplices). In the category of simplices Simplex (example 2.3.9), we consider the functor $\sqcup \times [1] := \sqcup \uplus_{<} [0]$, which adds a maximal element to a linear order. This is a multiplier for $[1]$ and is quantifiable with $\exists_{[1]}(W, \psi) = \psi^{-1}(0)$.

7.4.4 Modalities for Substitution

A substitution $\sigma : \Xi_1 \rightarrow \Xi_2$ gives rise to a functor $\Sigma/\sigma : \mathcal{W}/\Xi_1 \rightarrow \mathcal{W}/\Xi_2$ and hence [StaVC] to a triple of adjoint functors $\Sigma \sigma \dashv \Omega \sigma \dashv \Pi \sigma$ between the presheaf categories, where $\Omega \sigma$ has the exact same semantics as ordinary substitution. The two functors $\Omega \sigma$ and $\Pi \sigma$ give rise to DRAs and can be internalized as MTT modalities. We denote these as⁶ $\Omega \sigma$ or $\Omega(\xi_1 : \Xi_1, \xi_2 = \xi_2[\sigma]) : (\xi_2 : \Xi_2) \rightarrow (\xi_1 : \Xi_1)$ and as $\Pi \sigma$ or $\Pi(\xi_1 : \Xi_1, \xi_2 = \xi_2[\sigma]) : (\xi_1 : \Xi_1) \rightarrow (\xi_2 : \Xi_2)$. For example, in cubical type theory, we get $\Omega(i = 0) : (\Xi, i : \mathbb{I}) \rightarrow \Xi$ with right adjoint $\Pi(i = 0)$, and for semicartesian \mathbb{U} , we get $\Omega(u : \mathbb{U}) : \Xi \rightarrow (\Xi, u : \mathbb{U})$ with right adjoint $\Pi(u : \mathbb{U})$. The Π -modality is strictly functorial, whereas the Ω -modality is pseudofunctorial: we need explicit 2-cells witnessing $\Omega \tau \circ \Omega \sigma \cong \Omega(\tau \circ \sigma)$.⁷ These modalities are adjoint internally by virtue of the 2-cells for the unit $\text{const}_\sigma : \text{id} \Rightarrow \Pi \sigma \circ \Omega \sigma$ and co-unit $\text{app}_\sigma : \Omega \sigma \circ \Pi \sigma \Rightarrow \text{id}$. If σ introduces variables, then the codomain of the co-unit may be a variable renaming that is semantically the identity, e.g. $\text{app}_{(v/u:\mathbb{U})} : \Omega(v : \mathbb{U}) \circ \Pi(u : \mathbb{U}) \Rightarrow \Omega(v : \mathbb{U}, u = v)$.

Example 7.4.14. If \mathbb{U} is cartesian, then there is a diagonal substitution $(w/u, w/v) : (\Xi, w : \mathbb{U}) \rightarrow (\Xi, u, v : \mathbb{U})$. Writing

$$\alpha = \text{id}_{\Pi u} \star \text{id}_{\Pi v} \star \text{const}_{(w,u=w,v=w)},$$

this allows us to type the naively typed function $\lambda f.\lambda w.f w w : (\Pi u.\Pi v.A) \rightarrow \Pi w.A[w/u, w/v]$ as

$$\begin{aligned} & \langle \Pi(u : \mathbb{U}) \mid^{p_u} \langle \Pi(v : \mathbb{U}) \mid^{p_v} A \rangle \rangle \\ & \rightarrow \langle \Pi(w : \mathbb{U}) \mid^{p_w} \langle \Omega(w : \mathbb{U}, u = w, v = w) \mid^o A[\alpha \downarrow_{p_w}^{p_u p_v}] \rangle \rangle. \end{aligned}$$

Remark 7.4.15. The reframing of shape substitutions as a modality, has the annoying consequence that substitution no longer reduces. However, both $\langle \Omega \sigma \mid \sqsubset \rangle$ and $\text{mod}_{\Omega \sigma}$ are semantically an ordinary substitution.⁸ Thus, we could add computation rules such as:

$$\begin{aligned} \langle \Omega \sigma \mid^o A \times B \rangle &= \langle \Omega \sigma \mid^o A \rangle \times \langle \Omega \sigma \mid^o B \rangle, & \langle \Omega \sigma \mid^o \mathbb{U} \rangle &= \mathbb{U}, \\ \text{mod}_{\Omega \sigma}^o(a, b) &= (\text{mod}_{\Omega \sigma}^o a, \text{mod}_{\Omega \sigma}^o b), & \text{mod}_{\Omega \sigma}^o A &= \langle \Omega \sigma \mid^o A \rangle. \end{aligned}$$

This is fine in an extensional type system, but would not play well with the β -rule for modal types in an intensional system.

⁶By the second notation, we mean that we declare new variables (with their type if there is space) and write $u = t$ when we substitute t for u .

⁷This is because we defined the modality μ via $\llbracket \mu \rrbracket$ and only Ω is strictly functorial in the model. However, Gratzner et al. [Gra+20a] have a strictification theorem for models of MTT which can strictify the isomorphism.

⁸Not along $\sigma : \Xi_1 \rightarrow \Xi_2$, but along $\sigma.\eta_{\Sigma \sigma \dashv \Omega \sigma} : \Xi_1.\Gamma \cong \Xi_2.\Sigma \sigma \Gamma$, which happens to be an isomorphism.

7.4.5 Modalities for (Co)quantification

The fresh weakening functor $\downarrow_U : \mathcal{W} \rightarrow \mathcal{W}/U$ generalizes to a functor $\downarrow_U^{\Xi} : \mathcal{W}/\Xi \rightarrow \mathcal{W}/(\Xi \times \mathbf{y}U)$ between categories of elements. Assuming that the multiplier is quantifiable,⁹ there is a left adjoint $\exists_U^{\Xi} \dashv \downarrow_U^{\Xi}$ [Nuy20]. These two give rise to four adjoint functors $\exists(u : \mathbb{U}) \dashv \downarrow(u : \mathbb{U}) \dashv \forall(u : \mathbb{U}) \dashv \check{\exists}(u : \mathbb{U})$ between the presheaf categories. The latter three give rise to DRAs and can be internalized as MTT modalities. We denote the units and co-units as

$$\begin{aligned} \text{const}_{(u:\mathbb{U})} : \text{id} &\Rightarrow \forall(u : \mathbb{U}) \circ \downarrow(u : \mathbb{U}) & \text{app}_{(u:\mathbb{U})} : \downarrow(u : \mathbb{U}) \circ \forall(u : \mathbb{U}) &\Rightarrow \text{id} \\ \text{reid}_{(u:\mathbb{U})} : \text{id} &\Rightarrow \check{\exists}(u : \mathbb{U}) \circ \forall(u : \mathbb{U}) & \text{unmerid}_{(u:\mathbb{U})} : \forall(u : \mathbb{U}) \circ \check{\exists}(u : \mathbb{U}) &\Rightarrow \text{id} \end{aligned}$$

Again, we also write $\text{app}_{(v/u:\mathbb{U})} : \downarrow(v : \mathbb{U}) \circ \forall(u : \mathbb{U}) \Rightarrow \Omega(v : \mathbb{U}, u = v)$ and $\text{reid}_{(v/u:\mathbb{U})} : \Omega(v : \mathbb{U}, u = v) \Rightarrow \check{\exists}(v : \mathbb{U}) \circ \forall(u : \mathbb{U})$ to handle shape variable renamings that are semantically the identity.

Theorem 7.4.16 (Quantification). [Nuy20] If the multiplier is

- cancellative and affine, then $\text{const}_{(u:\mathbb{U})}$ and $\text{unmerid}_{(u:\mathbb{U})}$ are natural isomorphisms,
- semi-cartesian (so that $\Omega(u : \mathbb{U})$ and $\Pi(u : \mathbb{U})$ exist), then we have $\text{spoil}_{(u:\mathbb{U})} : \downarrow(u : \mathbb{U}) \Rightarrow \Omega(u : \mathbb{U})$ and $\text{cospoil}_{(u:\mathbb{U})} : \Pi(u : \mathbb{U}) \Rightarrow \forall(u : \mathbb{U})$,
- cartesian, then we can soundly identify $\downarrow(u : \mathbb{U}) = \Omega(u : \mathbb{U})$ and $\forall(u : \mathbb{U}) = \Pi(u : \mathbb{U})$.

To understand the difference between $\downarrow(u : \mathbb{U})$ and $\Omega(u : \mathbb{U})$, we can compare to the naïve system (section 7.2) or Moulin et al.’s system [BCM15; Mou16]. There, shape variables $u : \mathbb{U}$ are part of the context, and variables to the left of u are fresh for u . Here, shape variables are all in the shape context, but we use $\downarrow u$ and Ωu (as well as the semantically identical $\blacktriangleleft_{\forall u}$ and $\blacktriangleleft_{\Pi u}$) to keep track of whether or not other variables should be fresh for u . Thus, in terms of the naïve system, $\forall(u : \mathbb{U})$ introduces u at the end of Γ (marking all variables fresh with $\blacktriangleleft_{\forall u}$), and $\Pi(u : \mathbb{U})$ introduces u in front of Γ (marking them non-fresh with $\blacktriangleleft_{\Pi u}$). The naïve system allows exchanging shape and other variables in one direction only, which is represented here by the generally non-invertible 2-cells spoil and cospoil . We emphasize that the word ‘fresh’ needs to be taken with a grain of salt: only for cancellative and affine \mathbb{U} does invertibility of const_u guarantee that something fresh for u does not depend on u .

Example 7.4.17. As an instance of proposition 7.3.3, we obtain $\text{prmod}_{\forall(v/u:\mathbb{U})} : (\downarrow(v : \mathbb{U}) \dashv \langle \forall(u : \mathbb{U}) \mid^{\mathfrak{a}} A \rangle) \rightarrow \langle \Omega(v : \mathbb{U}, u = v) \mid^{\circ} A[\text{app}_{v/u} \downarrow_{\bullet}^{\mathfrak{f}\mathfrak{a}}] \rangle$, which says

⁹We have not encountered any interesting examples where this is not the case.

that a non-cartesian function g with naïve type $g : \forall u. A$ can be applied to $v : \mathbb{U}$ provided that g is fresh for v .

Example 7.4.18 (Higher-dimensional pattern matching). As in section 7.2, we create a function $f : \langle \forall (u : \mathbb{U}) \mid^a A \uplus B \rangle \rightarrow \langle \forall (u : \mathbb{U}) \mid^a A \rangle \uplus \langle \forall (u : \mathbb{U}) \mid^a B \rangle$, namely:

$$f \hat{c} = \text{prmod}_{\hat{\exists} u} \cdot^a \text{case} (\text{prmod}_{\forall u} \cdot^o (\hat{c} \text{const}_u \downarrow_{\alpha_0})) \text{ of}$$

$$\left\{ \begin{array}{l} \text{inl } a \mapsto \text{mod}_{\hat{\exists} u}^{\downarrow} (\text{inl} (\text{mod}_{\forall u}^{\uparrow} (a \text{reid}_{x_u} \downarrow_{\alpha'}))) \\ \text{inr } b \mapsto \text{mod}_{\hat{\exists} u}^{\downarrow} (\text{inr} (\text{mod}_{\forall u}^{\uparrow} (b \text{reid}_{x_u} \downarrow_{\alpha'}))) \end{array} \right\}.$$

We see that $\text{mod}_{\forall u}$ and $\text{prmod}_{\forall u}$ correspond to shape abstraction and application in the naïve system, $\text{prmod}_{\hat{\exists} u}$ to unmerid , and $\text{mod}_{\hat{\exists} u}$ to merid . The annotation const_u is an explicit weakening over $u : \mathbb{U}$, whereas reid_{x_u} replaces an explicit reindexing $|_{u=u}$.

7.5 An Informal Notation for Human Readers

At this point, we can instantiate MTT with the mode theory defined in the previous section 7.4. This is the type system that we will be working in, but direct usage of the notations from MTT (section 5.3) will be a bit obscure. For example, in section 7.2 we denoted quantification of a context using \forall . In the MTT instance, we will instead use a lock for the right adjoint to \forall , i.e. we will write $\mathbf{\Delta}_{\hat{\exists} u}$. Similarly, shape abstraction will not be denoted using a λ , but instead as $\text{mod}_{\forall u} a$, and application will be denoted as $\text{prmod}_{\forall u} f$.

We will actually use these notations, with the intention of making clear which MTT concepts we are using, because using an instance of MTT is precisely our solution to the syntactic complications involved in creating a syntax more similar to that of section 7.2. However, the MTT notation gets in the way of intuition and even readability. For this reason, we additionally introduce an alternative notation, which is purely informal (MTT being the formal notation).

Figure 7.3 lists the rule WDRA:INTRO in formal and informal notation for each of the available modalities, thus implicitly listing the informal notations for locks (LOCK), modal types (weak DRAs, WDRA) and their introduction rules (WDRA:INTRO). The introduction rule for $\forall (u : \mathbb{U}). A$ is in line with the abstraction rule in fig. 7.1: the premise's context is extended with a variable u , and all other variables are taken to be fresh for u . In the naïve system, we expressed this by putting u to the right of Γ ; here, we wrap Γ in $\perp u$.

The modality $\Pi(i = 0)$ abstracts over the assumption that $i = 0$. So inside it, we get to use that assumption. By consequence, we no longer need i as it just

Introduction rules for modal types:

$\Xi \mid \Gamma, \mathbf{\exists}_{\exists(u:\mathbb{U})}^f \vdash a : A$	\rightsquigarrow	$\Xi \mid \exists(u : \mathbb{U}).\Gamma \vdash a : A$
$\Xi, u : \mathbb{U} \mid \Gamma \vdash \mathbf{mod}_{\exists u}^f a : \langle \exists u \mid^f A \rangle$	\rightsquigarrow	$\Xi, u : \mathbb{U} \mid \Gamma \vdash \mathbf{fresh} u a : \exists u.A$
$\Xi \mid \Gamma, \mathbf{\exists}_{\Omega(u:\mathbb{U})}^o \vdash a : A$	\rightsquigarrow	$\Xi \mid \Sigma(u : \mathbb{U}).\Gamma \vdash a : A$
$\Xi, u : \mathbb{U} \mid \Gamma \vdash \mathbf{mod}_{\Omega u}^o a : \langle \Omega u \mid^o A \rangle$	\rightsquigarrow	$\Xi, u : \mathbb{U} \mid \Gamma \vdash \mathbf{wkn} u a : \Omega u.A$
$\Xi, i : \mathbb{I} \mid \Gamma, \mathbf{\exists}_{\Omega(i=0)}^o \vdash a : A$	\rightsquigarrow	$\Xi, i : \mathbb{I} \mid \Sigma(i = 0).\Gamma \vdash a : A$
$\Xi \mid \Gamma \vdash \mathbf{mod}_{\Omega(i=0)}^o a : \langle \Omega(i = 0) \mid^o A \rangle$	\rightsquigarrow	$\Xi \mid \Gamma \vdash \mathbf{wkn} (i = 0) a : \Omega(i = 0).A$
$\Xi, u : \mathbb{U} \mid \Gamma, \mathbf{\exists}_{\forall u}^o \vdash a : A$	\rightsquigarrow	$\Xi, u : \mathbb{U} \mid \exists u.\Gamma \vdash a : A$
$\Xi \mid \Gamma \vdash \mathbf{mod}_{\forall(u:\mathbb{U})}^o a : \langle \forall(u : \mathbb{U}) \mid^o A \rangle$	\rightsquigarrow	$\Xi \mid \Gamma \vdash \lambda u.a : \forall(u : \mathbb{U}).A$
$\Xi, u : \mathbb{U} \mid \Gamma, \mathbf{\exists}_{\Pi u}^p \vdash a : A$	\rightsquigarrow	$\Xi, u : \mathbb{U} \mid \Omega u.\Gamma \vdash a : A$
$\Xi \mid \Gamma \vdash \mathbf{mod}_{\Pi(u:\mathbb{U})}^p a : \langle \Pi(u : \mathbb{U}) \mid^p A \rangle$	\rightsquigarrow	$\Xi \mid \Gamma \vdash \lambda u.a : \Pi(u : \mathbb{U}).A$
$\Xi \mid \Gamma, \mathbf{\exists}_{\Pi(i=0)}^p \vdash a : A$	\rightsquigarrow	$\Xi \mid \Omega(i = 0).\Gamma \vdash a : A$
$\Xi, i : \mathbb{I} \mid \Gamma \vdash \mathbf{mod}_{\Pi(i=0)}^p a$ $\quad : \langle \Pi(i = 0) \mid^p A \rangle$	\rightsquigarrow	$\Xi, i : \mathbb{I} \mid \Gamma \vdash \lambda_.a : \Pi(i = 0).A$
$\Xi \mid \Gamma, \mathbf{\exists}_{\wp(u:\mathbb{U})}^t \vdash a : A$	\rightsquigarrow	$\Xi \mid \forall(u : \mathbb{U}).\Gamma \vdash a : A$
$\Xi, u : \mathbb{U} \mid \Gamma \vdash \mathbf{mod}_{\wp u}^t a : \langle \wp u \mid^t A \rangle$	\rightsquigarrow	$\Xi, u : \mathbb{U} \mid \Gamma \vdash \mathbf{merid} u a : \wp u.A$

Simplifications based on the quantification theorem 7.4.16:

Cancellative and affine multipliers:

- Write $\Gamma, \exists(u : \mathbb{U}).\Delta$ instead of $\exists(u : \mathbb{U}).(\exists u.\Gamma, \Delta)$,
- Write $\Gamma, \forall(u : \mathbb{U}).\Delta$ instead of $\forall(u : \mathbb{U}).(\exists u.\Gamma, \Delta)$.

Cartesian multipliers:

- Write Σ, Ω, Π instead of $\exists, \exists, \forall$.

Usage of parentheses:

(Co)quantifiers range until the next comma, e.g. $\forall(u : \mathbb{U}).\Gamma, x : A$ means $(\forall(u : \mathbb{U}).\Gamma), x : A$.

Figure 7.3: Informal notation for locks and modal types.

means 0, so it disappears from the context. Its left adjoint $\Omega(i = 0)$ does the converse: when entering that modality, we define i to be 0 and subsequently forget that $i = 0$. Thus, within $\Omega(i = 0)$, we have i inscope; outside it, we do not, as we can just use 0. The quantifier $\Sigma(i = 0)$ retains this information so that Γ still makes sense, but we do not have syntactic access to that information anymore.

It should be noted that, since Σ does not preserve the empty context, Ω has a different meaning when applied to contexts or to types. When applied to a context, it is an actual substitution of the shape context. When applied to a type, it is semantically still a substitution, but between two isomorphic shape-and-type contexts $(\Xi, u : \mathbb{U}).\Gamma$ and $\Xi.(\Sigma(u : \mathbb{U}).\Gamma)$. A similar remark holds for $\exists(u : \mathbb{U})$ and $\exists u$, with the additional note that when we apply $\exists u$ to a type, we transfer the responsibility of asserting freshness for u from the context (which hid u away via $\exists u$) to the type.

For cancellative and affine multipliers, we invoke the quantification theorem 7.4.16 to extract the fresh part of the context from the quantifier. Then the introduction rule for $\checkmark u.A$ is in line with the meridian rule in fig. 7.1: u disappears from the context and the non-fresh part of the context (in the naïve system: the part to the right of u) is quantified over:

$$\frac{\Xi \mid \Gamma, \mathbf{\blacktriangle}_{\forall u}^{\mathfrak{a}}, \Delta, \mathbf{\blacktriangle}_{\checkmark(u:\mathbb{U})}^{\mathfrak{t}} \vdash a : A}{\Xi, u : \mathbb{U} \mid \Gamma, \mathbf{\blacktriangle}_{\forall u}^{\mathfrak{a}}, \Delta \vdash \text{mod}_{\checkmark u}^{\mathfrak{t}} a : \langle \checkmark u \mid^{\mathfrak{t}} A \rangle} \sim \frac{\Xi \mid \Gamma, \forall(u : \mathbb{U}).\Delta \vdash a : A}{\Xi, u : \mathbb{U} \mid \exists u.\Gamma, \Delta \vdash \text{merid } u a : \checkmark u.A}$$

The introduction rule for $\exists u.A$ looks similar, but uses existential quantification:

$$\frac{\Xi \mid \Gamma, \mathbf{\blacktriangle}_{\forall u}^{\mathfrak{a}}, \Delta, \mathbf{\blacktriangle}_{\exists(u:\mathbb{U})}^{\mathfrak{f}} \vdash a : A}{\Xi, u : \mathbb{U} \mid \Gamma, \mathbf{\blacktriangle}_{\forall u}^{\mathfrak{a}}, \Delta \vdash \text{mod}_{\exists u}^{\mathfrak{f}} a : \langle \exists u \mid^{\mathfrak{f}} A \rangle} \sim \frac{\Xi \mid \Gamma, \exists(u : \mathbb{U}).\Delta \vdash a : A}{\Xi, u : \mathbb{U} \mid \exists u.\Gamma, \Delta \vdash \text{fresh } u a : \exists u.A}$$

The existential quantifier makes the variables in Δ syntactically unavailable (unless they bear a modal annotation $\exists u$), so a can really only depend on the fresh parts of the context of $\text{fresh } u a$.

For cartesian multipliers, we pretend that the isomorphisms given in the quantification theorem 7.4.16 are equalities to further simplify the notation.

Figure 7.4 lists the informal notation for modal function application and for projections out of the modal types (proposition 7.3.3). This is not particularly

Modal function application:

$$\begin{array}{lll}
f \cdot \overset{\exists}{\exists} u a \rightsquigarrow f a & f \cdot \overset{\forall}{\forall} u a \rightsquigarrow f(u.a) & f \cdot \overset{\exists}{\exists} u a \rightsquigarrow f a \\
f \cdot \overset{\Omega}{\Omega} u a \rightsquigarrow f a & f \cdot \overset{\Pi}{\Pi} u a \rightsquigarrow f(u.a) & \\
f \cdot \overset{\Omega}{\Omega}(i=0) a \rightsquigarrow f(i.a) & f \cdot \overset{\Pi}{\Pi}(i=0) a \rightsquigarrow f(_ . a) &
\end{array}$$

Modal projections:

$$\begin{array}{ll}
\text{prmod}_{\forall u} \cdot \overset{\exists}{\exists} u f \rightsquigarrow f u & \text{prmod}_{\exists u} \cdot \overset{\forall}{\forall} u t \rightsquigarrow \text{unmerid}(u.t) \\
\text{prmod}_{\Pi u} \cdot \overset{\Omega}{\Omega} u f \rightsquigarrow f u & \\
\text{prmod}_{\Pi(i=0)} \cdot \overset{\Omega}{\Omega}(i=0) f \rightsquigarrow f(i=0) &
\end{array}$$

Figure 7.4: Informal notation for modal functions.

exciting, we just need to make sure that we bind all the variables we introduce in the context.

Figure 7.5 lists notations for substitutions arising from 2-cells of the mode theory. We see that $\text{const}_u \downarrow$ (semantically the co-unit **drop** of $\exists(u : \mathbb{U}) \dashv \exists u$ or $\Sigma(u : \mathbb{U}) \dashv \Omega u$) is akin to weakening: it discards the variable $u : \mathbb{U}$ which is the first component of what can be thought of as a non-dependent pair type. In the case of a cancellative and affine multiplier, this first component is completely hidden, so forgetting it is an isomorphism. Conversely, $\text{app}_u \downarrow$ (semantically the unit **copy** of $\exists(u : \mathbb{U}) \dashv \exists u$ or $\Sigma(u : \mathbb{U}) \dashv \Omega u$) is akin to contraction: it operates in a shape context where u is available, and substitutes a copy of u for v .

The substitution $\text{reid}_{x_u} \downarrow$ (semantically the co-unit **app** of $\exists u \dashv \forall(u : \mathbb{U})$ or $\Omega u \dashv \Pi(u : \mathbb{U})$) operates in a shape context where u is available and applies the affine function $\gamma|_{v=\square} : \forall v. \Gamma$, which is fresh for u , to u . Conversely, $\text{unmerid}_u \downarrow$ (semantically the unit **const** of $\exists u \dashv \forall(u : \mathbb{U})$ or $\Omega u \dashv \Pi(u : \mathbb{U})$) creates what can be thought of as a non-dependent function, namely the constant one. In the case of a cancellative and affine multiplier, the output of the function must be completely fresh for the input, so constant functions are the only ones possible and hence the operation is an isomorphism.

Finally, we treat $\text{spoil}_u \downarrow$ (semantically **hide** : $\Sigma(u : \mathbb{U}) \rightarrow \exists(u : \mathbb{U})$) and $\text{cospoil}_u \downarrow$ (semantically **spoil** : $\exists u \rightarrow \Omega u$) as implicit coercions.

Units becoming co-units:

$$\begin{array}{l} \Xi \\ \sim \Xi \end{array} \quad \left| \begin{array}{l} \text{const}_u \downarrow_{\text{af}}^{\circ} \\ () \end{array} \right. \quad \begin{array}{l} : (\Gamma, \mathbf{\Delta}_{\forall}^{\circ} u, \mathbf{\Delta}_{\exists}^{\circ} u) \\ : \exists(u : \mathbb{U}). \exists u. \Gamma \end{array} \quad \begin{array}{l} \rightarrow \Gamma \\ \rightarrow \Gamma \end{array}$$

$$\begin{array}{l} \Xi \\ \sim \Xi \end{array} \quad \left| \begin{array}{l} \text{const}_u \downarrow_{\text{po}}^{\circ} \\ () \end{array} \right. \quad \begin{array}{l} : (\Gamma, \mathbf{\Delta}_{\Pi}^{\circ} u, \mathbf{\Delta}_{\Omega}^{\circ} u) \\ : \Sigma(u : \mathbb{U}). \Omega u. \Gamma \end{array} \quad \begin{array}{l} \rightarrow \Gamma \\ \rightarrow \Gamma \end{array}$$

$$\begin{array}{l} \Xi, i : \mathbb{I} \\ \sim \Xi, i : \mathbb{I} \end{array} \quad \left| \begin{array}{l} \text{const}_{(i=0)} \downarrow_{\text{po}}^{\circ} \\ () \end{array} \right. \quad \begin{array}{l} : (\Gamma, \mathbf{\Delta}_{\Pi(i=0)}^{\circ}, \mathbf{\Delta}_{\Omega(i=0)}^{\circ}) \\ : \Sigma(i = 0). \Omega(j = 0). \Gamma[j/i] \end{array} \quad \begin{array}{l} \rightarrow \Gamma \\ \rightarrow \Gamma \end{array}$$

$$\begin{array}{l} \Xi, u : \mathbb{U} \\ \sim \Xi, u : \mathbb{U} \end{array} \quad \left| \begin{array}{l} \text{reid}_u \downarrow_{\text{ta}}^{\circ} \\ (\gamma|_{v=u}/\gamma) \end{array} \right. \quad \begin{array}{l} : (\Gamma, \mathbf{\Delta}_{\exists}^{\circ} u, \mathbf{\Delta}_{\forall}^{\circ} u) \\ : \exists u. \forall(v : \mathbb{U}). \Gamma \end{array} \quad \begin{array}{l} \rightarrow \Gamma \\ \rightarrow \Gamma[u/v] \end{array}$$

Co-units becoming units:

$$\begin{array}{l} \Xi, u : \mathbb{U} \\ \sim \Xi, u : \mathbb{U} \end{array} \quad \left| \begin{array}{l} \text{app}_u \downarrow_{\circ}^{\text{fa}} \\ (u/v) \end{array} \right. \quad \begin{array}{l} : \Gamma \rightarrow (\Gamma, \mathbf{\Delta}_{\exists}^{\circ} u, \mathbf{\Delta}_{\forall}^{\circ} u) \\ : \Gamma[u/v] \rightarrow \exists u. \exists(v : \mathbb{U}). \Gamma \end{array}$$

$$\begin{array}{l} \Xi, u : \mathbb{U} \\ \sim \Xi, u : \mathbb{U} \end{array} \quad \left| \begin{array}{l} \text{app}_u \downarrow_{\circ}^{\text{op}} \\ (u/v) \end{array} \right. \quad \begin{array}{l} : \Gamma \rightarrow (\Gamma, \mathbf{\Delta}_{\Omega}^{\circ} u, \mathbf{\Delta}_{\Pi}^{\circ} u) \\ : \Gamma[u/v] \rightarrow \Omega u. \Sigma(v : \mathbb{U}). \Gamma \end{array}$$

$$\begin{array}{l} \Xi \\ \sim \Xi \end{array} \quad \left| \begin{array}{l} \text{app}_{(i=0)} \downarrow_{\circ}^{\text{op}} \\ () \end{array} \right. \quad \begin{array}{l} : \Gamma \rightarrow (\Gamma, \mathbf{\Delta}_{\Omega(i=0)}^{\circ}, \mathbf{\Delta}_{\Pi(i=0)}^{\circ}) \\ : \Gamma \rightarrow \Omega(i = 0). \Sigma(i = 0). \Gamma \end{array}$$

$$\begin{array}{l} \Xi \\ \sim \Xi \end{array} \quad \left| \begin{array}{l} \text{unmerid}_u \downarrow_{\circ}^{\text{at}} \\ (\lambda_{_}. \gamma / \gamma|_{u=_}) \end{array} \right. \quad \begin{array}{l} : \Gamma \rightarrow (\Gamma, \mathbf{\Delta}_{\forall}^{\circ} u, \mathbf{\Delta}_{\exists}^{\circ} u) \\ : \Gamma \rightarrow \forall(u : \mathbb{U}). \exists u. \Gamma \end{array}$$

Affine and cancellative multipliers:

$$\begin{array}{l} \Xi \\ \sim \Xi \end{array} \quad \left| \begin{array}{l} \text{const}_u \downarrow_{\text{af}}^{\circ} \\ () \end{array} \right. \quad \begin{array}{l} : (\Gamma, \mathbf{\Delta}_{\forall}^{\circ} u, \mathbf{\Delta}_{\exists}^{\circ} u) \cong \Gamma \\ : \Gamma, \exists(u : \mathbb{U}). () \cong \Gamma \end{array}$$

$$\begin{array}{l} \Xi \\ \sim \Xi \end{array} \quad \left| \begin{array}{l} \text{unmerid}_u \downarrow_{\circ}^{\text{at}} \\ () \end{array} \right. \quad \begin{array}{l} : \Gamma \cong (\Gamma, \mathbf{\Delta}_{\forall}^{\circ} u, \mathbf{\Delta}_{\exists}^{\circ} u) \\ : \Gamma \cong \Gamma, \forall(u : \mathbb{U}). () \end{array}$$

Semicartesian multipliers:

$$\begin{array}{l} \Xi \\ \sim \Xi \end{array} \quad \left| \begin{array}{l} \text{spoil} \downarrow_{\circ}^{\text{f}} \\ () \end{array} \right. \quad \begin{array}{l} : (\Gamma, \mathbf{\Delta}_{\exists}^{\circ} u) \rightarrow (\Gamma, \mathbf{\Delta}_{\Omega}^{\circ} u) \\ : \Sigma(u : \mathbb{U}). \Gamma \rightarrow \exists(u : \mathbb{U}). \Gamma \end{array}$$

$$\begin{array}{l} \Xi, u : \mathbb{U} \\ \sim \Xi, u : \mathbb{U} \end{array} \quad \left| \begin{array}{l} \text{cospoil} \downarrow_{\circ}^{\text{p}} \\ () \end{array} \right. \quad \begin{array}{l} : (\Gamma, \mathbf{\Delta}_{\forall}^{\circ} u) \rightarrow (\Gamma, \mathbf{\Delta}_{\Pi}^{\circ} u) \\ : \exists u. \Gamma \rightarrow \Omega u. \Gamma \end{array}$$

Figure 7.5: Informal notation for 2-cell substitutions.

As an example, note that if we write example 7.4.18 in the informal notation, then we get very close again to the function as defined in section 7.2.

7.6 Additional Typing Rules

In this section, we add a few extensions to MTT in order to reason about boundaries (definition 7.4.3) in the *type* theory, rather than in the shape theory, and in order to recover all known presheaf operators in section 7.8.

Basic presheaf operators We add propositions (figs. 6.1 and 6.2) and the strictness type (fig. 6.7). Of course we take the extensional versions.

Boundary predicate We add a predicate $\Xi, u : \mathbb{U} \mid \cdot \vdash (u \in \partial\mathbb{U}) : \text{Prop}$ corresponding in the model to the subobject $(\Xi, u : \partial\mathbb{U}) \subseteq (\Xi, u : \mathbb{U})$. A naïve introduction rule would be $\Xi, u : \partial\mathbb{U} \mid \cdot \vdash _ : \langle \Omega(u \in \partial\mathbb{U}) \mid^\circ (u \in \partial\mathbb{U}) \rangle$. As all our modalities are proper DRAs [Bir+20] as opposed to the weaker concepts required by the general model of MTT, the modal introduction rule is invertible in the model, so we may as well take $\Xi, u : \mathbb{U} \mid \cdot, \mathbf{!}_{\Omega(u \in \partial\mathbb{U})} \vdash \text{on}\partial \downarrow_o : (u \in \partial\mathbb{U})$ as an introduction rule. If we absorb a substitution into these rules, we get

$$\frac{\text{BOUNDARY} \quad \Xi, u : \mathbb{U} \mid \Gamma \text{ ctx}}{\Xi, u : \mathbb{U} \mid \Gamma \vdash (u \in \partial\mathbb{U}) \text{ prop}},$$

$$\frac{\text{BOUNDARY:INTRO} \quad \Xi, u : \mathbb{U} \mid \Gamma, \Delta \text{ ctx} \quad \alpha : \Omega(u \in \partial\mathbb{U}) \Rightarrow \text{locks}(\Delta)}{\Xi, u : \mathbb{U} \mid \Gamma, \Delta \vdash \text{on}\partial \alpha \downarrow_{\text{ticks}(\Delta)} : [u \in \partial\mathbb{U}]}$$

An elimination rule could build a function $(_ : u \in \partial\mathbb{U}) \rightarrow A_$ from $\Pi(u \in \partial\mathbb{U}) \circ \Omega(u \in \partial\mathbb{U})$ applied to A , but it is not clear how to formulate the β -rule. Instead, we will eliminate to the transpension type (theorem 7.7.1).

7.7 Investigating the Transpension Type

In this section, we investigate the structure of the transpension type.

Our first observation is that on the boundary, the transpension type is trivial. Let $\top : \Xi_1 \rightarrow \Xi_2$ be the modality, between any two shape contexts, which maps

any presheaf to the terminal presheaf. We clearly have $\top \circ \mu = \top$ for any μ , but also $\mu \circ \top \cong \top$ because all internal modalities are right adjoints and therefore preserve the terminal object.

Theorem 7.7.1 (Pole). We have $\Omega(u \in \partial\mathbb{U}) \circ \check{\Omega}(u : \mathbb{U}) \cong \top$. We can thus postulate a term $(u \in \partial\mathbb{U}) \vdash \text{pole} : \langle \check{\Omega}(u : \mathbb{U}) \mid^t T \rangle$ for any T , with an η -rule $(u \in \partial\mathbb{U}) \vdash t = \text{pole} : \langle \check{\Omega}(u : \mathbb{U}) \mid^t T \rangle$.

Sketch of proof. The left adjoints $\forall(u : \mathbb{U}) \circ \Sigma(u \in \partial\mathbb{U})$ and \perp are isomorphic because $\forall(u : \mathbb{U}).(u \in \partial\mathbb{U})$ is false. We give a full proof in the technical report [Nuy20]. \square

Definition 7.4.3 of the boundary relied on the notion of dimensional splitness. The following result shows that it was a good one: the transpension is *only* trivial on the boundary:

Theorem 7.7.2. In the model, we have $\cdot, u : \mathbb{U} \mid \Gamma \vdash (u \in \partial\mathbb{U}) \cong \langle \check{\Omega}(u : \mathbb{U}) \mid^t \text{Empty} \rangle$. [Nuy20]

Meridians As all our modalities are proper DRAs [Bir+20], the modal introduction rule is invertible in the model. This immediately shows that sections¹⁰ of the transpension type $\Xi \mid \Gamma \vdash f : \langle \forall(u : \mathbb{U}) \mid^a \langle \check{\Omega}(u : \mathbb{U}) \mid^t T \rangle \rangle$ (which we call meridians) are in 1-1 correspondence with terms $\Xi \mid \Gamma, \mathbf{lock}_{\forall(u:\mathbb{U})}^a, \mathbf{lock}_{\check{\Omega}(u:\mathbb{U})}^t \vdash t : T$. If it were not for the locking of the context, this characterization in terms of poles and meridians would make the transpension type look quite similar to a dependent version of the suspension type in HoTT [Uni13], whence our choice of name. If \mathbb{U} is cancellative and affine, then the locks can actually be ignored (theorem 7.4.16). In any case, proposition 7.3.3 tells us that the let-rule for $\check{\Omega}(u : \mathbb{U})$ has the same power as

$$\text{prmod}_{\check{\Omega}(u:\mathbb{U})} : (\forall(u : \mathbb{U}) \mid^a \langle \check{\Omega}(u : \mathbb{U}) \mid^t T \rangle) \rightarrow T[\text{unmerid}_u \downarrow_{\bullet}^{a,t}] \quad (\text{formal MTT})$$

$$\text{unmerid} : (\forall(u : \mathbb{U}) \mid \check{\Omega} u.T) \rightarrow T[\lambda_{_}.\gamma/\gamma|_{u=_}] \quad (\text{informal})$$

which extracts meridians. If \mathbb{U} is cancellative and affine, then unmerid_u is invertible (theorem 7.4.16) and we can also straightforwardly *create* meridians from elements of $T[\text{unmerid}_u \downarrow_{\bullet}^{a,t}]$.

Pattern matching The eliminator $\text{prmod}_{\check{\Omega}(u:\mathbb{U})}$ is only capable of eliminating *sections* of the transpension type. Sometimes we can eliminate locally by pattern matching:

¹⁰By a section of a dependent type, we mean a dependent function with the same domain as the type.

In formal MTT syntax:

TRANSP:ELIM

$$\Xi, u : \mathbb{U} \mid \Gamma \text{ ctx}$$

$$\Xi \mid \Gamma, \mathbf{a}_{\check{Q}u}^t \vdash A \text{ type}$$

$$\Xi, u : \mathbb{U} \mid \Gamma, r : \langle \check{Q}u \mid^t A \rangle \vdash C \text{ type}$$

$$\Xi, u : \mathbb{U} \mid \Gamma, u \in \partial\mathbb{U} \vdash c_{\text{pole}} : C[\text{pole}/r]$$

$$\Xi, u : \mathbb{U} \mid \Gamma, \mathbf{a}_{\check{Q}u}^t, x : A, \mathbf{a}_{\forall u}^a \vdash c_{\text{merid}} : C[\text{reid}_{x_u} \downarrow_{t\alpha}^*] \left[\text{mod}_{\check{Q}u}^{t'} (x \text{ unmerid}_u^{-1} \downarrow_{\alpha t'}) / r \right]$$

$$\Xi, u : \mathbb{U} \mid \Gamma, \mathbf{a}_{\check{Q}u}^t, x : A, \mathbf{a}_{\forall u}^a, u \in \partial\mathbb{U} \vdash$$

$$c_{\text{merid}} = c_{\text{pole}}[\text{reid}_{x_u} \downarrow_{t\alpha}^*] : C[\text{reid}_{x_u} \downarrow_{t\alpha}^*, \text{pole}/r]$$

$$\Xi, u : \mathbb{U} \mid \Gamma \vdash t : \langle \check{Q}u \mid^t A \rangle$$

$$\Xi, u : \mathbb{U} \mid \Gamma \vdash c := \text{case } t \text{ of } \{ \text{pole} \mapsto c_{\text{pole}} \mid \text{merid}(t, x, \mathbf{a}) \mapsto c_{\text{merid}} \} : C[t/r]$$

where $c[\text{pole}/t] = c_{\text{pole}}$

(TRANSP:ELIM:POLE)

$$\Delta, \mathbf{a}_{\forall u}^a \vdash c = c_{\text{merid}}[\text{prmod}_{\check{Q}u} \cdot^a (t[\text{reid}_{x_u} \downarrow_{t\alpha}^*]) / x, \downarrow_{\alpha}^a][(\text{unmerid}_u \star \text{id}_{\forall u}) \downarrow_{\alpha t\alpha}^{a'}]$$

(TRANSP:ELIM:SECTION)

In informal notation:

$$\Xi, u : \mathbb{U} \mid \Gamma \text{ ctx}$$

$$\Xi \mid \forall (u : \mathbb{U}). \Gamma \vdash A \text{ type}$$

$$\Xi, u : \mathbb{U} \mid \Gamma, r : \check{Q}u.A \vdash C \text{ type}$$

$$\Xi, u : \mathbb{U} \mid \Gamma, u \in \partial\mathbb{U} \vdash c_{\text{pole}} : C[\text{pole}/r]$$

$$\Xi, v : \mathbb{U} \mid \exists v. (\forall (u : \mathbb{U}). \Gamma, x : A) \vdash c_{\text{merid}} : C[\gamma|_{u=v}/\gamma, \text{merid } v \ x / r]$$

$$\Xi, v : \mathbb{U} \mid \exists v. (\forall (u : \mathbb{U}). \Gamma, x : A), v \in \partial\mathbb{U} \vdash$$

$$c_{\text{merid}} = c_{\text{pole}}[\gamma|_{u=v}/\gamma] : C[\gamma|_{u=v}/\gamma, \text{pole}/r]$$

$$\Xi, u : \mathbb{U} \mid \Gamma \vdash t : \check{Q}u.A$$

$$\Xi, u : \mathbb{U} \mid \Gamma \vdash c := \text{case } t \text{ of } \{ \text{pole} \mapsto c_{\text{pole}} \mid \text{merid } v \ x \mapsto c_{\text{merid}} \} : C[t/r]$$

where $c[\text{pole}/t] = c_{\text{pole}}$

$$\exists u. \Delta \vdash c = c_{\text{merid}}[u/v, \text{unmerid}(w.t[\gamma|_{u=w}/\gamma]) / x]$$

Figure 7.6: Transpension elimination by pattern matching (sound if \mathbb{U} is cancellative, affine and connection-free).

Theorem 7.7.3. If \mathbb{U} is cancellative, affine and connection-free, then the rule `TRANS:ELIM` in fig. 7.6 is sound [Nuy20].

We get a context Γ depending on $u : \mathbb{U}$, a type A depending on sections of Γ (recall that $\mathbf{a}_{\exists u}$ is semantically $\forall u$), a type C depending on u and $\langle \exists u \mid A \rangle$, and an argument t of type $\langle \exists u \mid A \rangle$. To obtain a value of type C , we need to give an action c_{pole} on the boundary, where t is necessarily `pole` (pole theorem 7.7.1), and a compatible action on sections of the transpension type, i.e. meridians, which are essentially elements of A (quantification theorem 7.4.16), to sections of C (the quantifier has been brought to the left as $\mathbf{a}_{\forall u}$). Thanks to connection-freedom, we know that everything that is not (degenerate on) a section, is on the boundary, so this suffices. The computation rule for meridians is in a non-general context and needs to be forcibly closed under substitution.

7.8 Recovering Known Operators

In this section, we explain how to recover the amazing right adjoint \surd [Lic+18], Moulin et al.'s Φ and Ψ combinators [BCM15; Mou16] and `Glue` [Coh+17; NVD17a], `Weld` [NVD17a] and `mill` [ND18b] from the transpension, the strictness axiom [OP18] and certain pushouts.

The amazing right adjoint \surd Licata et al. [Lic+18] use presheaves over a cartesian base category of cubes and introduce \surd as the right adjoint to the non-dependent exponential $\mathbb{I} \rightarrow \sqcup$. We generalize to semicartesian systems and look for a right adjoint to $\mathbb{U} \multimap \sqcup$, which decomposes as substructural quantification after cartesian weakening $\forall(u : \mathbb{U}) \circ \Omega(u : \mathbb{U})$. Then the right adjoint is obviously $\surd_{\mathbb{U}} := \Pi(u : \mathbb{U}) \circ \exists(u : \mathbb{U})$. The type constructor has type $\langle \surd_{\mathbb{U}} \mid \sqcup \rangle : (\surd_{\mathbb{U}} \uparrow^{\mathbb{U}} \mathbb{U}_{\ell}) \rightarrow \mathbb{U}_{\ell}$ and the transposition rule is as in proposition 7.3.4. This is an improvement in two ways: First, we have computation rules, so that we do not need to postulate functoriality of $\surd_{\mathbb{U}}$ and invertibility of transposition. Secondly, we have no need for a global sections modality \flat . Instead, we use the modality $\surd_{\mathbb{U}}$ to escape Licata et al.'s no-go theorems. Our overly general mode theory does contain $\flat : \mathbb{T} \rightarrow \mathbb{T}$, and Licata et al.'s formation and functoriality axioms can be proven because $\flat \Rightarrow \surd_{\mathbb{U}}$. Their transposition rule is provable by applying $\langle \flat \mid \sqcup \rangle$ to both sides of the isomorphism in proposition 7.3.4 and using that $\flat \circ \surd_{\mathbb{U}} \cong \flat$, which follows from an isomorphism between their left adjoints $(\mathbb{I} \multimap \sqcup) \circ \int \cong \int$, where $\int \dashv \flat$ is the connected components functor.

Binary and destrictified reformulation of Moulin et al.'s original Φ -rule:
[BCM15; Mou16]

$$\begin{array}{l}
\Delta, i : \mathbb{I} \vdash B \text{ type} \\
\Delta, i : \mathbb{I}, y : B \vdash C \text{ type} \\
\Delta, y : B[\epsilon/i] \vdash c_\epsilon : C[\epsilon/i] \quad (\epsilon \in \{0, 1\}) \\
\Delta, h : \forall(i : \mathbb{I}). B, i : \mathbb{I} \vdash c : C[h i/b] \\
\Delta, h : \forall(i : \mathbb{I}). B \vdash c[\epsilon/i] = c_\epsilon[h \epsilon/b] : C[h \epsilon/b] \quad (\epsilon \in \{0, 1\}) \\
\Delta, i : \mathbb{I} \vdash b : B \\
\hline
\Delta, i : \mathbb{I} \vdash \Phi_i c_0 c_1 c b : C[b/y] \\
\text{where } \Phi_\epsilon c_0 c_1 c b = c_\epsilon \quad (\epsilon \in \{0, 1\}) \\
\Phi_i c_0 c_1 c b = c[\lambda i. b/h]
\end{array}$$

In formal MTT syntax:

PHI

$$\begin{array}{l}
\Xi, u : \mathbb{U} \mid \Gamma \vdash B \text{ type} \\
\Xi, u : \mathbb{U} \mid \Gamma, y : B \vdash C \text{ type} \\
\Xi, u : \mathbb{U} \mid \Gamma, y : B, u \in \partial\mathbb{U} \vdash c_\partial : C \\
\Xi, u : \mathbb{U} \mid \Gamma, y : B, \mathbf{a}_{\forall u}^{\mathbf{a}}, \mathbf{a}_{\forall u}^{\mathbf{a}} \vdash c : C[\text{reid}_{x_u} \downarrow_{\text{ta}}^*] \\
\Xi, u : \mathbb{U} \mid \Gamma, y : B, \mathbf{a}_{\forall u}^{\mathbf{a}}, \mathbf{a}_{\forall u}^{\mathbf{a}}, u \in \partial\mathbb{U} \vdash c = c_\partial[\text{reid}_{x_u} \downarrow_{\text{ta}}^*] : C[\text{reid}_{x_u} \downarrow_{\text{ta}}^*] \\
\Xi, u : \mathbb{U} \mid \Gamma \vdash b : B \\
\hline
\Xi, u : \mathbb{U} \mid \Gamma \vdash \Phi_u (y.c_\partial) (y.t.a.c) b : C[b/y] \\
\text{where } \Gamma, u \in \partial\mathbb{U} \vdash \Phi_u (y.c_\partial) (y.t.a.c) b = c_\partial[b/y] \\
(\text{PHI:BOUNDARY}) \\
\Delta, \mathbf{a}_{\forall u}^{\mathbf{a}} \vdash (\Phi_u (y.c_\partial) (y.t.a.c) b) = c[b/y, \downarrow_{\text{ta}}^{\text{ta}}][(\text{unmerid}_u \star \text{id}_{\forall u}) \downarrow_{\mathbf{a}'}^{\mathbf{a}'}] \\
(\text{PHI:SECTION})
\end{array}$$

In informal notation:

$$\begin{array}{l}
\Xi, u : \mathbb{U} \mid \Gamma \vdash B \text{ type} \\
\Xi, u : \mathbb{U} \mid \Gamma, y : B \vdash C \text{ type} \\
\Xi, u : \mathbb{U} \mid \Gamma, y : B, u \in \partial\mathbb{U} \vdash c_\partial : C \\
\Xi, v : \mathbb{U} \mid \exists v. \forall(u : \mathbb{U}). (\Gamma, y : B) \vdash c : C[\gamma|_{u=v}/\gamma, y|_{u=v}/y] \\
\Xi, v : \mathbb{U} \mid \exists v. \forall(u : \mathbb{U}). (\Gamma, y : B), v \in \partial\mathbb{U} \vdash \\
\quad c = c_\partial[v/u, \gamma|_{u=v}/\gamma, b|_{u=v}/b] : C[\gamma|_{u=v}/\gamma, y|_{u=v}/y] \\
\Xi, u : \mathbb{U} \mid \Gamma \vdash b : B \\
\hline
\Xi, u : \mathbb{U} \mid \Gamma \vdash \Phi_u (y.c_\partial) (y.v.c) b : C[b/y] \\
\text{where } \Gamma, u \in \partial\mathbb{U} \vdash \Phi_u (y.c_\partial) (y.v.c) b = c_\partial[b/y] \\
\exists u. \Delta \vdash \Phi_u (y.c_\partial) (y.v.c) b = c[u/v, \lambda_. b/y|_{v=\perp}]
\end{array}$$

Figure 7.7: The Φ -rule for $(y : B) \rightarrow C$ (sound if \mathbb{U} is cancellative, affine and connection-free).

The Φ -combinator In fig. 7.7, we *state* Moulin et al.'s Φ -rule [BCM15; Mou16]; both a slight reformulation adapted to the naïve system (section 7.2) and the rule PHI adapted to the current system. For types B and C (depending on $u : \mathbb{U}$), the combinator allows us to define functions of naïve type $\forall u.(y : B u) \rightarrow C u y$ from an action c_∂ on the boundary (in Moulin et al.'s work: on every endpoint of the interval) and a compatible action c on sections $\forall u.B u$. Given a section of B (recall that $\mathbf{a}_{\exists u}$ is semantically $\forall u$), c provides a section of C (but the quantification has been moved to the left as $\mathbf{a}_{\forall u}$), compatible with c_∂ on the boundary. The result is a term of type C which matches the given actions on the boundary and on sections. Again, the computation rule for sections is in a non-general context and needs to be forcibly closed under substitution.

The main difference with Moulin et al.'s formulation is that they require that $i : \mathbb{I}$ be the last variable before y , i.e. it comes after Γ . Here, this would mean that Γ is fresh for $u : \mathbb{U}$, i.e. $\Gamma = (\Delta, \mathbf{a}_{\forall u})$ or informally $\Gamma = \exists u.\Delta$. In that case, because Moulin et al.'s system is cancellative and affine, the informal notation of the context of c then becomes $\exists v.(\Delta, \forall(u : \mathbb{U}).(y : B))$, which corresponds more closely to Moulin et al.'s rules. The greater generality of our Φ -rule means that there is in fact no reason not to absorb B into Γ .

We remark that if $C = \langle \exists u |^t D \rangle$ and \mathbb{U} is cancellative and affine, then the Φ -rule follows from the introduction rule of the transpension type by pole theorem 7.7.1 and quantification theorem 7.4.16. Indeed, we can then define:

$$\begin{aligned} \Phi_u(y.c_\partial)(y.t.a.c)y &:= \text{mod}_{\exists u}^t(\text{prmod}_{\exists u} \cdot^a c) : \langle \exists u |^t D \rangle \\ \Phi_u(y.c_\partial)(y.v.c)y &:= \text{merid } u(\text{unmerid}(v.c)) : \exists u.D. \end{aligned}$$

Theorem 7.8.1. If \mathbb{U} is cancellative, affine and connection-free, then the Φ -rule PHI in fig. 7.7 is derivable for all B and C .

Proof. Absorb B into Γ and use the case-eliminator for $\langle \exists u |^t \text{Unit} \rangle$ (theorem 7.7.3):

$$\begin{aligned} \Phi_u(y.c_\partial)(y.t.a.c)y &:= \text{case}(\text{mod}_{\exists u}^t \text{unit}) \text{ of } \left\{ \begin{array}{ll} \text{pole} & \mapsto c_\partial \\ \text{merid}(t, _, a) & \mapsto c \end{array} \right\} \\ \Phi_u(y.c_\partial)(y.v.c)y &:= \text{case}(\text{merid } u \text{ unit}) \text{ of } \left\{ \begin{array}{ll} \text{pole} & \mapsto c_\partial \\ \text{merid } v _ & \mapsto c \end{array} \right\} \quad \square \end{aligned}$$

The Ψ -combinator Moulin et al.'s Ψ -combinator constructs an edge in the universe with endpoints A_ϵ from a relation $R : \times_\epsilon A_\epsilon \rightarrow \mathbb{U}$. In fig. 7.8, we adapt the typing rules, replacing the concept of endpoints with the boundary.

Binary and destrictified reformulation of Moulin et al.'s original Ψ -type:
[BCM15; Mou16]

$$\begin{array}{c}
 \Delta \vdash A_\epsilon \text{ type} \qquad (\epsilon \in \{0, 1\}) \\
 \Delta, x_0 : A_0, x_1 : A_1 \vdash R \text{ type} \\
 \hline
 \Delta, i : \mathbb{I} \vdash \Psi_i A_0 A_1 (x_0.x_1.R) \text{ type} \\
 \text{where } \Psi_\epsilon A_0 A_1 R = A_\epsilon \qquad (\epsilon \in \{0, 1\}) \\
 \\
 \Delta \vdash a_\epsilon : A_\epsilon \qquad (\epsilon \in \{0, 1\}) \\
 \Delta \vdash r : R[a_0/x_0, a_1/x_1] \\
 \hline
 \Delta, i : \mathbb{I} \vdash \text{in}\Psi_i a_0 a_1 r : \Psi_i A_0 A_1 (x_0.x_1.R) \\
 \text{where } \text{in}\Psi_\epsilon a_0 a_1 r = a_\epsilon \qquad (\epsilon \in \{0, 1\}) \\
 q = \text{in}\Psi_i a_0 a_1 (\text{out}\Psi(j.q[j/i])) \\
 \hline
 \Delta, i : \mathbb{I} \vdash q : \Psi_i A_0 A_1 (x_0.x_1.R) \\
 \hline
 \Delta \vdash \text{out}\Psi(i.q) : R[q[0/i]/x_0, q[1/i]/x_1] \\
 \text{where } \text{out}\Psi(i.\text{in}\Psi_i a_0 a_1 r) = r
 \end{array}$$

In formal MTT syntax:

$$\begin{array}{c}
 \text{PSI} \\
 \Xi, u : \mathbb{U} \mid \Gamma, u \in \partial\mathbb{U} \vdash A \text{ type} \\
 \Xi \mid \Gamma, \alpha : (_ : [u \in \partial\mathbb{U}]) \rightarrow A, \mathbf{\Delta}_{\forall(u:\mathbb{U})}^{\dagger} \vdash R \text{ type} \\
 \hline
 \Xi, u : \mathbb{U} \mid \Gamma \vdash \Psi_u A (\alpha.t.R) \text{ type} \\
 \text{where } u \in \partial\mathbb{U} \vdash \Psi_u A (\alpha.t.R) = A \quad (\text{PSI:BOUNDARY}) \\
 \\
 \text{PSI:INTRO} \\
 \Xi, u : \mathbb{U} \mid \Gamma, u \in \partial\mathbb{U} \vdash a : A \\
 \Xi \mid \Gamma, \mathbf{\Delta}_{\forall(u:\mathbb{U})}^{\dagger} \vdash r : R[\lambda_ . a/\alpha, \downarrow_i^{\dagger}] \\
 \hline
 \Xi, u : \mathbb{U} \mid \Gamma \vdash \text{in}\Psi_u a (t.r) : \Psi_u A (\alpha.t.R) \\
 \text{where } u \in \partial\mathbb{U} \vdash \text{in}\Psi_u a (t.r) = a \quad (\text{PSI:INTRO:BOUNDARY}) \\
 q = \text{in}\Psi_u q (t.\text{out}\Psi \cdot^{\alpha} q[\text{reid}_{x_u} \downarrow_{t_0}^{\alpha}]) \quad (\text{PSI:BETA}) \\
 \\
 \text{PSI:ELIM} \\
 \Xi, u : \mathbb{U} \mid \Delta, \mathbf{\Delta}_{\forall(u:\mathbb{U})}^{\dagger} \vdash q : \Psi_u A (\alpha.t.R) \\
 \hline
 \Xi \mid \Delta \vdash \text{out}\Psi \cdot^{\alpha} q : R[\lambda_ . q/\alpha, \downarrow_i^{\dagger}][\text{unmerid}_u \downarrow_{\ast}^{\alpha \dagger}] \\
 \text{where } \text{out}\Psi \cdot^{\alpha} \text{in}\Psi_u a (t.r) = r[\text{unmerid}_u \downarrow_{\ast}^{\alpha \dagger}] \quad (\text{PSI:ETA})
 \end{array}$$

In informal notation:

$$\begin{array}{c}
 \Xi, u : \mathbb{U} \mid \Gamma, u \in \partial\mathbb{U} \vdash A \text{ type} \\
 \Xi \mid \forall(u : \mathbb{U}).(\Gamma, \alpha : (_ : [u \in \partial\mathbb{U}]) \rightarrow A) \vdash R \text{ type} \\
 \hline
 \Xi, u : \mathbb{U} \mid \Gamma \vdash \Psi_u A (\alpha.R) \text{ type} \\
 \text{where } u \in \partial\mathbb{U} \vdash \Psi_u A (\alpha.R) = A \\
 \\
 \Xi, u : \mathbb{U} \mid \Gamma, u \in \partial\mathbb{U} \vdash a : A \\
 \Xi \mid \forall(u : \mathbb{U}).\Gamma \vdash r : R[\lambda u. \lambda_ . q/\alpha|_{u=_}] \\
 \hline
 \Xi, u : \mathbb{U} \mid \Gamma \vdash \text{in}\Psi_u a r : \Psi_u A (\alpha.R) \\
 \text{where } u \in \partial\mathbb{U} \vdash \text{in}\Psi_u a r = a \\
 q = \text{in}\Psi_u a (\text{out}\Psi(v.q[v/u, \gamma|_{u=v/\gamma}])) \\
 \hline
 \Xi, u : \mathbb{U} \mid \exists u. \Delta \vdash q : \Psi_u A (\alpha.R) \\
 \hline
 \Xi \mid \Delta \vdash \text{out}\Psi(u.q) : R[\lambda_ . \delta/\delta|_{u=_}, \lambda u. \lambda_ . q/\alpha|_{u=_}] \\
 \text{where } \text{out}\Psi(u.\text{in}\Psi_u a r) = r[\lambda_ . \delta/\delta|_{u=_}]
 \end{array}$$

Figure 7.8: Typing rules for the Ψ -type.

The formation rule takes a type A that exists on the boundary, and a type R depending on sections of A (with $\mathbf{A}_{\forall u}$ being $\forall u$). It yields a Ψ -type which equals A on the boundary. The introduction rule is only necessary when we are not on the boundary (otherwise we can just coerce elements of A). It requires an element of A on the boundary and, for every section of Γ (which need not exist), a proof that the resulting boundary section satisfies R . The elimination rule sends sections of the Ψ -type (the quantification has been brought to the left as $\mathbf{A}_{\forall u}$) to proofs that the boundary part satisfies R . We have β - and η -rules. In cancellative, affine and connection-free systems, the Φ -rule yields a pattern-matching eliminator. Again, the main difference with Moulin et al. is that Γ need not be fresh for u .

The Ψ -type can be implemented by strictifying the following using `Strict`:

$$\Psi_u A (\alpha.t.R) := (\alpha : (_ : u \in \partial U) \rightarrow A) \times \langle \delta u \mid^\dagger R \rangle,$$

The fact that it is isomorphic to A on the boundary follows from the pole theorem 7.7.1.

Transpensity The Φ -rule is extremely powerful but not available in all systems. However, when the codomain C is a Ψ -type, then the `inPsi`-rule is actually quite similar to the Φ -rule. As such, we take an interest in types that are very Ψ -like. We have a monad (idempotent if \mathbb{U} is cancellative and affine)

$$\bar{\Psi}_u A := \Psi_u A (\alpha.t.\langle \forall u \mid^\square A \text{ext}\{u \in \partial \mathbb{U} ? \alpha _ \} [\text{reid} \times_u \downarrow_{\text{ta}}^\circ] \rangle)$$

$$\bar{\Psi}_u A := \Psi_u A (\alpha.\forall v.A \text{ext}\{u \in \partial \mathbb{U} ? \alpha _ \} [v/u, \gamma|_{u=v}/\gamma])$$

where $A \text{ext}\{\varphi ? a\}$ is the type of elements of A that are equal to a when φ holds (fig. 6.6).

Definition 7.8.2. A type is **transpensive over u** if it is a monad-algebra for $\bar{\Psi}_u$.

For cancellative, affine and connection-free multipliers, Φ entails that all types are transpensive. For other systems, many interesting types will still be transpensive, allowing to eliminate *to* them in a Φ -like way.

Glue, Weld, mill $\text{Glue}\{A \leftarrow (\varphi ? T, f)\}$ and $\text{Weld}\{A \rightarrow (\varphi ? T, g)\}$ are similar to `Strict` but extend unidirectional functions. As already discussed in section 6.3, Orton and Pitts [OP18] show that `Glue` [Coh+17; NVD17a] can be implemented by strictifying a pullback along $A \rightarrow (\varphi \rightarrow A)$ [ND18b] which is definable internally using a Σ -type. Dually, `Weld` [NVD17a] can be implemented if there

is a type former for pushouts along $\varphi \times A \rightarrow A$ where $\varphi : \mathbf{Prop}$ [ND18b], which is sound in all presheaf categories. Finally, mill [ND18b] states that $\forall(u : \mathbb{U})$ preserves \mathbf{Weld} and is provable by higher-dimensional pattern matching.

Locally fresh names Nominal type theory is modelled in the Schanuel topos [Pit14] which is a subcategory of $\mathbf{Psh}({}^0\mathbf{Cube}_{\square})$. As fibrancy is not considered in this chapter, we will work directly in $\mathbf{Psh}({}^0\mathbf{Cube}_{\square})$. Names can be modelled using the multiplier $\sqsubset * (i : \mathbb{I})$. Interestingly, the fresh weakening functor $\sqsubset_{(i:\mathbb{I})}$ is then *inverse* to its left adjoint $\exists_{(i:\mathbb{I})}$. By consequence, we get $\exists i \cong \forall i$ (the fresh name quantifier) and $\sqsubset i \cong \exists i$.

Moreover, by theorem 7.4.16, we have $\forall i \circ \sqsubset i \cong 1$. An element of $\langle \forall i \circ \sqsubset i \mid A \rangle$ is created by first abstracting over i , then making sure that the body is fresh for i . This is exactly how locally fresh name abstraction $\nu(i : \mathbb{I})$ works, and the fact that $\forall i \circ \sqsubset i \cong 1$ allows us to use it anywhere.

Consider Pitts's implementation of higher dimensional pattern matching [Pit14]:

$$f : (\forall i.X \uplus Y) \rightarrow (\forall i.X) \uplus (\forall i.Y)$$

$$f \hat{c} = \nu(i : \mathbb{I}).\text{case } \hat{c} \text{ of } \left\{ \begin{array}{ll} \text{inl } a & \mapsto \text{inl } (\lambda i.a) \\ \text{inr } b & \mapsto \text{inr } (\lambda i.b) \end{array} \right\}$$

The right to use ν is derived from the isomorphism $\forall i \circ \sqsubset i \cong 1$. Recalling that $\sqsubset i \cong \exists i$, this translates to *unmerid*. Then ν itself translates to abstraction over i , and instead of making the body fresh for i , we can transpense over it, allowing us to view a and b as functions, justifying the variable capture in nominal type theory.

7.9 Conclusion

To summarize, the transpension type can be defined in a broad class of presheaf models and generalizes previous internalization operators. For now, we only present an extensional type system without an algorithmic typing judgement. The major hurdles towards producing an intensional version with decidable type-checking, are the following:

- We need to decide equality of 2-cells. Solutions may exist in the literature on higher-dimensional rewriting.
- If we want the substitution modality to reduce (remark 7.4.15), we need to solve the following problem: when $\hat{a} = \mathbf{mod}_{\Omega \sigma}^{\circ} a$ definitionally, then we

need to infer a up to definitional equality from \hat{a} in order to β -reduce the let-rule for $\langle \Omega \sigma \mid A \rangle$.

- We need a syntax-directed way to close the section computation rules of Φ (fig. 7.7) and transpension elimination (section 7.7) under substitution.
- We need to decide whether a proposition is true. This problem has been dealt with in special cases, e.g. in implementations of cubical type theory [VMA19].

Chapter 8

Fibrancy

8.1 Introduction

As explained in section 1.6, many models of type theory restrict the semantics of the type judgement $\Gamma \vdash T$ type to fibrant types, which are those types $\Gamma \vdash T$ type such that the weakening substitution $(\Gamma, x : T) \rightarrow \Gamma$ is a fibration, i.e. belongs to the right class of an NWFS on the category of contexts (remark 2.4.12). In order to carry out as much as possible of the construction of a model of type theory internally – which helps abstracting away certain details about the model and also makes feasible the use of a proof assistant – we wish to study fibrancy internally. This chapter builds on the theory of factorization systems in section 2.4.

As motivated in section 1.6, we have three concrete goals:

- To obtain an internal fibrant replacement monad on types. As an internal operator, we want it to be stable under substitution.
- To obtain feasible conditions for the Π -type to be fibrant. In example 8.1.26, we will see that the conditions for the Π -type to be Kan fibrant (necessary in models of HoTT) are stronger than one could expect, and in example 8.1.27, we will see that that the conditions for the Π -type to be Segal fibrant (which is at least desirable in directed type theory) are truly problematic.
- To define internally a predicate $\text{Fib}(T)$ stating that the type T is fibrant.

We will introduce the *robustness* criterion for NWFSs and find that robust NWFSs have a stable fibrant replacement and enjoy the property that the

Π -type is fibrant if its codomain is. Moreover, we will find that, by moving to *contextual* fibrancy, the robustness condition can be more easily satisfied. In contextual fibrancy, types $\Gamma, \Theta \vdash T$ type are fibrant not over the entire context, but only over a telescope Θ , while Γ is just the context in which we consider fibrancy.

The fibrancy predicate can be defined either using the internal fibrant replacement monad (section 8.6), or by exploiting knowledge about the specific NWFS at hand, perhaps using the transpension type (section 8.7).

Overview In section 8.1.1, we define fibrancy for types and consider plenty of examples. In section 8.1.2, we revisit these examples and consider whether their fibrant replacement can be internalized as a type operator stable under substitution. We informally introduce robustness and contextual fibrancy. In section 8.1.3, we revisit the examples again and consider fibrancy of the Π -type.

In section 8.2, we generalize the theory on NWFSs from sections 2.4.5 and 2.4.6 to *damped* NWFSs, which is the categorical counterpart of contextual fibrancy. In section 8.3, we define what we want to achieve: an NWFS is *stable* if its right replacement monad behaves well w.r.t. pullbacks, which are the semantic counterpart of substitution. In section 8.4, we introduce and study the robustness condition. We prove that robustness guarantees stability and fibrancy of the Π -type if its codomain is fibrant. In section 8.5, we present typing rules that internalize both the right/fibrant replacement monad \mathbf{R} and the left coreplacement comonad \mathbf{L} of a *stable* NWFS. In section 8.6, we use the internal fibrant replacement monad to define fibrant types as those equipped with an Eilenberg-Moore algebra structure for the fibrant replacement. In section 8.7, we consider some examples – even non-stable ones – where we can internally define fibrancy by exploiting knowledge about the precise notion of fibrancy at hand.

Contributions We make the following contributions:

- We define damped NWFSs as a categorical foundation for contextual fibrancy. Note that contextual fibrancy in itself is not novel [BT17].
- We define the robustness condition more precisely than I had done so far [Nuy18b; Nuy18a].
- From the technical report of Degrees of Relatedness [Nuy18a], we polish the proof of fibrancy of the Π -type and we strengthen the statement and proof of stability of the fibrant replacement.

- We present typing rules for internalizing a stable NWFS, and use these to define fibrancy internally.
- We demonstrate how the transpension type can be used to define notions of fibrancy internally where this was previously impossible.

How to read We recommend a prior lecture of section 1.6 and of section 2.4 with or without the proofs. After that, this introductory section is of course a good start. The rest of the chapter may on a first lecture be read with special focus on the case of non-damped NWFSs and non-contextual fibrancy.

8.1.1 Fibrant types

Definition 8.1.1. Given an NWFS on a CwF \mathcal{C} , we say that a type $\Gamma \vdash T$ type is **fibrant**¹ if $\pi : \Gamma.T \rightarrow \Gamma$ is a right map, i.e. it is equipped with an Eilenberg-Moore algebra structure f for the right replacement monad $\mathbf{R} : \mathcal{C}^\dagger \rightarrow \mathcal{C}^\dagger$.

Example 8.1.2 (Inhabitation). Continuing examples 2.4.11, 2.4.15, 2.4.22 and 2.4.28, a type $\Gamma \vdash T$ type @ \mathbf{Set} is called **inhabited** if $\Gamma.T \rightarrow \Gamma$ is surjective.

Example 8.1.3 (Subsingletons). Continuing examples 2.4.3 and 2.4.29, a type $\Gamma \vdash T$ type @ $\mathbf{Psh}(\mathcal{W})$ is called a **subsingleton** if $\Gamma.T \rightarrow \Gamma$ is injective.

Example 8.1.4 (Codiscrete graphs). Continuing examples 2.4.5 and 2.4.31, a type $\Gamma \vdash T$ type @ $\mathbf{Psh}(\mathbf{RG})$ is called **codiscrete** if $\Gamma.T \rightarrow \Gamma$ is a codiscrete fibration. Recall from example 4.1.2 the meaning of the type judgement in the CwF $\mathbf{Psh}(\mathbf{RG})$ of reflexive graphs. Codiscreteness of T means that, for any edge $\gamma : \mathbb{I} \Rightarrow \Gamma$ and every two nodes $\mathbb{N} \triangleright t_s : T[\gamma \circ \mathbf{s}]$ and $\mathbb{N} \triangleright t_t : T[\gamma \circ \mathbf{t}]$ above its source and target, there is a unique edge $\mathbb{I} \triangleright t : T[\gamma]$ from t_s to t_t .

Example 8.1.5 (Discrete graphs). Continuing examples 2.4.6 and 2.4.32, a type $\Gamma \vdash T$ type @ $\mathbf{Psh}(\mathbf{RG})$ is called **discrete** if $\Gamma.T \rightarrow \Gamma$ is a discrete fibration. Recall from example 4.1.2 that the nodes of T above a node $\gamma : \mathbb{N} \Rightarrow \Gamma$ and the edges above the reflexive edge $\gamma \circ \mathbf{r}$, together formed a graph. Discreteness of T means that all such graphs are discrete, in the sense that all of their edges are reflexive. Nevertheless, T may contain non-reflexive edges. These must be **heterogeneous**, however, meaning that they live above a non-reflexive edge in Γ .

¹We may have special names for specific NWFSs, e.g. discrete instead of discrete fibrant.

Example 8.1.6 (Clock-irrelevance). Continuing example 2.4.34, a type $\Gamma \vdash T \text{ type @ Psh}(\text{Clock})$ is called **clock-irrelevant** if $\Gamma.T \rightarrow \Gamma$ is a clock-irrelevant fibration. From the semantics of the Π -type, we see that clock-irrelevance means exactly what we wanted it to, namely that $\odot \rightarrow T$ is isomorphic to T . Indeed, a cell $W \triangleright \lambda t : (\odot \rightarrow T)[\gamma]$ is a term $\mathbf{y}W.\odot \vdash t : T[\pi][\gamma+]$. Now since $\pi \circ (\gamma+) = \gamma \circ \pi$, and since \odot is closed so that $\mathbf{y}W.\odot = \mathbf{y}W \times \odot$, this is a diagram

$$\begin{array}{ccc} \mathbf{y}W \times \odot & \xrightarrow{(\gamma \circ \pi_1, t)} & \Gamma.T \\ \pi_1 \downarrow & \nearrow & \downarrow \pi \\ \mathbf{y}W & \xrightarrow{\gamma} & \Gamma \end{array}$$

whose lifting shows that t does not depend on the clock variable.

Example 8.1.7 (0-discrete depth cubical sets). Continuing examples 2.4.7 and 2.4.33, a type $\Gamma \vdash T \text{ type @ Psh}(\text{DCube}_d)$ is called **0-discrete** (or just discrete) if $\Gamma.T \rightarrow \Gamma$ is a 0-discrete fibration. 0-discreteness for $d \geq 0$ means that all cubes of T are reflexive in all (0) -dimensions in which they are **homogeneous**, i.e. in which the cube γ that they live above, is reflexive. Again, for $d = -1$ we are back at example 8.1.3 and then 0-discreteness just means that T is a subsingleton.

Alternatively, by similar reasoning as in example 8.1.6, one can show that a type T is 0-discrete precisely when all functions $\mathbf{y}(i : (0)) \rightarrow T$ are constant, or, if $d = -1$, when all functions $\text{Bool} \rightarrow T$ are constant.

Example 8.1.8 (Segal fibrancy). [From Nuy18b] Continuing example 2.4.35, a type $\Gamma \vdash T \text{ type @ Psh}(\text{Simplex})$ is called **Segal fibrant** or just **Segal** if $\Gamma.T \rightarrow \Gamma$ is a Segal fibration. The meaning of Segal fibrancy is surprisingly deep:

- The cells of T living above a point $\gamma : [0] \Rightarrow \Gamma$ constitute a category $T[\gamma]$, i.e. there are identity morphisms (simply from the degeneracy maps) and morphisms can be composed, and composition satisfies unit and associativity laws.
- The cells of T living above a line $\gamma : [1] \Rightarrow \Gamma$ are the elements of a profunctor $T[\gamma]$ between the categories $T[\gamma_0]$ and $T[\gamma_1]$ above the source and target of γ , i.e. they can be seen as heterogeneous morphisms from an object in $T[\gamma_0]$ to an object in $T[\gamma_1]$ which can be composed on either side with homogeneous morphisms.
- The structure of T above a triangle $\gamma : [2] \Rightarrow \Gamma$ constitutes a morphism of profunctors $T[\gamma_{12}] \circ T[\gamma_{01}] \rightarrow T[\gamma_{02}]$ between the edges of γ . A cell t

above a triangle γ witnesses that its sides t_{01} and t_{12} are sent to t_{02} by the morphism.

- The structure of T above a higher simplex asserts that the faces of the higher simplex constitute a commutative diagram of profunctor morphisms.

Example 8.1.9 (Kan fibrancy). Continuing example 2.4.36, a cubical type $\Gamma \vdash T$ type is called **Kan fibrant** or just **Kan** if $\Gamma.T \rightarrow \Gamma$ is a Kan fibration. Kan fibrant types are types that behave like proper HoTT types: their paths can be composed and their elements can be transported along paths in the context.

8.1.2 Stability of the Fibrant Replacement

The definition of an NWFS (definition 2.4.16) provides a right replacement monad \mathbf{R} . When we apply this to the arrow $\pi : \Gamma.T \rightarrow \Gamma$, we get a morphism $\mathbf{r}_\pi : \mathbf{Fac} \pi \rightarrow \Gamma$. Assuming that the CwF we are working in is locally democratic (definition 3.2.8), there is a type $\mathbf{R}T$ such that the slice \mathbf{r}_π is isomorphic to the slice $\pi : \Gamma.\mathbf{R}T \rightarrow \Gamma$. We call $\mathbf{R}T$ the **fibrant replacement** of T .

We would like to make the monad \mathbf{R} available internal to the type theory. However, it should then be stable under substitution, i.e. we want $(\mathbf{R}T)[\sigma] = \mathbf{R}(T[\sigma])$. This turns out to be far from granted.

Example 8.1.10 (Inhabitation: \blacktriangledown). Continuing examples 2.4.11, 2.4.15, 2.4.22, 2.4.28 and 8.1.2, the NWFS factors $\pi : \Gamma.T \rightarrow \Gamma$ over $[\pi, \text{id}] : (\Gamma.T) \uplus \Gamma \rightarrow \Gamma$, which is isomorphic to $\pi : \Gamma.(T \uplus \text{Unit}) \rightarrow \Gamma$. Thus, we see that the fibrant replacement monad for inhabitation is the **Maybe** monad. This monad forces a type to be inhabited by throwing in a new inhabitant. This monad acts componentwise, in the sense that $(W \triangleright (T \uplus \text{Unit})[\gamma]) \cong (W \triangleright T[\gamma]) \uplus \{\bullet\}$, so it is stable under substitution.

Example 8.1.11 (Subsingletons: \blacktriangledown). Continuing examples 2.4.3, 2.4.29 and 8.1.3, the OFS factors $\pi : \Gamma.T \rightarrow \Gamma$ over the image $\pi(\Gamma.T) \rightarrow \Gamma$, which is isomorphic to $\pi : \Gamma.\mathbf{R}T \rightarrow \Gamma$ where $\mathbf{R}T$ is the presheaf T (over \mathcal{W}/Γ) divided out by the equivalence relation ‘true’. This monad acts componentwise, so it is stable under substitution.

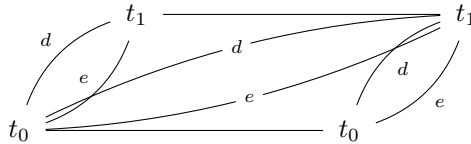
Example 8.1.12 (Codiscrete graphs: \blacktriangledown). Continuing examples 2.4.5, 2.4.31 and 8.1.4, the OFS factors π over a graph that has the nodes of $\Gamma.T$ but the edges of Γ . Thus, the codiscrete replacement $\mathbf{R}T$ has the nodes of T but a unique edge above every edge of Γ . This is stable under substitution.

Example 8.1.13 (Discrete graphs: \times). Continuing examples 2.4.6, 2.4.32 and 8.1.5, the discrete replacement $\mathbf{R}T$ is the type T divided out by the least restriction-respecting equivalence relation that identifies all homogeneous edges of T with the reflexive edge on their source.²

This is *not* stable under substitution. Indeed, consider the closed type $\vdash T$ type which contains two distinct nodes t_0, t_1 and two edges d, e from t_0 to t_1 . These edges are homogeneous, as they live above the sole edge of the empty context, which is reflexive. Therefore, the edges will be contracted in $\mathbf{R}T$, which is then the unit type / terminal graph. Substitution $\mathbf{y}\mathbb{I} \vdash (\mathbf{R}T)[()]$ type yields again the unit type. However, if we first substitute $\mathbf{y}\mathbb{I} \vdash T[()]$ type, then we get a type $T[()]$ that has two heterogeneous edges above $\text{id} : \mathbb{I} \Rightarrow \mathbf{y}\mathbb{I}$. These edges are not identified by applying \mathbf{R} .

Example 8.1.14 (0-discrete depth cubical sets: \heartsuit). Continuing examples 2.4.7, 2.4.33 and 8.1.7, the 0-discrete replacement $\mathbf{R}T$ is the type T divided out by the least restriction-respecting equivalence relation that identifies every cube with the reflexive cube on their source in every $\langle 0 \rangle$ -dimension in which it is homogeneous.

Interestingly, this operation *does* commute with substitution, as I have once laboriously proven for bridge/path cubical sets [Nuy17] and as will follow more straightforwardly from section 8.4. Consider the same closed type $\vdash T$ type as in the previous example 8.1.13, of course turning edges into $\langle 0 \rangle$ -edges and adding the necessary degenerate cubes. We still have $\mathbf{R}T \cong \text{Unit}$, but we now also have $\mathbf{R}(T[()]) \cong \text{Unit}$ in context \mathbb{I} . Indeed, we still have the two heterogeneous edges over $\text{id} : \mathbb{I} \Rightarrow \mathbf{y}\mathbb{I}$, but these edges are now diagonals of squares:



The entire left hand of the above diagram lives above $(0/i) : () \Rightarrow \mathbf{y}(i : \langle 0 \rangle)$, whereas the right hand lives above $(1/i)$. The squares live above (i/i) and are thus heterogeneous in the horizontal dimension but homogeneous in the front-to-back dimension. Therefore, they are flattened to a single horizontal line by the discrete replacement:



Both diagonals d and e then map to said line.

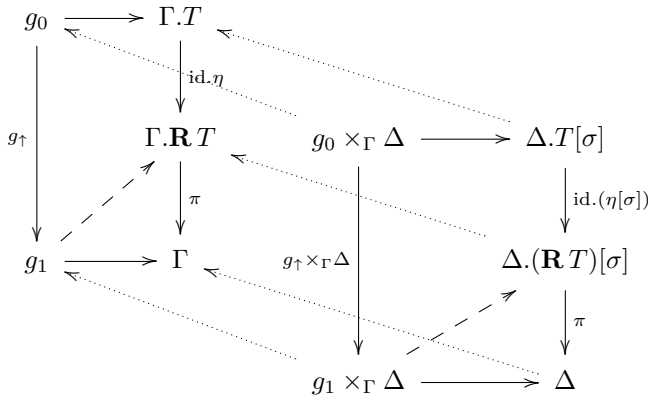
²Hence also with that on their target, as follows from restriction by $\mathbf{t} \circ \mathbf{r}$.

Example 8.1.15 (Segal fibrancy: \times). Continuing examples 2.4.35 and 8.1.8, the Segal replacement explicitly adds all composites and moreover identifies everything it needs to identify in order to ensure uniqueness of these composites.

This is absolutely not stable under substitution: consider the inclusion $\iota : \partial[2] \rightarrow \mathbf{y}[2]$ of the boundary of the triangle in the triangle. When we apply \mathbf{R} in context $\mathbf{y}[2]$, composites of edges above the triangle’s top sides will be added above the bottom side. When we apply \mathbf{R} in context $\partial[2]$, this will not happen, as it is the presence of the filler that requires existence of the composites in a Segal type.

Example 8.1.16 (Kan fibrancy: \times). A similar observation applies to Kan fibrancy (examples 2.4.36 and 8.1.9).

Robustness In each of the above cases of failure, the situation is the same: a lifting of a generating left map is explicitly added to $\mathbf{R}T$, which is then pulled back to something that is not a lifting of a left map:



Hence, it isn’t there in $\mathbf{R}(T[\sigma])$. Conversely, every lifting problem for $T[\sigma]$ is obviously also a lifting problem for T (by composition with the right hand square in the diagram above), so all the liftings added to $\mathbf{R}(T[\sigma])$ can be mapped to something in $(\mathbf{R}T)[\sigma]$. Hence, we will always have $\mathbf{R}(T[\sigma]) \rightarrow (\mathbf{R}T)[\sigma]$ but not necessarily the other way around.

The solution is to ask that pullbacks of generating left maps are again left maps, which therefore have a lifting. Moreover, the pullback square should be a morphism of left maps, in order to guarantee that the liftings match up. NWFSs generated by a left class satisfying this property, will be called **robust** (section 8.4). In the above examples, the ones that have a fibrant replacement

monad \mathbf{R} that is stable under substitution, are precisely the ones whose class of generating left maps as presented in section 2.4.6 satisfies this criterion.

Naïvely, one might hope to ‘fix’ the other examples by explicitly adding all pullbacks as *generating* left maps. A quick look at Kan fibrancy shows that this is a rather detrimental idea:

Example 8.1.17 (Teleportation). Consider the generating left map

$$\mathbf{y}(i : \mathbb{I}).(i \doteq 0) \rightarrow \mathbf{y}(i : \mathbb{I})$$

which includes the source endpoint into a path. The lifting operation against this left map is a transport operation between path-equal types.

Taking the pullback along $\mathbf{y}(i : \mathbb{I}).((i \doteq 0) \vee (i \doteq 1)) \rightarrow \mathbf{y}(i : \mathbb{I})$ yields

$$\mathbf{y}(i : \mathbb{I}).(i \doteq 0) \rightarrow \mathbf{y}(i : \mathbb{I}).((i \doteq 0) \vee (i \doteq 1)),$$

which includes a point in a set of two points. A lifting operation against this map would be a *teleport* operation between arbitrary Kan types. Under the Curry-Howard correspondence, this means that if something is true, then everything must be true, which is clearly inconsistent.

Contextual fibrancy Instead, we can resort to Boulier and Tabareau’s notion of **contextual Kan fibrancy** [BT17], which is inspired by Coquand, Huber, and Mörtberg’s *hcomp* operation [CHM18]:

Example 8.1.18 (Contextual Kan fibrancy: \checkmark). As in example 2.4.36, we introduce this notion only at a high level, as it may be defined slightly differently in every of the many treatments of cubical HoTT. We call a type $\Gamma.\Theta \vdash T$ type Kan fibrant over telescope Θ in context Γ if, for every cofibration $\mathbf{y}W.\varphi \rightarrow \mathbf{y}W$, the following diagram has a solution as depicted:

$$\begin{array}{ccc} \mathbf{y}(W, i : \mathbb{I}).(\varphi \vee (i \doteq 0)) & \longrightarrow & \Gamma.\Theta.T \\ \downarrow & \nearrow \text{dotted} & \downarrow \\ \mathbf{y}(W, i : \mathbb{I}) & \longrightarrow & \Gamma.\Theta \\ \downarrow & & \downarrow \\ \mathbf{y}W & \longrightarrow & \Gamma. \end{array}$$

In other words, T can only vary in the dimension of transport i through its dependencies on Θ , while the variables in Γ are taken to be fresh for i .

It turns out that a contextual fibrant replacement monad \mathbf{R}_Θ exists (section 8.2) and that it is stable under substitutions $\sigma : \Delta \rightarrow \Gamma$ (section 8.4). It is still unstable under substitutions $\Xi \rightarrow \Theta$, but still this allows for some internal reasoning if we take the telescope to be a single type, which can be quantified over. Example 8.1.17 no longer works as the pullback $\sigma : \Delta \rightarrow \Gamma$ can not interrupt the interval in the dimension i of transport, as the variables in Γ are fresh for i .

Note that in a type $\Gamma.\Theta \vdash T$ type that is Kan fibrant over Θ , we can still compose paths as in example 2.4.36, but only the middle one can be heterogeneous w.r.t. Γ . All three paths can still be heterogeneous w.r.t. Θ .

The above example inspires us to move to a theory of *damped* factorization systems, where we work not in the category of arrows, but in the category of damped arrows which have an additional damping morphism after the codomain. It will turn out that the robustness condition is easier to satisfy for damped NWFSs.

Example 8.1.19 (Contextual Segal fibrancy: \spadesuit). [From Nuy18b] We can similarly generalize Segal fibrancy (examples 2.4.35, 8.1.8 and 8.1.15). We call a type $\Gamma.\Theta \vdash T$ type Segal fibrant over telescope Θ in context Γ if the damped arrow $\Gamma.\Theta.T \rightarrow \Gamma.\Theta \rightarrow \Gamma$ uniquely lifts all damped arrows of the form $\Lambda_n \rightarrow \Delta_n \rightarrow \Delta_1$ where, as in example 2.4.35, the map $\Lambda_n \rightarrow \Delta_n = \mathbf{y}[n]$ includes the Hamiltonian path of n consecutive edges into the n -simplex. The damping $\Delta_n \rightarrow \Delta_1 = \mathbf{y}[1]$ can be *any* morphism; at most 1 arrow in the Hamiltonian path will be mapped to the non-trivial arrow in Δ_1 ; the others will all be collapsed at the source/target of Δ_1 . The damping square of the lifting problem just expresses that, of the n morphisms that we are trying to compose, at most one is heterogeneous w.r.t. Γ . Moreover, we require that T lifts all pullbacks of all damped arrows $\Lambda_n \rightarrow \Delta_n \rightarrow \Delta_1$. The meaning of contextual Segal fibrancy is similar to that of Segal fibrancy example 8.1.8, but the structure of T above a 2-simplex $(\gamma, \theta) : \mathbf{y}[2] \rightarrow \Gamma.\Theta$ is now merely a ‘profunctor relation’ between $T[\gamma_{12}, \theta_{12}] \circ T[\gamma_{01}, \theta_{01}]$ and $T[\gamma_{02}, \theta_{02}]$, which need only be a profunctor morphism if γ is in fact degenerate on a 1-simplex.

Note that example 8.1.15 no longer works as we are there talking about composites of 2 morphisms that are both heterogeneous w.r.t. the part Γ of the context that we are trying to be stable for.

8.1.3 Fibrancy of Π -types

In this section, we are concerned with the question: what does A need to satisfy so that ΠA_{\perp} preserves fibrancy? The ideal answer is, of course, nothing, and

we will see that this is sometimes but not always the case. Second best would be that A needs to be fibrant, but even that is not granted.

Example 8.1.20 (Inhabitation: \checkmark). Continuing example 8.1.2, clearly ΠAB is inhabited if B is.

Example 8.1.21 (Subsingletons: \checkmark). Continuing example 8.1.3, it follows immediately from the presheaf construction of the Π -type (section 4.3.1) that ΠAB is a subsingleton if B is.

Example 8.1.22 (Codiscrete graphs: \checkmark). Continuing example 8.1.4, in order to prove codiscreteness of ΠAB , we need to show that, given cells $\mathbb{N} \triangleright f_0 : (\Pi AB)[\gamma \circ \mathbf{s}]$ and $\mathbb{N} \triangleright f_1 : (\Pi AB)[\gamma \circ \mathbf{t}]$, there is a unique edge f from f_0 to f_1 above γ . Recall from example 4.3.2 that f consists of 5 actions: on nodes/edges at the source/target of γ , and on edges above γ . The first 4 are fixed as f_0 and f_1 are given. The fifth is unique by codiscreteness of B .

Example 8.1.23 (Discrete graphs: \times). Continuing examples 2.4.6, 2.4.32, 8.1.5 and 8.1.13, we need to show that every homogeneous edge in ΠAB is reflexive. So pick a homogeneous edge $\mathbb{I} \triangleright f : (\Pi AB)[\gamma \circ \mathbf{r}]$ above the node γ . If we can show that $f\langle \mathbf{s} \rangle = f\langle \mathbf{t} \rangle$ then that's a good start. Write $f = \lambda b$, so $\mathbf{y}\mathbb{I}.A[\gamma \circ \mathbf{r}] \vdash b : B[(\gamma \circ \mathbf{r})+]$. For every node $\mathbb{N} \triangleright a : A[\gamma]$, we can apply f to the reflexive edge $a\langle \mathbf{r} \rangle$ and get an edge $\mathbb{I} \triangleright b[\text{id}, a\langle \mathbf{r} \rangle] : B[\gamma \circ \mathbf{r}, a\langle \mathbf{r} \rangle]$. This edge is homogeneous and therefore reflexive by discreteness of B , equating its source $b[\mathbf{s}, a]$ and target $b[\mathbf{t}, a]$. But these are precisely the actions of $f\langle \mathbf{s} \rangle$ and $f\langle \mathbf{t} \rangle$ on a , so the source and target functions have the same action on nodes.

So far so good. In order to prove by the same reasoning that the source and target functions have the same action on edges, we would like to apply f to the reflexive square on a given edge. But reflexive graphs have no squares.

In fact, we can exploit this to give a counterexample. Consider the type $\mathbf{y}\mathbb{I} \vdash T$ type which has a single node $\mathbb{N} \triangleright t_0 : T[\mathbf{s}]$ above the source of $\mathbf{y}\mathbb{I}$ and a single node $\mathbb{N} \triangleright t_1 : T[\mathbf{t}]$ above the target, as well as three distinct edges $\mathbb{I} \triangleright c, d, e : T[\text{id}_{\mathbb{I}}]$ from t_0 to t_1 . This type is discrete: the only homogeneous edges are $t_0\langle \mathbf{r} \rangle$ and $t_1\langle \mathbf{r} \rangle$, which are indeed reflexive.

We can then give two nodes $\mathbb{N} \triangleright f_c, f_e : (\Pi(\mathbf{y}\mathbb{I})T)[()]$ which map the non-trivial edge in $\mathbf{y}\mathbb{I}$ to c and e respectively. An edge from f_c to f_e is an action on edges in $\mathbf{y}\mathbb{I}$. The following is such an action:

$$\mathbf{s} \circ \mathbf{r} \mapsto t_0\langle \mathbf{r} \rangle, \quad \mathbf{t} \circ \mathbf{r} \mapsto t_1\langle \mathbf{r} \rangle, \quad \text{id}_{\mathbb{I}} \mapsto d.$$

So we see that we are not only lacking a reflexive square on the non-trivial edge $\text{id}_{\mathbb{I}}$ of $\mathbf{y}\mathbb{I}$ whose image would witness that $c = e$, we are also left to deal with a dangling diagonal of that square whose image d can be yet another edge!

Example 8.1.24 (0-discrete depth cubical sets: \checkmark). Continuing examples 2.4.7, 2.4.33, 8.1.7 and 8.1.14, in $\text{Psh}(\text{DCube}_d)$ for $d \geq 0$ we can finish the reasoning in the above example as we do have squares available. This shows that all homogeneous edges in ΠAB are loops, but we still have to show that they are reflexive. We know that the action of a cube $(W, i : \langle 0 \rangle) \triangleright f : (\Pi AB)[\gamma]$ on cubes $(W, i : \langle 0 \rangle) \triangleright a : A[\gamma]$ sends them to cubes that are reflexive in dimension i . However, we also have to take into account the action on cubes $(V, i : \langle k \rangle) \triangleright a : A[\gamma \circ (\varphi, i/i)]$ for all $\varphi : (V, i : \langle k \rangle) \rightarrow W$.³ But these must be compatible with the action of f above γ and are therefore also reflexive.

We can diagrammatically summarize this reasoning *very* roughly as follows: given an edge f in the Π -type, we need to prove that it is reflexive.

$$f_0 \xrightarrow{f} f_1 \quad : \quad a_0 \xrightarrow{a} a_1 \quad \mapsto \quad b_0 \xrightarrow{b} b_1$$

This means that it acts on cells the same way as $f_0 \langle \mathbf{r} \rangle$. So pick a $(W, i : \langle k \rangle)$ -cell $a_0 = a_1$ on which f and $f_0 \langle \mathbf{r} \rangle$ may act. This cell has a reflexive edge $a = a_0 \langle \mathbf{r} \rangle$, to which we apply f , yielding an edge b . This edge is homogeneous and therefore reflexive by discreteness of B . This means that its source (which is $f_0 \langle \mathbf{r} \rangle$ applied to a_0) and its diagonal (which is f applied to a_0) are equal.

Alternatively, we can use the criterion that ΠAB is 0-discrete if and only if all functions $\mathbf{y}(i : \langle 0 \rangle) \rightarrow \Pi AB$ are constant, which follows straightforwardly from 0-discreteness of B by swapping arguments. However, the more low-level approach allows a more interesting comparison with the other examples.

As usual, for $d = -1$, we are back in example 8.1.21.

Example 8.1.25 (Clock-irrelevance: \checkmark). Continuing examples 2.4.34 and 8.1.6, the fact that ΠAB is clock-irrelevant if B is follows straightforwardly by swapping the arguments of type A and \oplus .

Example 8.1.26 (Kan fibrancy: if A is Kan). Continuing examples 2.4.36, 8.1.9 and 8.1.16, let us try to prove a corollary of Kan fibrancy: that ΠAB lifts $\sqcup \rightarrow \square$, where $\square = \mathbf{y}(j : \mathbb{I}, i : \mathbb{I})$ and $\sqcup = \square.((j \doteq 0) \vee (j \doteq 1) \vee (i \doteq 0))$. To this end, pick a \sqcup -shape $\sqcup \vdash f_{\sqcup} : (\Pi AB)[\gamma_{\sqcup}]$ which lives above the restriction γ_{\sqcup} of $\gamma : \square \rightarrow \Gamma$ and has sides $f_{0/j}, f_{0/i}$ and $f_{1/j}$. We will try to create a composite $f_{1/i}$:

$$\begin{array}{ccc}
 \begin{array}{ccc} \bullet & \xrightarrow{f_{1/i}} & \bullet \\ f_{0/j} \Big| & & \Big| f_{1/j} \\ \bullet & \xrightarrow{f_{0/i}} & \bullet \end{array} & : & \begin{array}{ccc} \bullet & \xrightarrow{a_{1/i}} & \bullet \\ a_{0/j} \Big| & & \Big| a_{1/j} \\ \bullet & \xrightarrow{a_{0/i}} & \bullet \end{array} & \mapsto & \begin{array}{ccc} \bullet & \xrightarrow{b_{1/i}} & \bullet \\ b_{0/j} \Big| & & \Big| b_{1/j} \\ \bullet & \xrightarrow{b_{0/i}} & \bullet \end{array}
 \end{array}$$

³For morphisms that actually substitute $0/i$ or $1/i$, we fall back to the actions of the source and target of f which we already know are equal.

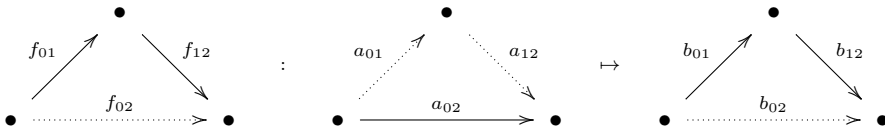
This composite should act on edges $a_{1/i}$ living above $\gamma_{1/i}$.

If we assume that A is also Kan, then we can transport $a_{1/i}$ the other way and get a full square a which restricts to a_{\sqcup} . Applying f_{\sqcup} to a_{\sqcup} yields b_{\sqcup} , which can be composed to $b_{1/i}$ by Kan fibrancy of B .

This reasoning is a simplification of how Kan fibrancy of the Π -type is proven in (models of) cubical type theory [Coh+17].

Note that when we compare to category theory, it is in a way unexpected that the domain needs to be Kan fibrant. Understanding Kan fibrancy as having a composition operation, the categorical counterpart of a non-Kan type is a symmetric reflexive graph. Now the collection of symmetric reflexive graph morphisms $\mathcal{H} \rightarrow \mathcal{G}$ from an arbitrary symmetric reflexive graph \mathcal{H} to a groupoid \mathcal{G} is already a groupoid; we have no need for a composition operation on \mathcal{H} to achieve this result.

Example 8.1.27 (Segal fibrancy: if A is Conduché). Continuing examples 2.4.35, 8.1.8 and 8.1.15, let us try to prove a corollary of Segal fibrancy: that ΠAB lifts $\Delta_2 \rightarrow \Delta_2$, i.e. has all composites of two heterogeneous morphisms. To this end, pick two morphisms f_{01} and f_{12} above the top edges γ_{01} and γ_{12} of a triangle $\gamma : \Delta_2 \Rightarrow \Gamma$, we will try to create a composite f_{02} :



This composite needs to act on arrows a_{02} above γ_{02} . If we can somehow factor a_{02} into arrows a_{01} and a_{12} living above γ_{01} and γ_{12} , then we can apply f_{01} and f_{12} to those and obtain b_{01} and b_{12} , which compose to b_{02} by Segal fibrancy of B .

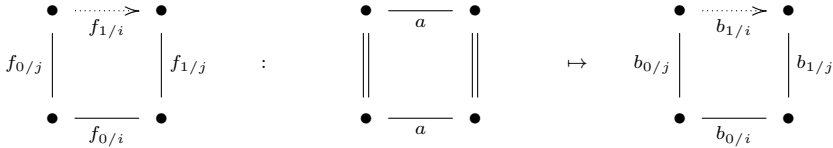
However, the factorization of a_{02} does not follow from Segal fibrancy. Instead, the precise criterion that $\pi : \Gamma.A \rightarrow \Gamma$ should satisfy is known as the Conduché condition [Gir64; nLa20b]. Thus, we can conclude that ΠAB is Segal if A is Conduché and B is Segal.

Unfortunately, the Conduché condition is rather bizarre and is likely invented with the sole purpose of making the Π -type fibrant. As such, Conduché fibrancy is not so much a solution as a precise description of the problem.

Again when we compare to category theory, a condition on the domain is in a way unexpected. Since a category can be defined as a simplicial set satisfying the Segal condition, the categorical counterpart of a non-Segal type is just a

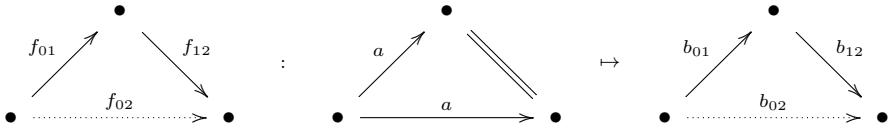
simplicial set. Now the collection of simplicial set morphisms $\mathcal{D} \rightarrow \mathcal{C}$ from an arbitrary simplicial set \mathcal{D} to a category \mathcal{C} is already a category; we have no need for a composition operation on \mathcal{D} to achieve this result.

Example 8.1.28 (Contextual Kan fibrancy: \blacktriangleright). Continuing example 8.1.18, let us ask the question when $\Gamma.\Theta \vdash \Pi AB$ type is Kan fibrant over Θ , *assuming that A only depends on Γ* . In that case, we do not need Kan composition for A as we can instead take a degenerate square:

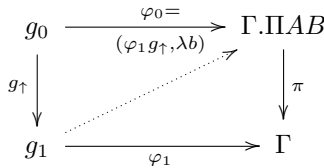


Indeed, if a lives above the source edge of a degenerate square in Γ , then it also lives above the target edge.

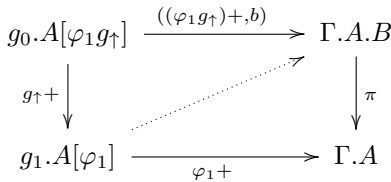
Example 8.1.29 (Contextual Segal fibrancy: \blacktriangleright). Continuing example 8.1.18 and again assuming A depends only on Γ , we similarly do not need Conduché fibrancy of A as we can simply take a degenerate triangle. Assume that of the arrows f_{01} and f_{12} which we are trying to compose, f_{01} is the one that is heterogeneous w.r.t. Γ . Then we can do:



We will see that the same robustness criterion from section 8.1.2 will ensure that ΠA preserves fibrancy. Indeed, a lifting problem



can be written equivalently as:



and the left arrow of the second problem is a pullback of \vec{g} , so if that arrow too is a left map, then there will be a lifting.

8.2 Damped Factorization Systems

In this section, we straightforwardly generalize the content of sections 2.4.5 and 2.4.6 to the damped situation, so that we can consider contextual fibrancy. All proofs carry over from sections 2.4.5 and 2.4.6, as the dampings do not interfere with the proofs given there, nor with the omitted duals of these proofs (i.e. codampings do not interfere either). We refer to section 2.4 for the motivation behind all these definitions and theorems.

Definition 8.2.1. The **walking damped arrow** $\uparrow\downarrow$ is the category with three objects 0, 1 and 2 and two non-identity morphisms $0 \rightarrow 1 \rightarrow 2$.

Definition 8.2.2. The **damped arrow category** of \mathcal{C} is defined to be the functor category $\mathcal{C}^{\uparrow\downarrow}$. It has as objects quintuples $\vec{x} = ((x_0 \xrightarrow{x_\uparrow} x_1 \xrightarrow{x_\downarrow} x_2))$ and as morphisms natural transformations $\vec{\varphi} = (\varphi_0, \varphi_1, \varphi_2)$. It is equipped with obvious functors $\text{Dom}, \text{Cod}, \text{Dmp} : \mathcal{C}^{\uparrow\downarrow} \rightarrow \mathcal{C}$ and obvious natural transformations $\text{mor} : \text{Dom} \rightarrow \text{Cod}$ and $\text{dmp} : \text{Cod} \rightarrow \text{Dmp}$.

Definition 8.2.3. Two damped arrows $\vec{\ell}, \vec{r} \in \mathcal{C}^{\uparrow\downarrow}$ have the **left/right lifting property** w.r.t. each other (denoted $\vec{\ell} \uparrow \vec{r}$) if, for every lifting problem $\vec{\varphi} : \vec{\ell} \rightarrow \vec{r}$, there exists *some* solution $\sigma : \ell_1 \rightarrow r_0$ making the following diagram commute:

$$\begin{array}{ccc}
 \ell_0 & \xrightarrow{\varphi_0} & r_0 \\
 \ell_\uparrow \downarrow & \nearrow \sigma & \downarrow r_\uparrow \\
 \ell_1 & \xrightarrow{\varphi_1} & r_1 \\
 \ell_\downarrow \downarrow & & \downarrow r_\downarrow \\
 \ell_2 & \xrightarrow{\varphi_2} & r_2
 \end{array}$$

We also write $\vec{\ell} \uparrow \vec{r}$ to denote the set of lifting operations producing such a solution for every $\vec{\varphi}$.

Definition 8.2.4. A **damped NWFS** on a category \mathcal{C} consists of the following data:

- A functor $\text{Fac} : \mathcal{C}^{\uparrow\downarrow} \rightarrow \mathcal{C}$ and natural transformations $\ell : \text{Dom} \rightarrow \text{Fac}$ and $\mathbf{r} : \text{Fac} \rightarrow \text{Cod}$ such that $\mathbf{r} \circ \ell = \text{mor}$.

- Categories \mathcal{L} and \mathcal{R} of left/right maps.
- Two adjoint functors $C_{\mathbf{L}} \dashv U_{\mathbf{L}}$ where $U_{\mathbf{L}} : \mathcal{L} \rightarrow \mathcal{C}^{\uparrow\downarrow}$.

If $\ell \in \mathcal{L}$, then we write $\vec{\ell} = (\ell_0 \xrightarrow{\ell^{\uparrow\downarrow}} \ell_1 \xrightarrow{\ell^{\downarrow}} \ell_2) := U_{\mathbf{L}} \ell$.

We write $\mathbf{L} = U_{\mathbf{L}} C_{\mathbf{L}}$ for the composite **left coreplacement** comonad, $\eta^{\mathbf{L}} : \text{Id}_{\mathcal{L}} \rightarrow C_{\mathbf{L}} U_{\mathbf{L}}$ for the unit, and $\varepsilon^{\mathbf{L}} : \mathbf{L} = U_{\mathbf{L}} C_{\mathbf{L}} \rightarrow \text{Id}_{\mathcal{C}^{\uparrow}}$ for the co-unit.

We require

$$\begin{aligned} \mathbf{L} \vec{x} &= (x_0 \xrightarrow{\ell_{\vec{x}}} \text{Fac } \vec{x} \xrightarrow{x_{\downarrow} \circ r_{\vec{x}}} x_2), \\ \varepsilon_{\vec{x}}^{\mathbf{L}} &= (\text{id}_{x_0}, r_{\vec{x}}, \text{id}_{x_2}) && : \mathbf{L} \vec{x} \rightarrow \vec{x}, \\ U_{\mathbf{L}} \eta_{\ell}^{\mathbf{L}} &= (\text{id}_{\ell_0}, \eta_{\ell}^{\mathbf{L}}, \text{id}_{\ell_2}) && : \vec{\ell} \rightarrow \mathbf{L} \vec{\ell}, \end{aligned}$$

where $\eta^{\mathbf{L}} : \text{Cod } U_{\mathbf{L}} \rightarrow \text{Fac } U_{\mathbf{L}}$.

- Two adjoint functors $U_{\mathbf{R}} \dashv F_{\mathbf{R}}$ where $U_{\mathbf{R}} : \mathcal{R} \rightarrow \mathcal{C}^{\uparrow}$.

If $\hat{r} \in \mathcal{R}$, then we write $\vec{r} = (r_0 \xrightarrow{r^{\uparrow}} r_1 \xrightarrow{r^{\downarrow}} r_2) := U_{\mathbf{R}} \hat{r}$.

We write $\mathbf{R} = U_{\mathbf{R}} F_{\mathbf{R}}$ for the composite **right replacement** monad, $\bar{\eta}^{\mathbf{R}} : \text{Id}_{\mathcal{C}^{\uparrow\downarrow}} \rightarrow U_{\mathbf{R}} F_{\mathbf{R}} = \mathbf{R}$ for the unit, and $\varepsilon^{\mathbf{R}} : F_{\mathbf{R}} U_{\mathbf{R}} \rightarrow \text{Id}_{\mathcal{L}}$ for the co-unit.

We require

$$\begin{aligned} \mathbf{R} \vec{x} &= (\text{Fac } \vec{x} \xrightarrow{r_{\vec{x}}} x_1 \xrightarrow{x_{\downarrow}} x_2), \\ \bar{\eta}_{\vec{x}}^{\mathbf{R}} &= (\ell_{\vec{x}}, \text{id}_{x_1}, \text{id}_{x_2}) && : \vec{x} \rightarrow \mathbf{R} \vec{x}, \\ U_{\mathbf{R}} \varepsilon_{\hat{r}}^{\mathbf{R}} &= (\varepsilon_{\hat{r}}^{\mathbf{R}}, \text{id}_{r_1}, \text{id}_{r_2}) && : \mathbf{R} \vec{r} \rightarrow \vec{r}, \end{aligned}$$

where $\varepsilon^{\mathbf{R}} : \text{Fac } U_{\mathbf{R}} \rightarrow \text{Dom } U_{\mathbf{R}}$.

These must satisfy the following conditions:

- Left maps are precisely maps equipped with a natural lifting operation w.r.t. all right maps and vice versa, in the following sense:⁴

$$\begin{aligned} \mathcal{L} &\cong \int_{\vec{x} \in \mathcal{C}^{\uparrow\downarrow}} \forall (\hat{r} \in \mathcal{R}). (\vec{x} \pitchfork \vec{r}), \\ \mathcal{R} &\cong \int_{\vec{x} \in \mathcal{C}^{\uparrow\downarrow}} \forall (\ell \in \mathcal{L}). (\vec{\ell} \pitchfork \vec{x}), \end{aligned}$$

⁴Recall that we take categories of elements over functors from the twisted arrow category (definition 2.2.37).

and in such a way that the operators associated to $\vec{\ell}$ and \hat{r} yield the same element of $\vec{\ell} \hat{\cap} \vec{r}$. Of course the carriers in the right-hand categories should be obtained via $U_{\mathbf{L}}$ and $U_{\mathbf{R}}$.

- For any $\vec{\ell} \in \mathcal{L}$, the canonical solution to the lifting problem $\vec{\eta}_{U_{\mathbf{L}}\vec{\ell}}^{\mathbf{R}} : U_{\mathbf{L}}\vec{\ell} \rightarrow \mathbf{R}U_{\mathbf{L}}\vec{\ell}$ is $\eta_{\vec{\ell}}^{\mathbf{L}} := \text{Cod}(U_{\mathbf{L}}\eta_{\vec{\ell}}^{\mathbf{L}}) : \ell_1 \rightarrow \text{Fac}\vec{\ell}$.
- For any $\hat{r} \in \mathcal{R}$, the canonical solution to the lifting problem $\vec{\varepsilon}_{U_{\mathbf{R}}\hat{r}}^{\mathbf{L}} : \mathbf{L}U_{\mathbf{R}}\hat{r} \rightarrow U_{\mathbf{R}}\hat{r}$ is $\varepsilon_{\hat{r}}^{\mathbf{R}} := \text{Dom}(U_{\mathbf{R}}\varepsilon_{\hat{r}}^{\mathbf{R}}) : \text{Fac}\vec{r} \rightarrow r_0$.

$$\begin{array}{ccc}
 \ell_0 & \xrightarrow{\ell_{\vec{\ell}}} & \text{Fac}\vec{\ell} \\
 \ell_{\uparrow} \downarrow & \nearrow \eta_{\vec{\ell}}^{\mathbf{L}} & \downarrow r_{\vec{\ell}} \\
 \ell_1 & \xrightarrow{\text{id}} & \ell_1 \\
 \ell_{\downarrow} \downarrow & & \downarrow \ell_{\downarrow} \\
 \ell_2 & \xrightarrow{\text{id}} & \ell_2
 \end{array}
 \qquad
 \begin{array}{ccc}
 r_0 & \xrightarrow{\text{id}} & r_0 \\
 \ell_{\vec{r}} \downarrow & \nearrow \varepsilon_{\hat{r}}^{\mathbf{R}} & \downarrow r_{\uparrow} \\
 \text{Fac}\vec{r} & \xrightarrow{r_{\vec{r}}} & r_1 \\
 r_{\downarrow} \downarrow & & \downarrow r_{\downarrow} \\
 r_2 & \xrightarrow{\text{id}} & r_2
 \end{array}$$

Definition 8.2.5. Given a damped NWFS on a CwF \mathcal{C} , we say that a type $\Gamma.\Theta \vdash T$ type is **contextually fibrant** in context Γ if $(\Gamma.\Theta.T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma)$ is a right map, i.e. it is equipped with an Eilenberg-Moore algebra structure f for the right/fibrant replacement monad $\mathbf{R} : \mathcal{C}^{\uparrow\downarrow} \rightarrow \mathcal{C}^{\uparrow\downarrow}$.

If Θ is empty, we say that $\Gamma \vdash T$ type is **degenerately fibrant** [BT17].

Lemma 8.2.6. The adjoint factorization $\mathbf{R} = U_{\mathbf{R}}F_{\mathbf{R}}$ is isomorphic to the adjoint factorization over the Eilenberg-Moore category $\text{EM}(\mathbf{R})$. The dual result holds for \mathbf{L} .

Proof. See lemma 2.4.18. □

Lemma 8.2.7. In a damped NWFS, the solution to a lifting problem $\vec{\varphi} : U_{\mathbf{L}}\vec{\ell} \rightarrow U_{\mathbf{R}}\hat{r}$ is given by $\varepsilon_{\hat{r}}^{\mathbf{R}} \circ \text{Fac}\vec{\varphi} \circ \eta_{\vec{\ell}}^{\mathbf{L}}$.

Proof. See lemma 2.4.19. □

Definition 8.2.8. A **damped Grandis-Tholen NWFS** consists of:

- A right replacement monad $(\mathbf{R}, \vec{\eta}^{\mathbf{R}}, \vec{\mu}^{\mathbf{R}})$ on $\mathcal{C}^{\uparrow\downarrow}$ which is trivial at $\text{Cod}, \text{Dmp}, \text{dmp}$,
- A left coreplacement comonad $(\mathbf{L}, \vec{\varepsilon}^{\mathbf{L}}, \vec{\delta}^{\mathbf{L}})$ on $\mathcal{C}^{\uparrow\downarrow}$ which is trivial at $\text{Dom}, \text{Dmp}, \text{dmp} \circ \text{mor}$,

such that

- $\text{Dom } \mathbf{R} = \text{Cod } \mathbf{L} =: \text{Fac}$,
- $\text{mor } \mathbf{L} = \text{Dom } \vec{\eta}^{\mathbf{R}} =: \ell : \text{Dom} \rightarrow \text{Fac}$,
- $\text{mor } \mathbf{R} = \text{Cod } \vec{\varepsilon}^{\mathbf{L}} =: \mathbf{r} : \text{Fac} \rightarrow \text{Cod}$.

Theorem 8.2.9. A damped Grandis-Tholen NWFS determines a unique damped NWFS (up to isomorphism).

Proof. See theorem 2.4.21. Again $\mathcal{L} = \text{EM}(\mathbf{L})$ and $\mathcal{R} = \text{EM}(\mathbf{R})$. □

Proposition 8.2.10. A damped OFS (of which we omit the definition) can be equivalently defined as an NWFS for which \mathbf{L} and \mathbf{R} are idempotent. The classes of the NWFS are then automatically full subcategories of $\mathcal{C}^{\uparrow\downarrow}$.

Proof. See proposition 2.4.23. □

Theorem 8.2.11 (Small object argument). Write $\mathcal{C} = \text{Psh}(\mathcal{W})$. Assume given a category \mathcal{G} of **generating left maps** equipped with a functor $U_{\mathcal{G}} : \mathcal{G} \rightarrow \mathcal{C}^{\uparrow\downarrow}$. If $\check{g} \in \mathcal{G}$, we write $\vec{g} := U_{\mathcal{G}}\check{g}$.

Then there exists a **left generated** damped NWFS on \mathcal{C} such that

$$\mathcal{R} = \int_{\vec{x} \in \mathcal{C}^{\uparrow\downarrow}} \forall \check{g} \in \mathcal{G}. \vec{g} \pitchfork \vec{x},$$

$$\mathcal{L} = \int_{\vec{x} \in \mathcal{C}^{\uparrow\downarrow}} \forall \check{r} \in \mathcal{R}. \vec{x} \pitchfork \vec{r}.$$

We write $I_{\mathcal{G}}$ for the canonical functor $\mathcal{G} \rightarrow \mathcal{L}$ and $\acute{g} := I_{\mathcal{G}}\check{g}$. We have $U_{\mathbf{L}} \circ I_{\mathcal{G}} = U_{\mathcal{G}}$.

Proof. See theorem 2.4.25. □

Remark 8.2.12. If all the generating left maps are damped by the identity $g_{\downarrow} = \text{id} : g_1 \rightarrow g_2$, then going through the proofs above, we find that we get the exact same \mathbf{L} and \mathbf{R} as for the corresponding non-damped NWFS. Thus, the left/right maps are the left/right maps of the non-damped NWFS, with arbitrary dampings, since these dampings are immaterial when you're trying to be a (co)algebra for a (co)monad that ignores them.

In particular, for types, the position of the semicolon in $\Gamma; \Theta \vdash (T, f) \text{ fib}$ becomes immaterial.

8.3 Stability

In this chapter, we are after a type operator

$$\frac{\Gamma.\Theta \vdash T \text{ type}}{\Gamma.\Theta \vdash \mathbf{R}T \text{ type}} \quad (8.1)$$

where $(\Gamma.\Theta.\mathbf{R}_\Theta T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma)$ should be isomorphic to the fibrant replacement $\mathbf{R}(\Gamma.\Theta.T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma)$. (In the non-damped case, remove Θ .)

If \mathbf{R} is to become an internal type operator, then we would like it to be stable under substitution.

Proposition 8.3.1. In any (damped) NWFS, (contextual) fibrancy is preserved under substitution (w.r.t. both parts of the context).

Note that non-fibrancy is not preserved.

Proof. Note that type substitution takes a pullback. The solution to the following lifting problem factors through the pullback:

$$\begin{array}{ccccc}
 g_0 & \longrightarrow & \Delta.\Xi.T[\sigma.\tau] & \xrightarrow{\sigma.\tau+} & \Gamma.\Theta.T \\
 \downarrow & & \downarrow \pi & \nearrow \text{dotted} & \downarrow \pi \\
 g_1 & \longrightarrow & \Delta.\Xi & \xrightarrow{\sigma.\tau} & \Gamma.\Theta \\
 \downarrow & & \downarrow \pi & & \downarrow \pi \\
 g_2 & \longrightarrow & \Delta & \xrightarrow{\sigma} & \Gamma
 \end{array}$$

In non-damped NWFSs, the situation simplifies. □

The above result implies that there will always be a function $\mathbf{R}(T[\sigma]) \rightarrow (\mathbf{R}T)[\sigma]$, since we have $\eta^{\mathbf{R}}[\sigma] : T[\sigma] \rightarrow (\mathbf{R}T)[\sigma]$ and $(\mathbf{R}T)[\sigma]$ is fibrant. We would like this function to be an isomorphism or, ideally, the identity.

Recall that \mathcal{C}^\uparrow and $\mathcal{C}^{\uparrow\setminus}$ are functor categories. A cartesian natural transformation (definition 2.2.24) is one whose naturality squares are pullback squares. As such, a morphism in \mathcal{C}^\uparrow is a cartesian natural transformation if it is a pullback square, whereas a morphism in $\mathcal{C}^{\uparrow\setminus}$ is one if it is *two* pullback squares.

Definition 8.3.2. We call a (damped) NWFS on \mathcal{C} **stable** if the monad \mathbf{R} sends cartesian natural transformations to cartesian natural transformations.

If \mathcal{C} is a CwF, we say that the (damped) NWFS is **strictly stable** if there is a type operator as in eq. (8.1) such that $(\Gamma.\Theta.\mathbf{R}_\Theta T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma) \cong \mathbf{R}(\Gamma.\Theta.T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma)$ (in the non-damped case, remove Θ) which preserves substitution on the nose.

In the non-damped case, note that type substitution takes a pullback; hence if \mathbf{R} preserves pullbacks, it preserves type substitution at least up to isomorphism:

$$\begin{array}{ccc} \Delta.T[\sigma] \xrightarrow{\sigma^+} \Gamma.T & & \Delta.\mathbf{R}(T[\sigma]) \xrightarrow{\sigma^+} \Gamma.\mathbf{R}T \\ \pi \downarrow & & \pi \downarrow \\ \Delta & \xrightarrow{\sigma} & \Gamma \end{array} \quad \begin{array}{ccc} \Delta.\mathbf{R}(T[\sigma]) \xrightarrow{\sigma^+} \Gamma.\mathbf{R}T & & \Delta.\mathbf{R}(T[\sigma]) \xrightarrow{\sigma^+} \Gamma.\mathbf{R}T \\ \pi \downarrow & & \pi \downarrow \\ \Delta & \xrightarrow{\sigma} & \Gamma \end{array}$$

In damped NWFSs, we can read a damped arrow as a chain $\Gamma.\Theta.T \rightarrow \Gamma.\Theta \rightarrow \Gamma$ and the stability condition only guarantees that substitutions in the first part of the context are preserved:

$$\begin{array}{ccc} \Delta.\Theta[\sigma].T[\sigma^+] \xrightarrow{\sigma^{++}} \Gamma.\Theta.T & & \Delta.\Theta[\sigma].\mathbf{R}_\Theta(T[\sigma]) \xrightarrow{\sigma^{++}} \Gamma.\Theta.\mathbf{R}_\Theta T \\ \pi \downarrow & & \pi \downarrow \\ \Delta.\Theta[\sigma] \xrightarrow{\sigma^+} \Gamma.\Theta & & \Delta.\Theta[\sigma] \xrightarrow{\sigma^+} \Gamma.\Theta \\ \pi \downarrow & & \pi \downarrow \\ \Delta & \xrightarrow{\sigma} & \Gamma \end{array} \quad \begin{array}{ccc} \Delta.\Theta[\sigma].\mathbf{R}_\Theta(T[\sigma]) \xrightarrow{\sigma^{++}} \Gamma.\Theta.\mathbf{R}_\Theta T & & \Delta.\Theta[\sigma].\mathbf{R}_\Theta(T[\sigma]) \xrightarrow{\sigma^{++}} \Gamma.\Theta.\mathbf{R}_\Theta T \\ \pi \downarrow & & \pi \downarrow \\ \Delta.\Theta[\sigma] \xrightarrow{\sigma^+} \Gamma.\Theta & & \Delta.\Theta[\sigma] \xrightarrow{\sigma^+} \Gamma.\Theta \\ \pi \downarrow & & \pi \downarrow \\ \Delta & \xrightarrow{\sigma} & \Gamma \end{array}$$

8.4 Robust NWFSs

Motivated by our findings in sections 8.1.2 and 8.1.3, we define robustness:

Definition 8.4.1. [Nuy18b] We call a (damped) NWFS over $\mathcal{C} = \text{Psh}(\mathcal{W})$ **robust** if it is left generated by \mathcal{G} and every cartesian natural transformation $\vec{\varphi} : \vec{x} \rightarrow \vec{g}$ to a generating left map $\vec{g} \in \mathcal{G}$ is itself a morphism of left maps (i.e.

in the image of $U_{\mathbf{L}}$), and such that every square of natural transformations

$$\begin{array}{ccc} \vec{x} & \xrightarrow[\text{cart.}]{\vec{\varphi}} & \vec{g} \\ \vec{\chi} \downarrow & & \vec{\gamma} \downarrow \\ \vec{x}' & \xrightarrow[\text{cart.}]{\vec{\varphi}'} & \vec{g}' \end{array} \quad (8.2)$$

with right side $\vec{\gamma} : \vec{g} \rightarrow \vec{g}'$ a morphism of generating left maps, is correspondingly a commutative square of left maps.

What this means in practice is that pullbacks of generating left maps will also be lifted by fibrant types, and indeed naturally so.

8.4.1 Constructing Robust NWFSs

The following propositions provide two ways to obtain robust (damped) NWFSs. We will never use the former but it seems worth mentioning.

Proposition 8.4.2. A (damped) NWFS is robust if every cartesian natural transformation $\varphi : \vec{x} \rightarrow \vec{\ell}$ to a left map $\vec{\ell} \in \mathcal{G}$ is itself a morphism of left maps, and similar for squares analogous to eq. (8.2).

Proof. It follows from the definition of a (damped) NWFS that it is trivially generated by \mathcal{L} . Thus, it satisfies the definition of robustness. \square

Proposition 8.4.3. A left generated (damped) NWFS over a presheaf category is robust if every cartesian natural transformation $\varphi : \vec{x} \rightarrow \vec{g}$ to a generating left map $\vec{g} \in \mathcal{G}$ with x_1 (damped: x_2) representable, is itself a morphism of left maps, and similar for squares arising from a morphism between \vec{g} 's and a morphism in the base category.

Note that if x_i (where $i = 1$ in the non-damped case and $i = 2$ in the damped case) is representable, i.e. $x_i = \mathbf{y}W$, then $\varphi_i : x_i \rightarrow g_i$ is essentially a cell $W \Rightarrow g_i$. So the criterion says that pullbacks (not necessarily base pullbacks!) of generating left maps along presheaf cells, should be left maps again.

Proof. We prove this for damped NWFSs. Pick an arbitrary cartesian natural transformation $\vec{\varphi} : \vec{y} \rightarrow \vec{g}$. We have to show that \vec{y} solves any lifting problem $\vec{\chi} : \vec{y} \rightarrow \vec{r}$ with $\vec{r} \in \mathcal{R}$, i.e. we need to define $\sigma : y_1 \rightarrow r_0$. For every cell $\gamma : W \Rightarrow y_1$, we get a morphism $y_\vee \circ \gamma : \mathbf{y}W \rightarrow y_2$ along which we can take a

pullback that we call $\psi : \vec{z} \rightarrow \vec{y}$ and, by the universal property of the pullback, a cell $\zeta : W \Rightarrow z_1$ such that $\psi_1 \circ \zeta = \gamma$. Now \vec{z} is lifted by \vec{r} , and naturally so in (W, γ) . This constitutes a presheaf morphism $\sigma : y_1 \rightarrow r_0$. It is straightforward to verify that σ is a solution to the lifting problem. \square

8.4.2 Robustness of Examples of NWFSs

Example 8.4.4 (Surjective functions: \blacktriangleright). Continuing examples 2.4.22, 2.4.28, 8.1.2, 8.1.10 and 8.1.20, any pullback of the sole generating left map $\emptyset \rightarrow \{\bullet\}$ (example 2.4.28) is again an injective function (i.e. left map) $\emptyset \rightarrow X$, and moreover the pullback square is a morphism of left maps in the sense of example 2.4.22. Thus, this NWFS is robust.

Example 8.4.5 (Injective presheaf morphisms: \blacktriangleright). Continuing examples 2.4.3, 2.4.29, 8.1.3, 8.1.11 and 8.1.21, any pullback of a generating left map $\mathbf{y}W \uplus \mathbf{y}W \rightarrow \mathbf{y}W$ (example 2.4.28) is again a surjective morphism (i.e. left map) $\Gamma \uplus \Gamma \rightarrow \Gamma$. As we are dealing with an OFS, the pullback square is automatically a morphism of left maps. Thus, this OFS is robust.

Example 8.4.6 (Codiscrete graphs: \blacktriangleright). Continuing examples 2.4.5, 2.4.31, 8.1.4, 8.1.12 and 8.1.22, any pullback of the generating left map $\mathbf{y}\mathbb{N} \uplus \mathbf{y}\mathbb{N} \rightarrow \mathbf{y}\mathbb{I}$ (example 2.4.28) is again bijective on nodes, i.e. a left map. Thus, this OFS is robust.

Example 8.4.7 (Discrete graphs: \blacktimes). Continuing examples 2.4.6, 2.4.32, 8.1.5, 8.1.13 and 8.1.23, the pullback of the sole generating left map $\mathbf{y}\mathbf{r} : \mathbf{y}\mathbb{I} \rightarrow \mathbf{y}\mathbb{N}$ along itself is $\pi_1 : \mathbf{y}\mathbb{I} \times \mathbf{y}\mathbb{I} \rightarrow \mathbf{y}\mathbb{I}$. This is not lifted by all discrete fibrations: it contains three edges in the domain (two parallel sides and a diagonal) which are not homogeneous (i.e. they are not mapped to reflexive edges in the codomain) and therefore need not be reflexive in a discrete type. Thus, this is not a robust way of generating an OFS. That does not mean that there is no other robust way, but we know that there is not because the discrete replacement monad for reflexive graphs is not stable under substitution.

Example 8.4.8 (Clock-irrelevance: \blacktriangleright). Continuing examples 2.4.34, 8.1.6 and 8.1.25, the pullback of a generating left map $\mathbf{y}W \times \odot \rightarrow \mathbf{y}W$ from a representable object $\mathbf{y}V$ is again a (generating) left map $\mathbf{y}V \times \odot \rightarrow \mathbf{y}V$. Thus, this OFS is robust by proposition 8.4.3.

Example 8.4.9 (0-discrete depth cubical sets: \blacktriangleright). Continuing examples 2.4.6, 2.4.32, 8.1.5, 8.1.13 and 8.1.23, the pullback of a generating left map $\mathbf{y}W \times \mathbf{y}(i : \langle 0 \rangle) \rightarrow \mathbf{y}W$ from a representable object $\mathbf{y}V$ is again a (generating) left map $\mathbf{y}V \times \mathbf{y}(i : \langle 0 \rangle) \rightarrow \mathbf{y}V$. Thus, this OFS is robust by proposition 8.4.3.

Example 8.4.10 (Kan fibrancy: \blackstar). Continuing examples 2.4.36, 8.1.9, 8.1.16 and 8.1.26, it is clear from example 8.1.17 that the way we generated the Kan NWFS was not robust.

Example 8.4.11 (Contextual Kan fibrancy: \blacktriangledown). Continuing examples 8.1.18 and 8.1.28, the pullback of the damped arrow

$$\mathbf{y}(W, i : \mathbb{I}).(\varphi[\mathbf{y}(i/\emptyset)] \vee (i \doteq 0)) \rightarrow \mathbf{y}(W, i : \mathbb{I}) \rightarrow \mathbf{y}W$$

along $\mathbf{y}\chi : \mathbf{y}V \rightarrow \mathbf{y}W$ is the damped arrow

$$\mathbf{y}(V, i : \mathbb{I}).(\varphi[\mathbf{y}(\chi, i/\emptyset)] \vee (i \doteq 0)) \rightarrow \mathbf{y}(V, i : \mathbb{I}) \rightarrow \mathbf{y}V$$

which is again a generating left map, so the damped Kan NWFS is robust by proposition 8.4.3.

Example 8.4.12 (Segal fibrancy: \blackstar). Continuing examples 2.4.35, 8.1.8, 8.1.15 and 8.1.27, the pullback of the generating left map $\Delta_2 \rightarrow \Delta_2$ along the inclusion of the composite side in Δ_2 is the endpoint inclusion in $\mathbf{y}[1]$. This is not a left map; indeed, if all Segal types would (uniquely⁵) lift this map, then they would have codiscrete edges. Thus, the way we generated the Segal OFS was not robust.

Example 8.4.13 (Contextual Segal fibrancy: \blacktriangledown). Inspired by proposition 8.4.3, we amend example 8.1.19: instead of explicitly adding *all* pullbacks of $\Lambda_n \rightarrow \Delta_n \rightarrow \Delta_1$ as generating left maps, we only add the pullbacks from representable objects. Mindful of remark 2.4.27, we also add all pullbacks of $\Delta_n \uplus \Lambda_n \Delta_n \rightarrow \Delta_n \rightarrow \Delta_1$ from representable objects. Still, the damped Segal OFS is robust.

8.4.3 Fibrancy of Π -types over a Robust NWFS

Theorem 8.4.14. [Nuy18a; Nuy18b] If $\Gamma.A \vdash B$ type is fibrant w.r.t. a robust NWFS, then $\Gamma \vdash \Pi AB$ is fibrant.

If $\Gamma.A.\Theta[\pi] \vdash B$ type is contextually fibrant in context $\Gamma.A$ w.r.t. a robust damped NWFS, then $\Gamma.\Theta \vdash \Pi(A[\pi])(B[\text{swap}])$ type is contextually fibrant in context Γ , where $\text{swap} : \Gamma.\Theta.A[\pi] \cong \Gamma.A.\Theta[\pi]$.

Proof. We only prove the damped statement, the other follows by simplifying the proof or via remark 8.2.12.

⁵The unique-making construction in remark 2.4.27 is preserved by pullback, at least in presheaf categories.

Pick a lifting problem $\vec{\varphi}$ against a generating left map \vec{g} :

$$\begin{array}{ccc}
 g_0 & \xrightarrow[\text{(\varphi}_1 g_\uparrow, \lambda(b[\text{swap}]))]{\varphi_0=} & \Gamma.\Theta.\Pi(A[\pi])(B[\text{swap}]) \\
 g_\uparrow \downarrow & \nearrow & \downarrow \pi \\
 g_1 & \xrightarrow{\varphi_1} & \Gamma.\Theta \\
 g_\downarrow \downarrow & & \downarrow \pi \\
 g_2 & \xrightarrow{\varphi_2} & \Gamma
 \end{array}$$

It can be written equivalently as:

$$\begin{array}{ccc}
 g_0.A[\varphi_2 g_\downarrow g_\uparrow] & \xrightarrow[\text{((\varphi}_1 g_\uparrow)_+, b)}{=} & \Gamma.A.\Theta[\pi].B \\
 g_\uparrow + \downarrow & \nearrow & \downarrow \pi \\
 g_1.A[\varphi_2 g_\downarrow] & \xrightarrow{\varphi_{1+}} & \Gamma.A.\Theta[\pi] \\
 g_\downarrow + \downarrow & & \downarrow \pi \\
 g_2.A[\varphi_2] & \xrightarrow{\varphi_{2+}} & \Gamma.A
 \end{array}$$

in the sense that there is a natural bijection between lifting solutions. But in the lower problem, we are lifting against a map that is a pullback of \vec{g} and therefore a left map, so the problem has a solution.

We still have to show naturality of the solution. Naturality w.r.t. B is straightforward. Naturality w.r.t. \vec{g} follows from the square condition in eq. (8.2) in the definition of robustness. \square

8.4.4 Stability of Robust NWFSSs

Theorem 8.4.15. [Nuy18b] Robust (damped) presheaf CwFs are stable.

Proof. We prove this in the damped case; the non-damped case follows by remark 8.2.12.

Pick a type $\Gamma.\Theta \vdash T \text{ type}$ and a substitution $\sigma : \Delta \rightarrow \Gamma$, and note that stability does not only completely characterize $\mathbf{R}(T[\sigma+])$ as $(\mathbf{R}T)[\sigma+]$; it also characterizes $F_{\mathbf{R}}(T[\sigma+])$ (which is an algebra over $\mathbf{R}(T[\sigma+])$) as the contextual

fibration which solves lifting problems as in the proof of proposition 8.3.1. Indeed, the carrier of $F_{\mathbf{R}}(T[\sigma+])$ is fixed, while the lifting structure needs to be respected by the pullback along σ since $F_{\mathbf{R}}$ is a functor.

Because $F_{\mathbf{R}} \dashv U_{\mathbf{R}}$, we know that $F_{\mathbf{R}} \vec{x}$ has the universal property that algebra morphisms $F_{\mathbf{R}} \vec{x} \rightarrow \hat{\nu}$ are in natural bijection with morphisms $\vec{x} \rightarrow \vec{r} = U_{\mathbf{R}} \hat{\nu}$. We will prove that $(\mathbf{R}T)[\sigma+]$ with the pulled-back structure (together denoted $(F_{\mathbf{R}}T)[\sigma+]$, i.e. we apply the substitution to the algebra) satisfies the universal property of $F_{\mathbf{R}}(T[\sigma+])$.

Since presheaf CwFs are locally democratic, we can assume without loss of generality that $\hat{\nu}$ is also a type. By the universal property of the pullback, without loss of generality we can pull back $\hat{\nu}$ to the same context Δ and telescope Θ , i.e. we can assume that we are dealing with a fibrant type $\Delta.\Theta[\sigma] \vdash S$ type with structure s , and we need to show that algebra morphisms $\Delta; \Theta[\sigma] \vdash (F_{\mathbf{R}}T)[\sigma+] \rightarrow (S, s)$ are in natural correspondence with functions $\Delta.\Theta[\sigma] \vdash (T[\sigma+]) \rightarrow S$.

Again by local democracy, we can assume without loss of generality that $\sigma = \pi : \Delta = \Gamma.A \rightarrow \Gamma$. Then we have

$$\begin{aligned} & (\Gamma.A; \Theta[\pi] \vdash (F_{\mathbf{R}}T)[\pi+] \rightarrow (S, s)) \\ & \cong (\Gamma; \Theta \vdash F_{\mathbf{R}}T \rightarrow (\Pi(A[\pi])(S[\text{swap}])), p)) \\ & \cong (\Gamma.\Theta \vdash T \rightarrow \Pi(A[\pi])(S[\text{swap}]))) \\ & \cong (\Gamma.A.\Theta[\pi] \vdash T[\pi+] \rightarrow S). \end{aligned}$$

In the first step, we use abstraction/application and the definition of the standard fibrancy structure on Π -types (theorem 8.4.14). In the second step, we use $F_{\mathbf{R}} \dashv U_{\mathbf{R}}$ (recall that $U_{\mathbf{R}}$ forgets the structure). The third step is again abstraction/application. \square

Lemma 8.4.16. Assume on a presheaf CwF a type operator $E : \text{Ty}(\Gamma) \rightarrow \text{Ty}(\Gamma)$ that preserves substitution up to isomorphism (i.e. it is a pseudonatural transformation $\text{Ty} \rightarrow \text{Ty}$). Then there is a naturally isomorphic operator F that preserves substitution on the nose (i.e. it is a natural transformation $\text{Ty} \rightarrow \text{Ty}$).

Proof. Take $\Gamma \vdash T$ type. We need to define $\Gamma \vdash FT$ type. Pick $\gamma : W \Rightarrow \Gamma$. Then we need to define $(W \triangleright (FT)[\gamma])$ which by theorem 4.1.4 is isomorphic to $(\mathbf{y}W \vdash (FT)[\gamma]) = (\mathbf{y}W \vdash F(T[\gamma])) \cong (\mathbf{y}W \vdash E(T[\gamma]))$. So we *must* (up to isomorphism) define

$$(W \triangleright (FT)[\gamma]) := (\mathbf{y}W \vdash E(T[\gamma])).$$

This respects substitution:

$$(W \triangleright (F(T[\sigma]))[\gamma]) = (\mathbf{y}W \vdash E(T[\sigma\gamma])) = (W \triangleright (FT)[\sigma][\gamma]),$$

and is clearly isomorphic to ET . \square

Corollary 8.4.17. Robust (damped) CwFs are strictly stable. \square

8.5 Internalizing Stable NWFSSs

8.5.1 Extensional Typing Rules for the Fibrant Replacement

Figures 8.1 and 8.2 list typing rules that internalize the fibrant replacement of a strictly stable (damped) NWFSS. We do this only in an extensional setting. The reason is that we want to remain agnostic as to which equalities hold judgementally and which ones only propositionally. Indeed, in specific cases, we might be able to implement the fibrant replacement operator and then the computational behaviour may depend on the implementation.

Basically, we just internalize the fact that \mathbf{R} is a monad on the category of (in the non-damped case) types T or (in the damped case) types T depending on another type D (which replaces the telescope Θ used in the previous sections). Note that while a monad is just a functor equipped with well-behaved natural transformations for unit and multiplication, the internalization in figs. 8.1 and 8.2 is completely different to what we saw for functors and natural transformations in figs. 5.1 and 5.2. This is because those figures were about internalizing functors between CwFs, whereas here we are internalizing functors between categories of (dependent) types within a given CwF.

Theorem 8.5.1 (Soundness). The typing rules in fig. 8.1 (fig. 8.2) are sound in a CwF equipped with a strictly stable (damped) NWFSS.

Proof. We only prove the damped result.

The rule \mathbf{RR} is derivable from

$$\frac{\Gamma, w : D \vdash T}{\Gamma, w : D \vdash \mathbf{R}(w.T)w}$$

which is modelled by the operator assumed in definition 8.3.2.

The premises of the functoriality rule create a morphism of damped arrows, to which we can apply the functorial action of the aforementioned operator.

Prerequisites: DTT (fig. 3.2)

Formation:

$$\frac{\text{RR} \quad \Gamma \vdash T \text{ type}}{\Gamma \vdash \mathbf{R} T \text{ type}}$$

where $(\mathbf{R} T)[\sigma] = \mathbf{R}(T[\sigma])$

Functoriality:

$$\frac{\text{RR:FMAP} \quad \Gamma \vdash S, T \text{ type} \quad \Gamma \vdash f : S \rightarrow T \quad \Gamma \vdash s' : \mathbf{R} S}{\Gamma \vdash \mathbf{R} f s' : \mathbf{R} T}$$

where $s' = \mathbf{R} \text{id } s'$ (RR:FMAP:ID)
 $\mathbf{R} f_2 (\mathbf{R} f_1 r') = \mathbf{R} (f_2 \circ f_1) r'$ (RR:FMAP:COMP)

Monad structure:

$$\frac{\text{RR:UNIT} \quad \Gamma \vdash t : T}{\Gamma \vdash \eta^{\mathbf{R}} t : \mathbf{R} T}$$

where $\mathbf{R} f (\eta^{\mathbf{R}} s) = \eta^{\mathbf{R}} (f s)$ (RR:UNIT:NAT)

$$\frac{\text{RR:MULT} \quad \Gamma \vdash T \text{ type} \quad \Gamma \vdash t'' : \mathbf{R}^2 T}{\Gamma \vdash \mu^{\mathbf{R}} t'' : \mathbf{R} T}$$

where $\mathbf{R} f (\mu^{\mathbf{R}} s'') = \mu^{\mathbf{R}} ((\mathbf{R}^2 f) s'')$ (RR:MULT:NAT)
 $\mu^{\mathbf{R}} (\eta^{\mathbf{R}} t') = t'$ (RR:LUNIT)
 $\mu^{\mathbf{R}} (\mathbf{R} \eta^{\mathbf{R}} t') = t'$ (RR:RUNIT)
 $\mu^{\mathbf{R}} (\mu^{\mathbf{R}} t''') = \mu^{\mathbf{R}} (\mathbf{R} \mu^{\mathbf{R}} t''')$ (RR:ASSOC)

In case of an OFS: Postulate an inverse to $\mu^{\mathbf{R}}$.

Figure 8.1: Extensional typing rules for the fibrant replacement of a strictly stable NWFS. Most substitution rules are omitted.

Prerequisites: DTT (fig. 3.2)

Formation:

$$\frac{\text{DRR} \quad \Gamma \vdash D \text{ type} \quad \Gamma \vdash d : D \quad \Gamma, w : D \vdash T \text{ type}}{\Gamma \vdash \mathbf{R}(w.T) d \text{ type}} \\ \text{where } (\mathbf{R}(w.T) d)[\sigma] = \mathbf{R}(w.T[\sigma+_w]) (d[\sigma])$$

Functoriality:

$$\frac{\text{DRR:FMAP} \quad \Gamma \vdash C \text{ type} \quad \Gamma, v : C \vdash S \text{ type} \quad \Gamma \vdash D \text{ type} \quad \Gamma, w : D \vdash T \text{ type} \quad \Gamma \vdash e : C \rightarrow D \quad \Gamma \vdash f : (v : C) \rightarrow S \rightarrow T[ev/w] \quad \Gamma \vdash c : C \quad \Gamma \vdash s' : \mathbf{R}(v.S) c}{\Gamma \vdash \mathbf{R} e f c s' : \mathbf{R}(w.T) (e c)} \\ \text{where } s' = \mathbf{R} \text{ id } (\lambda v. \text{id}) c s' \quad (\text{DRR:FMAP:ID}) \\ \mathbf{R} e_2 f_2 (e_1 b) (\mathbf{R} e_1 f_1 b r') = \mathbf{R} (e_2 \circ e_1) (\lambda u. f_2 (e_1 u) \circ f_1 u) b r' \quad (\text{DRR:FMAP:COMP})$$

Monad structure:

$$\frac{\text{DRR:UNIT} \quad \Gamma, w : D \vdash T \text{ type} \quad \Gamma \vdash d : D \quad \Gamma \vdash t : T[d/w]}{\Gamma \vdash \eta^{\mathbf{R}} d t : \mathbf{R}(w.T) d} \\ \text{where } \mathbf{R} e f c (\eta^{\mathbf{R}} c s) = \eta^{\mathbf{R}} (e c) (f s) \quad (\text{DRR:UNIT:NAT})$$

$$\frac{\text{DRR:MULT} \quad \Gamma, w : D \vdash T \text{ type} \quad \Gamma \vdash d : D \quad \Gamma \vdash t'' : \mathbf{R}(v.\mathbf{R}(w.T) v) d}{\Gamma \vdash \mu^{\mathbf{R}} d t'' : \mathbf{R}(w.T) d} \\ \text{where } \mathbf{R} e f c (\mu^{\mathbf{R}} c s'') = \mu^{\mathbf{R}} (e c) (\mathbf{R} e (\mathbf{R} e f) c s'') \quad (\text{DRR:MULT:NAT}) \\ \mu^{\mathbf{R}} d (\eta^{\mathbf{R}} d t') = t' \quad (\text{DRR:LUNIT}) \\ \mu^{\mathbf{R}} d (\mathbf{R} \text{id } \eta^{\mathbf{R}} d t') = t' \quad (\text{DRR:RUNIT}) \\ \mu^{\mathbf{R}} d (\mu^{\mathbf{R}} d t''') = \mu^{\mathbf{R}} d (\mathbf{R} \text{id } \mu^{\mathbf{R}} d t''') \quad (\text{DRR:ASSOC})$$

In case of a damped OFS: Postulate an inverse to $\mu^{\mathbf{R}} d$.

Figure 8.2: Extensional typing rules for the fibrant replacement of a strictly stable damped NWFSS. Most substitution rules are omitted. Removal of C , D and e yields fig. 8.1.

This is, of course, known to be a monad by definition of a damped NWFS.

If the damped NWFS is a damped OFS (i.e. the monad and comonad are idempotent) then an inverse to $\mu^{\mathbf{R}} d$ is also sound. \square

8.5.2 Extensional Typing Rules for the Left Coreplacement

The previous section has a soundness result, but no completeness result. The reason is obvious: a (damped) NWFS consists of not only a fibrant (right) replacement monad \mathbf{R} , but also a left coreplacement monad \mathbf{L} . We internalize that one in the current section.

Recall that in a strictly stable damped NWFS, we have

$$\mathbf{R}(\Gamma.\Theta.T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma) \cong (\Gamma.\Theta.\mathbf{R}_\Theta T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma).$$

(In the non-damped case, remove Θ .) Hence, we have

$$\mathbf{Fac}(\Gamma.\Theta.T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma) \cong \Gamma.\Theta.\mathbf{R}_\Theta T,$$

and therefore

$$\mathbf{L}(\Gamma.\Theta.T \xrightarrow{\pi} \Gamma.\Theta \xrightarrow{\pi} \Gamma) \cong (\Gamma.\Theta.T \xrightarrow{\text{id}.\eta^{\mathbf{R}}} \Gamma.\Theta.\mathbf{R}_\Theta T \xrightarrow{\pi \circ \pi} \Gamma). \quad (8.3)$$

All the constituents of the right hand of the above isomorphism have already been internalized functorially, so \mathbf{L} as a functor is already there. Moreover, a natural co-unit $\bar{\varepsilon}^{\mathbf{L}} : \mathbf{L}\vec{x} \rightarrow \vec{x}$ is constructed simply from the weakening $\pi : \Gamma.\Theta.\mathbf{R}_\Theta T \rightarrow \Gamma.\Theta$ which is the internalization of $\mathbf{r}_{\vec{x}} : \mathbf{Fac}\vec{x} \rightarrow x_1$. So what remains to be done is to internalize the comonadic duplication $\bar{\delta}^{\mathbf{L}} : \mathbf{L} \rightarrow \mathbf{L}^2$, its naturality, and the comonad laws.

Instead, we will internalize the characterization of comonads using co-unit and **extend** (proposition 2.2.47). Then we need to internalize **extend**, its naturality, and the comonad laws. This is done in figs. 8.3 and 8.4. The idea of the \mathbf{R}' -operation (LC:EXTEND, DLC:EXTEND) is that it creates a lifting $\mathbf{extend}(\bar{\varphi}) : \mathbf{L}\vec{x} \rightarrow \mathbf{L}\vec{y}$ of a diagram $\bar{\varphi} : \mathbf{L}\vec{x} \rightarrow \vec{y}$. By pullback, without loss of generality, we can assume that the lowermost arrow of this diagram is the identity. Then

Prerequisites: DTT (fig. 3.2), \mathbf{R} (fig. 8.1)

LC:EXTEND

$\Gamma \vdash S$ type $\Gamma, x' : \mathbf{R} S \vdash T$ type

$\Gamma \vdash f : (x : S) \rightarrow T[\eta^{\mathbf{R}}(x)/x'] \quad \Gamma \vdash s' : \mathbf{R} S$

$\Gamma \vdash \mathbf{R}' f s' : \mathbf{R} T[s'/x']$

where $\mathbf{R}' f_2 (\mathbf{R} f_1) r' = \mathbf{R}' (f_2 \circ f_1) r'$ (LC:EXTEND:NAT:DOM)

$\mathbf{R}' (f_2 r') (\mathbf{R}' f_1 r') = \mathbf{R}' (\lambda x. f_2 x (f_1 x)) r'$ (LC:EXTEND:NAT:COD)

$\mathbf{R}' \text{id } s' = s'$ (LC:EXTEND:RCOUNT)

$\mathbf{R}' (f_2 r') (\mathbf{R}' f_1 r') = \mathbf{R}' (\lambda x. f_2 x (f_1 x)) r'$ (LC:EXTEND:ASSOC)

$\mathbf{R}' f (\eta^{\mathbf{R}} s) = \eta^{\mathbf{R}}(f s)$ (LC:EXTEND:UNIT)

$\mathbf{R}' f (\mu^{\mathbf{R}} s'')$
 $= \mu^{\mathbf{R}} (\mathbf{R}'_{(x''. \mathbf{R} T[\mu^{\mathbf{R}} x''/x'])} (\mathbf{R}' f) s'')$ (LC:EXTEND:MULT)

In case of an OFS:

LC:EXTEND:ETA

$\Gamma, x : S \vdash t : T$

$\Gamma, x' : \mathbf{R} S \vdash t' : \mathbf{R} T$

$\Gamma, x : S \vdash \eta^{\mathbf{R}} t = t'[\eta^{\mathbf{R}} x/x'] : \mathbf{R} T[\eta^{\mathbf{R}} x/x']$

$\Gamma \vdash s' : \mathbf{R} S$

$\Gamma \vdash t'[s'/x'] = \mathbf{R}' (\lambda x. t) s' : \mathbf{R} T[s'/x']$

Figure 8.3: Extensional typing rules for the left coreplacement of a strictly stable NWFS. Substitution rules are omitted.

this is what they look like (non-damped on the left, damped on the right):

$$\begin{array}{ccc}
 \Gamma.S & \xrightarrow{\text{id.}(\eta^{\mathbf{R}},f)} & \Gamma.\mathbf{R}S.T \\
 \text{id.}\eta^{\mathbf{R}} \downarrow & & \downarrow \text{id.}\eta^{\mathbf{R}} \\
 \Gamma.\mathbf{R}S & \xrightarrow{\text{id.}\mathbf{R}'f} & \Gamma.\mathbf{R}S.\mathbf{R}T \\
 \pi \downarrow & \searrow \text{id} & \downarrow \pi \\
 \Gamma & & \Gamma.\mathbf{R}S
 \end{array}
 \qquad
 \begin{array}{ccc}
 \Gamma.C.S & \xrightarrow{\text{id.}(e\circ(\text{id.}\eta^{\mathbf{R}}),f)} & \Gamma.D.T \\
 \text{id.}\eta^{\mathbf{R}} \downarrow & & \downarrow \text{id.}\eta^{\mathbf{R}} \\
 \Gamma.C.\mathbf{R}_C.S & \xrightarrow{\text{id.}(e,\mathbf{R}'ef)} & \Gamma.D.\mathbf{R}_D.T \\
 \pi \downarrow & \searrow \text{id.}e & \downarrow \pi \\
 \Gamma.C & & \Gamma.D \\
 \pi \downarrow & & \downarrow \pi \\
 \Gamma & \xrightarrow{\text{id}} & \Gamma
 \end{array}
 \tag{8.4}$$

Prerequisites: DTT (fig. 3.2), damped \mathbf{R} (fig. 8.2)

DLC:EXTEND

$\Gamma \vdash C$ type

$\Gamma \vdash D$ type

$\Gamma \vdash e : (v : C) \rightarrow \mathbf{R}(v.S) v \rightarrow D$

$\Gamma \vdash f : (v : C) \rightarrow (x : S) \rightarrow T[ev(\eta^{\mathbf{R}}(x))/w]$

$\Gamma \vdash c : C$

$\Gamma, v : C \vdash S$ type

$\Gamma, w : D \vdash T$ type

$\Gamma \vdash s' : \mathbf{R}(v.S) c$

$\Gamma \vdash \mathbf{R}' e f c s' : \mathbf{R}(w.T) (e c s')$

where

$\mathbf{R}' e_2 f_2 (e_1 b) (\mathbf{R}' e_1 f_1 b r')$

$= \mathbf{R}' (\lambda u. (e_2 (e_1 u)) \circ (\mathbf{R}' e_1 f_1 u)) (\lambda u. (f_2 (e_1 u)) \circ (f_1 u)) b r'$

(DLC:EXTEND:NAT:DOM)

$\mathbf{R}' (e_2 r') (f_2 r') (e_1 b r') (\mathbf{R}' e_1 f_1 b r')$

$= \mathbf{R}' (\lambda u. \lambda x'. e_2 x' (e_1 u x')) (\lambda u. \lambda x. f_2 (\eta^{\mathbf{R}} u x) (e_1 u (\eta^{\mathbf{R}} u x)) (f_1 u x))$

(DLC:EXTEND:NAT:COD)

$s' = \mathbf{R}' (\lambda v. \lambda x'. v) (\lambda v. \text{id}) c s'$

(DLC:EXTEND:RCOUNT)

$\mathbf{R}' (e_2 r') (f_2 r') (e_1 b r') (\mathbf{R}' e_1 f_1 b r')$

$= \mathbf{R}' (\lambda u. \lambda x'. e_2 x' (e_1 u x')) (\mathbf{R}' e_1 f_1 u x')$

$(\lambda u. \lambda x. f_2 (\eta^{\mathbf{R}} u x) (e_1 u (\eta^{\mathbf{R}} u x)) (f_1 u x))$

(DLC:EXTEND:ASSOC)

$\mathbf{R}' e f c (\eta^{\mathbf{R}} c s) = \eta^{\mathbf{R}} (e c (\eta^{\mathbf{R}} c s)) (f c s)$

(DLC:EXTEND:UNIT)

$\mathbf{R}' e f c (\mu^{\mathbf{R}} c s'')$

$= \mu^{\mathbf{R}} (e c (\mu^{\mathbf{R}} c s'')) (\mathbf{R}' (\lambda v. \lambda x''. e v (\mu^{\mathbf{R}} v x'')) (\mathbf{R}' e f) c s'')$

(DLC:EXTEND:MULT)

In case of a damped OFS:

DLC:EXTEND:ETA

$\Gamma, v : C, x : S \vdash t : T[ev(\eta^{\mathbf{R}} v x)/w]$

$\Gamma, v : C, x' : \mathbf{R}(v.S) v \vdash t' : \mathbf{R}(w.T) (e v x')$

$\Gamma, v : C, x : S \vdash \eta^{\mathbf{R}} v x t = t' [ev(\eta^{\mathbf{R}} v x)/w] : \mathbf{R}(w.T) (e v (\eta^{\mathbf{R}} v x))$

$\Gamma \vdash c : C \quad \Gamma \vdash s' : \mathbf{R}(v.S) c$

$\Gamma \vdash t' [c/v, s'/x'] = \mathbf{R}' e (\lambda v. \lambda x. t) c s' : \mathbf{R}(w.T) (e c s')$

Figure 8.4: Extensional typing rules for the left coreplacement of a strictly stable damped NWFS. Substitution rules are omitted. Removal of C , setting $D = \mathbf{R} S$ and $e = \text{id}_{\mathbf{R} S}$ yields fig. 8.3.

The fact that the upper square commutes is not automatic and is internalized using `DLC:EXTEND:UNIT`. The left co-unit law (adding the co-unit on the right of the diagram) is however automatic, as the co-unit is just weakening.

We choose the notation \mathbf{R}' because internally it behaves like a dependent functoriality operator. In fact it generalizes the non-dependent operator `DRR:FMAP`, as is proven using `DLC:EXTEND:RCOUNT` and `DLC:EXTEND:NAT:DOM`:

$$\mathbf{R} e f c s' = (\mathbf{R}' (\lambda w. \lambda y'. w) (\lambda w. \text{id})) (e c) \circ \mathbf{R} e f c s' = \mathbf{R}' e f c s'.$$

Under this generalization, `DLC:EXTEND:NAT:DOM` and `DLC:EXTEND:NAT:COD` both generalize `DRR:FMAP:COMP` and show that \mathbf{R} preserves composition of one dependent map with a non-dependent map. The rule `DLC:EXTEND:UNIT` generalizes `DRR:UNIT:NAT`. The rule `DLC:EXTEND:ASSOC` is then the most general composition rule and expresses that if you have two diagrams as in eq. (8.4) juxtaposed, then it does not matter whether you take the lifting in each diagram and then compose, or first lift the left diagram, then compose, and then lift the entirety.

Theorem 8.5.2 (Soundness). The typing rules in fig. 8.3 (fig. 8.4) are sound in a CwF equipped with a strictly stable (damped) NWFS.⁶

Proof. We only prove the damped result.

The rule `DLC:EXTEND` is the lifting shown in eq. (8.4) and as such is modelled by the `extend` operation of the comonad \mathbf{L} . The rule `DLC:EXTEND:UNIT` expresses that the upper square in eq. (8.4) commutes, which is true. The rule `DLC:EXTEND:MULT` is proven in lemma 8.A.1. The other rules are tedious internalizations of the corresponding comonad laws.

Finally, if the model is an OFS, then postcomposition with $\varepsilon^{\mathbf{L}}$ is inverse to applying `extend`. The premises of `DLC:EXTEND:ETA` constitute a morphism of damped arrows such as those produced by `DLC:EXTEND`. Recall that $\text{Cod } \varepsilon^{\mathbf{L}} = \mathbf{r}$ is just weakening, i.e. forgetting t' . In other words, if we forget t' and then reconstruct a term of the same type using \mathbf{R}' , then we should get the same result, which is exactly what is expressed by `DLC:EXTEND:ETA`. \square

Theorem 8.5.3 (Completeness). A locally democratic⁷ CwF that models the typing rules in figs. 8.1 and 8.3 (figs. 8.2 and 8.4), omitting those for OFSs, is equipped with a strictly stable (damped) NWFS.

⁶In the case of `LC:EXTEND:UNIT`, `DLC:EXTEND:UNIT`, this is only true insofar as the proof of lemma 8.A.1 is correct.

⁷To be pedantic, we need the CwF to be *algebraically* locally democratic, i.e. there is an operator that elects a type $\Gamma \vdash T$ type for any slice $\sigma : \Delta \rightarrow \Gamma$, and moreover this operator should be the identity on $\pi : \Gamma.T' \rightarrow \Gamma$. This is unproblematic if we can use the axiom of choice.

Proof. We construct a (damped) Grandis-Tholen NWFS (definitions 2.4.20 and 8.2.8). By local democracy, we can assume (up to isomorphism) that any damped arrow \vec{x} is of the form $(\Gamma.T \xrightarrow{\pi} \Gamma)$ (or $(\Gamma.C.T \xrightarrow{\pi} \Gamma.C \xrightarrow{\pi} \Gamma)$ in the damped case).

The functor \mathbf{R} is constructed from the interpretation of RR (DRR). The other rules in fig. 8.1 (fig. 8.2) state that this is a monad, and this monad is trivial at the codomain (and damping) as required.

The functor \mathbf{L} is then constructed as in eq. (8.3). The co-unit is obtained by weakening.

The extend operation is not trivially obtained from LC:EXTEND (DLC:EXTEND) as a general morphism $\vec{\varphi} : \mathbf{L}\vec{x} \rightarrow \vec{z}$ may not satisfy $\varphi_1 = \text{id}$ ($\varphi_2 = \text{id}$) so that the context of T in that rule is insufficiently general.

However, any morphism $\vec{\varphi} : \mathbf{L}\vec{x} \rightarrow \vec{z}$ factors as $\vec{\varphi} = \vec{\psi} \circ \vec{\chi}$ where $\psi : \vec{y} \rightarrow \vec{z}$ is the pullback of \vec{z} along $\varphi_1 : \text{Fac}\vec{x} \rightarrow y_1$ ($\varphi_2 : x_2 \rightarrow y_2$) and $\chi_1 = \text{id}$ ($\chi_2 = \text{id}$). Now $\text{extend}(\vec{\chi})$ determines $\text{extend}(\vec{\varphi})$ by naturality and conversely $\text{extend}(\vec{\varphi})$ determines $\text{extend}(\vec{\chi})$ by the universal property of the pullback. Thus, without loss of generality, we do get to assume that $\varphi_1 = \text{id}$ ($\varphi_2 = \text{id}$), i.e. that the context of T is as in LC:EXTEND (DLC:EXTEND).

Then we use that rule to construct the extend operation, which produces morphisms of damped arrows by LC:EXTEND:UNIT (DLC:EXTEND:UNIT). Naturality, associativity and the right co-unit law are assumed in the rules (again making an assumption on the context of other morphisms involved, but this is again justified via the pullback trick). The left co-unit law follows from the structural rules of DTT.

Finally, the NWFS is strictly stable by construction. □

Strangely, we did not need DLC:EXTEND:MULT.

8.6 Internal Fibrancy via Eilenberg-Moore Algebras

We would like to define fibrancy internally. There are two ways to go about this: we can either define fibrant types to be Eilenberg-Moore algebras for the fibrant replacement (this section), or we can seek to generalize the approach by Licata et al. [Lic+18] and use the right adjoint to the function type, implemented via the transpension type (section 8.7).

8.6.1 Definition

Here, given $\Gamma, w : D \vdash T$ type, we define $\mathbf{Fib}(w.T)$ to be the type of Eilenberg-Moore algebra structures on $w.T$, i.e. it is a record type containing the following fields:

$$\begin{aligned} f &: (w : D) \rightarrow \mathbf{R}(w.T) w \rightarrow T, \\ _ &: (w : D) \rightarrow f w \circ \eta^{\mathbf{R}} w \equiv \text{id}, \\ _ &: (w : D) \rightarrow f w \circ \mu^{\mathbf{R}} w \equiv f w \circ \mathbf{R} \text{id} f w. \end{aligned}$$

We can thus define an (a priori) non-fibrant universe of fibrant types:

$$\mathbf{U}_\ell^{\text{NFF}}(D) := (T : D \rightarrow \mathbf{U}_\ell) \times \mathbf{Fib}(w.T w). \quad (8.5)$$

Of course, for non-contextual fibrancy w.r.t. a non-damped NWFS, we suppress D and we get:

$$\mathbf{U}_\ell^{\text{NFF}} := (T : \mathbf{U}_\ell) \times \mathbf{Fib}(T). \quad (8.6)$$

8.6.2 Fibrant Replacement Elimination

Theorem 8.6.1. The type $\mathbf{R}(w.A) w$ behaves like the inductive type family with constructors $\eta^{\mathbf{R}} : (w : D) \rightarrow A w \rightarrow \mathbf{R}(w.A) w$ and $(\mu^{\mathbf{R}}, _)$: $\mathbf{Fib}(w.\mathbf{R}(w.A) w)$ in the sense that the following typing rule holds:

DRR:ELIM

$$\begin{array}{l} \Gamma, w : D \vdash A \text{ type} \\ \Gamma, w : D, x' : \mathbf{R}(w.A) w \vdash T \text{ type} \\ \Gamma \vdash \tau : \mathbf{Fib}((w, x').T) \\ \Gamma, w : D, x : A \vdash t_\eta : T[\eta^{\mathbf{R}} w x / x'] \\ \Gamma \vdash d : D \\ \Gamma \vdash a' : \mathbf{R}(w.A) d \end{array}$$

$$\begin{array}{l} \Gamma \vdash t := \text{case } (d, a') \text{ of } \{(w, \eta^{\mathbf{R}} w x) \mapsto t_\eta \mid \tau\} : T[d/w, a'/x'] \\ \text{where } t[\eta^{\mathbf{R}} d a / a'] = t_\eta[d/w, a/x] \quad (\text{DRR:UNIT:BETA}) \\ \quad t[\mu^{\mathbf{R}} d a'' / a'] = \tau(d, \mu^{\mathbf{R}} d a'') \\ \quad (\mathbf{R}'(\lambda v. \lambda x''. (v, \mu^{\mathbf{R}} v x'')))(\lambda v. \lambda x'. t[v/d, x'/a']) d a \quad (\text{DRR:MULT:BETA}) \end{array}$$

In the non-damped case, this simplifies to:

$$\begin{array}{l}
\text{RR:ELIM} \\
\Gamma \vdash A \text{ type} \\
\Gamma, x' : \mathbf{R} A \vdash T \text{ type} \\
\Gamma \vdash \tau : \text{Fib}(T) \\
\Gamma, x : A \vdash t_\eta : T[\eta^{\mathbf{R}} x/x'] \\
\Gamma \vdash a' : \mathbf{R} A \\
\hline
\Gamma \vdash t := \text{case } a' \text{ of } \{ \eta^{\mathbf{R}} x \mapsto t_\eta \mid \tau \} : T[a'/x'] \\
\text{where } t[\eta^{\mathbf{R}} a/a'] = t_\eta[a/x] \quad (\text{RR:UNIT:BETA}) \\
t[\mu^{\mathbf{R}} a''/a'] = \tau \left(\mathbf{R}'_{x''.T[\mu^{\mathbf{R}} x''/x']} (\lambda x'. t[x'/a']) a'' \right) \quad (\text{RR:MULT:BETA})
\end{array}$$

Proof. We prove the damped statement. Thanks to the fibrancy structure τ , it is sufficient to construct $t' : \mathbf{R}((w, x').T)(d, a')$. This can be defined as

$$t' := \mathbf{R}'(\lambda w. \lambda x'. (w, x')) (\lambda w. \lambda x. t_\eta) d a'.$$

Then for DRR:UNIT:BETA indeed we have $t'[\eta^{\mathbf{R}} d a/a'] = \eta^{\mathbf{R}}(d, \eta^{\mathbf{R}} d a) t_\eta[d/w, a/x]$ which, after applying τ , yields $t[\eta^{\mathbf{R}} d a/a'] = t_\eta[d/w, a/x]$ by the Eilenberg-Moore laws.

For DRR:MULT:BETA we have

$$\begin{aligned}
& t'[\mu^{\mathbf{R}} d a''/a'] \\
&= \mu^{\mathbf{R}}(d, \mu^{\mathbf{R}} d a'') (\mathbf{R}'(\lambda v. \lambda x''. (v, \mu^{\mathbf{R}} v x'')) (\lambda v. \lambda x'. t'[v/d, x'/a']) d a'')
\end{aligned}$$

which, after applying τ , yields

$$\begin{aligned}
& (\lambda v. \tau v \circ \mathbf{R} \text{id } \tau v)(d, \mu^{\mathbf{R}} d a'') \\
& \quad (\mathbf{R}'(\lambda v. \lambda x''. (v, \mu^{\mathbf{R}} v x'')) (\lambda v. \lambda x'. t'[v/d, x'/a']) d a'') \\
&= \tau(d, \mu^{\mathbf{R}} d a'') (\mathbf{R}'(\lambda v. \lambda x''. (v, \mu^{\mathbf{R}} v x'')) (\lambda v. \lambda x'. t[v/d, x'/a']) d a).
\end{aligned}$$

This completes the proof \square

8.6.3 Fibrancy of Type Formers

In this section, we prove fibrancy of some type formers internally in the sense of section 8.6. We take the liberty to use a single notation for both fibrancy structures and their underlying operation.

Π -types

Proposition 8.6.2. We can prove:

$$\frac{\text{PI:FIB} \quad \Gamma \vdash A \text{ type} \quad \Gamma, x : A, w : D \vdash B \text{ type} \quad \Gamma, x : A \vdash \beta : \text{Fib}(w.B)}{\Gamma \vdash \psi : \text{Fib}(w.(x : A) \rightarrow B)}$$

The following proof is an internalization of the proof of theorem 8.4.14.

Proof. We define

$$\psi : (w : D) \rightarrow \mathbf{R}(w.(x : A) \rightarrow B) w \rightarrow (x : A) \rightarrow B$$

$$\psi w f' x = \beta w (\mathbf{R} \text{ id } (\lambda w. \text{ev}_x) w f'),$$

where $\text{ev}_x f := f x$. The algebra laws are satisfied:

$$\begin{aligned} & \psi w (\eta^{\mathbf{R}} w f) \\ &= \lambda x. \beta w (\mathbf{R} \text{ id } (\lambda w. \text{ev}_x) w (\eta^{\mathbf{R}} w f)) && \text{by def. } \psi, \\ &= \lambda x. \beta w (\eta^{\mathbf{R}} w ((\lambda w. \text{ev}_x) w f)) && \text{DRR:UNIT:NAT} \\ &= \lambda x. (\lambda w. \text{ev}_x) w f = f. && \text{algebra law.} \\ & \psi w (\mu^{\mathbf{R}} w f'') \\ &= \lambda x. \beta w (\mathbf{R} \text{ id } (\lambda w. \text{ev}_x) w (\mu^{\mathbf{R}} w f'')) && \text{by def. } \psi, \\ &= \lambda x. \beta w (\mu^{\mathbf{R}} w ((\mathbf{R} \text{ id})^2 (\lambda w. \text{ev}_x) w f'')) && \text{DRR:MULT:NAT} \\ &= \lambda x. \beta w (\mathbf{R} \text{ id } \beta w ((\mathbf{R} \text{ id})^2 (\lambda w. \text{ev}_x) w f'')) && \text{algebra law.} \\ &= \lambda x. \beta w (\mathbf{R} \text{ id } (\lambda w. \lambda f'. \beta w (\mathbf{R} \text{ id } (\lambda w. \text{ev}_x) w f')) w f'') && \text{DRR:FMAP:COMP} \\ &= \lambda x. \beta w (\mathbf{R} \text{ id } (\lambda w. \lambda f'. \psi w f' x) w f'') && \text{by def. } \psi, \\ &= \lambda x. \beta w (\mathbf{R} \text{ id } (\lambda w. \text{ev}_x \circ \psi w) w f'') && \text{by def. ev,} \\ &= \lambda x. \beta w (\mathbf{R} \text{ id } (\lambda w. \text{ev}_x) w (\mathbf{R} \text{ id } \psi w f'')) && \text{DRR:FMAP:COMP} \\ &= \psi w (\mathbf{R} \text{ id } \psi w f''). && \text{by def. } \psi. \end{aligned}$$

This ends the proof. \square

Σ -types

Proposition 8.6.3. We can prove:

$$\frac{\text{SIGMA:FIB} \quad \Gamma, w : D \vdash A \text{ type} \quad \Gamma, w : D, x : A \vdash B \text{ type} \quad \Gamma \vdash \alpha : \text{Fib}(w.A) \quad \Gamma \vdash \beta : \text{Fib}((w, x).B)}{\Gamma \vdash \sigma : \text{Fib}(w.(x : A) \times B)}$$

Proof. We define

$$\begin{aligned} \sigma &: (w : D) \rightarrow \mathbf{R}(w.(x : A) \times B) w \rightarrow (x : A) \times B \\ \text{fst}(\sigma w z') &= \alpha w (\mathbf{R} \text{id} (\lambda w. \text{fst}) w z') : A \\ \text{snd}(\sigma w z') &= \beta (w, \text{fst}(\sigma w z')) \\ &(\mathbf{R}' (\lambda w. \lambda z'. (w, \text{fst}(\sigma w z')))) (\lambda w. \text{snd}) z' \\ &: B[\text{fst}(\sigma w z')/x] \end{aligned}$$

One can verify that this is an algebra structure. \square

Identity types

Proposition 8.6.4. Using the rules internalizing an OFS, we can prove:

$$\frac{\text{ID:FIB} \quad \Gamma, w : D \vdash A \text{ type} \quad \Gamma \vdash \alpha : \text{Fib}(w.A) \quad \Gamma, w : D \vdash a, b : A}{\Gamma \vdash \delta : \text{Fib}(w.a \equiv_A b)}$$

Proof. We need to create a function $\delta : (w : D) \rightarrow \mathbf{R}(w.a \equiv_A b) w \rightarrow a \equiv_A b$. In other words, we need to prove

$$\lambda w. \lambda i'. a = \lambda w. \lambda i'. b : (w : D) \rightarrow \mathbf{R}(w.a \equiv_A b) w \rightarrow A.$$

First of all, from the algebra laws, we know that $a = \alpha w (\eta^{\mathbf{R}} w a)$ and similar for b . So it suffices to prove

$$\lambda w. \lambda i'. \eta^{\mathbf{R}} w a = \lambda w. \lambda i'. \eta^{\mathbf{R}} w b : (w : D) \rightarrow \mathbf{R}(w.a \equiv_A b) w \rightarrow \mathbf{R}(w.A) w.$$

Now, since we know that \mathbf{R}' generalizes \mathbf{R} , we get from DLC:EXTEND:ETA that we can replace both hands with the judgementally equal terms

$$\mathbf{R} \text{id} (\lambda w. \lambda i. a) = \mathbf{R} \text{id} (\lambda w. \lambda i. b) : (w : D) \rightarrow \mathbf{R}(w.a \equiv_A b) w \rightarrow \mathbf{R}(w.A) w.$$

Now this equation would follow from

$$\lambda w. \lambda i. a = \lambda w. \lambda i. b : (w : D) \rightarrow (a \equiv_A b) \rightarrow A,$$

and that is just true by the reflection rule.

We do not need to prove algebra laws as we can rely on ID:UIP □

Strictness type

Proposition 8.6.5. We can prove

$$\begin{array}{l} \text{STRICT:FIB} \\ \Gamma, w : D \vdash A \text{ type} \quad \Gamma \vdash \alpha : \text{Fib}(w.A) \\ \Gamma, \varphi \vdash T \text{ type} \\ \Gamma, \varphi \vdash i : T \cong A \\ \hline \Gamma \vdash \sigma : \text{Fib}(w.\text{Strict}\{A \cong (\varphi ? T, i)\}) \end{array}$$

Proof. Fibrancy is obviously preserved by isomorphism. □

Glue type

Proposition 8.6.6. We can prove

$$\begin{array}{l} \text{GLUE:FIB} \\ \Gamma, w : D \vdash A \text{ type} \quad \Gamma \vdash \alpha : \text{Fib}(w.A) \\ \Gamma, w : D, \varphi \vdash T \text{ type} \quad \Gamma \vdash \tau : \text{Fib}((w, p : [\varphi]).T) \text{ type} \\ \Gamma, w : D, \varphi \vdash f : T \rightarrow A \quad \Gamma, w : D, \varphi \vdash f \circ \tau (w, \text{tt}) \\ \quad = \alpha w \circ \mathbf{R} \text{id} (\lambda w. f) w : \mathbf{R}((w, p).T) (w, \text{tt}) \\ \hline \Gamma \vdash \gamma : \text{Fib}(\text{Glue}\{A \leftarrow (\varphi ? T, f)\}) \end{array}$$

Proof. We define

$$\begin{aligned} \gamma : (w : D) \rightarrow \mathbf{R}(w.\text{Glue}\{A \leftarrow (\varphi ? T, f)\}) w \rightarrow \text{Glue}\{A \leftarrow (\varphi ? T, f)\} \\ \gamma w g' = \text{glue}\{\alpha w (\mathbf{R} \text{id} (\lambda w. \text{unglue}\{\varphi ? f\}) g') \leftarrow (\varphi ? \tau (w, \text{tt}) g')\}. \end{aligned}$$

One can verify that this is well-defined and an algebra structure. □

Weld type and coproduct

Being a right map is not preserved by colimits. Consider the example of codiscrete reflexive graphs. The `Unit` type is codiscrete, but `Unit` \oplus `Unit` is not, and neither is

$$i : \mathbb{I} \vdash \text{Weld}\{[i \doteq_{\mathbb{I}} 0] \rightarrow (i \doteq_{\mathbb{I}} 1 ? \text{Unit}, _)\}.$$

(where \mathbb{I} is modelled as \mathbf{yI}) despite $[i \doteq_{\mathbb{I}} 0]$ and `Unit` being codiscrete and $_$ (vacuously) preserving the codiscrete structure. Indeed, the `Weld`-type contains an element `weld` $\{\perp ? _ \}$ `tt` if $i = 0$ and `unit` if $i = 1$, but there is no edge connecting these.

Universe

Few would probably expect the universe of all types to be fibrant for arbitrary notions of fibrancy. It is more interesting to look at the (a priori) non-fibrant universe of fibrant types \mathbf{U}^{NFF} . Consider again the example of codiscrete reflexive graphs. There, the nodes of \mathbf{U}^{NFF} are codiscrete reflexive graphs, whereas an edge from S to T is a proof-relevant relation between the nodes of S and T and codiscreteness requires that this proof-relevant relation is always uniquely true. This means that the relation itself is unique *up to isomorphism*, which simply is not good enough. We refer to Xu and Escardó [XE16] for further discussion.

8.7 Internal Fibrancy, Directly

Much of the discussion in the previous sections was related to internalizing the fibrant *replacement* operation. However, even in systems where this operation may be unstable under substitution and therefore not even axiomatizable as a type operator, proposition 8.3.1 tells us that we can still try to internalize the notion of being fibrant. An opaque way to do so is by having a judgement $\Gamma \vdash T \text{ fib}$ or, for contextual fibrancy, $\Gamma; \Theta \vdash T \text{ fib}$, or perhaps by axiomatizing a type `Fib`(T).

However, in some systems, this type `Fib`(T) can actually be defined:

Example 8.7.1 (Clock-irrelevance). Continuing examples 2.4.34, 8.1.6 and 8.1.25, a type $\Gamma \vdash T \text{ type}$ is clock-irrelevant if all functions $\odot \rightarrow T$ are constant. This is definable internally.

Example 8.7.2 (0-discrete depth cubical sets). Continuing examples 2.4.6, 2.4.32, 8.1.5, 8.1.13 and 8.1.23, a type $\Gamma \vdash T \text{ type}$ is 0-discrete if all functions

$\mathbf{y}(0) \rightarrow T$ are constant. This is definable internally:

$$\text{Fib}(T) := (f : \mathbf{y}(0) \rightarrow T) \rightarrow (f \equiv_{\mathbf{y}(0) \rightarrow T} \lambda_{_}.f 0).$$

For other systems, this is not so easy, since we need to be able to quantify functions to the context:

Example 8.7.3 (Kan fibrancy). Continuing examples 2.4.36, 8.1.9, 8.1.16 and 8.1.26, a type $\Gamma \vdash T$ type is Kan if, for every substitution from the interval $\gamma : (i : \mathbb{I}) \rightarrow \Gamma$, there is a Kan composition structure over $(i : \mathbb{I}) \vdash T[\gamma]$. Because the quantification over γ cannot be expressed internally in context Γ , Orton and Pitts [OP18; Ort18] go for a *meta-internal* approach: they define Kan fibrancy internal to some type system, but not internal to cubical HoTT itself. Indeed, in between cubical HoTT and the type system they use (the internal language of a topos) is a translation that promotes the context Γ to a type.

However, recall that the context $(\Gamma, \mathbf{a}_{\check{Q}(i:\mathbb{I})})$, denoted $\Pi(i : \mathbb{I}).\Gamma$ in the internal language of section 7.5 (note that $i : \mathbb{I}$ is interpreted by a cartesian multiplier), is semantically a Π -type over Γ . Instead, we need a non-dependent function type, which is given by $(\Gamma, \mathbf{a}_{\Pi i}, \mathbf{a}_{\check{Q}(i:\mathbb{I})})$, written $\Pi(i : \mathbb{I}).\Omega i.\Gamma$ in the informal notation. Thus, we can define (in formal MTT and informal notation resp.):

$$\Gamma \vdash \text{Fib}(T) := \left\langle \Pi(i : \mathbb{I}) \mid^{\mathfrak{p}} \left\langle \check{Q} i \mid^{\mathfrak{t}} \text{KanComp} \cdot \overset{\mathfrak{p}' \circ}{\Pi(i:\mathbb{I}) \circ \Omega i} T[\text{const}_i \downarrow_{\mathfrak{t} \circ}^{\bullet}][\downarrow_{\mathfrak{p}}, \text{reid} \times_i \downarrow_{\mathfrak{t} \mathfrak{p}'}^{\bullet}, \downarrow_{\mathfrak{c}}^{\bullet}] \right\rangle \right\rangle$$

$$\Gamma \vdash \text{Fib}(T) := \Pi(i : \mathbb{I}).\check{Q} i.\text{KanComp}(j.T[\gamma|_{i=j}/\gamma])$$

Then we can define \mathbf{U}^{NFF} as in eq. (8.6). Note that this is not a violation of Licata et al.'s no-go theorem [Lic+18]: for them, Γ is a type which lives in a context, and their no-go theorem tries to define a fibrant universe with type encoding and decoding operations which are stable w.r.t. the context of Γ . Here, Γ is itself a context, hence closed, so we are inherently working globally.

Example 8.7.4 (Segal fibrancy). Continuing examples 2.4.35, 8.1.8, 8.1.15 and 8.1.27, we would like to define Segal fibrancy by similar reasoning. However, it is not immediately clear what multiplier we should transpend over. Perhaps Pinyo and Kraus's twisted prism functor (example 7.4.11) [PK19] can come to the rescue.

Example 8.7.5 (Contextual Kan fibrancy). Continuing examples 8.1.18 and 8.1.28, to contextual Kan fibrancy of $\Gamma; \Theta \vdash T$ type internally, we have two options.

We can either require that Θ consist of a single type D , so that we can quantify over functions $\mathbb{I} \rightarrow D$ internally and simply define

$$\text{Fib}((w : D).A) = (\delta : \mathbb{I} \rightarrow D) \rightarrow \text{KanComp}(i.A[\delta i/w]).$$

Again, we can then define $\mathbf{U}^{\text{NFF}}(D)$ as in eq. (8.5).

Alternatively, we can leave Θ in the context and look for a modality whose left adjoint sends $\Gamma; \Theta$ to $\Gamma; \mathbb{I} \rightarrow \Theta$. This is the intention of the examples on embargoes in the technical report of chapter 7 [Nuy20]. There, we turn the semicolon (;) into a variable of the embargo shape (with Γ fresh for that variable) and the multiplier for \mathbb{I} is lifted to a multiplier that only affects the part Θ under the embargo.⁸

Appendices

8.A Semantics of Extend after Multiplication

Lemma 8.A.1. The typing rule LC:EXTEND:MULT in fig. 8.3 (DLC:EXTEND:MULT in fig. 8.4) is sound in a CwF equipped with a strictly stable (damped) NWFS.⁹

Proof. We prove the damped statement.

Write $\vec{s} = (\Gamma.C.S \xrightarrow{\pi} \Gamma.C \xrightarrow{\pi} \Gamma)$ and $\vec{t} = (\Gamma.D.T \xrightarrow{\pi} \Gamma.D \xrightarrow{\pi} \Gamma)$. The input to DLC:EXTEND is a morphism $\vec{\varphi} = (\text{id}.(e \circ (\text{id}.\eta^{\mathbf{R}}), f), \text{id}.e, \text{id}) : \mathbf{L} \vec{s} \rightarrow \vec{t}$.

Consider the diagram in fig. 8.5.

The right¹⁰ surface of the cuboid has as vertical edges the factorizations of \vec{s} and \vec{t} and the three black arrows are the components of $\vec{\varphi}$. The middle arrow is slanted, because $\vec{\varphi}$ has \mathbf{L} applied to the domain but not the codomain.

The extend operation of the comonad \mathbf{L} produces the dashed arrow, which is $\text{extend}(\vec{\varphi})_1$. Together with the upper and lower black arrows, this forms $\text{extend}(\vec{\varphi}) : \mathbf{L} \vec{s} \rightarrow \mathbf{L} \vec{t}$

As we know, $\text{extend}(\vec{\varphi}) = \mathbf{L} \vec{\varphi} \circ \delta_{\vec{s}}^{\mathbf{L}} : \mathbf{L} \vec{s} \rightarrow \mathbf{L} \vec{t}$. Now $\delta_{\vec{s}}^{\mathbf{L}} : \mathbf{L} \vec{s} \rightarrow \mathbf{L}^2 \vec{s}$ has as middle arrow the dotted arrow $\delta_{\vec{s}}^{\mathbf{L}} : \Gamma.C.\mathbf{R}_C S \rightarrow \Phi$ where $\Phi = \text{Fac } \mathbf{L} \vec{s}$. As the comonad \mathbf{L} is trivial on the domain and the damping, we have $\delta_{\vec{s}}^{\mathbf{L}} = (\text{id}_{\Gamma.C.S}, \delta_{\vec{s}}^{\mathbf{L}}, \text{id}_{\Gamma})$. The dashed arrow $\Phi \rightarrow \Gamma.D.\mathbf{R}_D T$ is then $(\mathbf{L} \vec{\varphi})_1 = \text{Fac } \vec{\varphi}$. The dashed arrow $\Gamma.C.S \rightarrow \Phi$ is $\ell_{\mathbf{L} \vec{s}}$.

The left surface has as vertical edges the factorizations of $\mathbf{R} \vec{s}$ and $\mathbf{R} \vec{t}$. The three horizontal black arrows form a morphism $\vec{\psi} : \mathbf{L} \mathbf{R} \vec{s} \rightarrow \mathbf{R} \vec{t}$. We define it to

⁸In a preprint of the technical report, the word ‘signpost’ was used instead of ‘embargo’.

⁹This lemma and proof are provided “as is” and without warranty.

¹⁰After turning your page/head so you can read the diagram, that is.

be $\vec{\psi} = (\text{extend}(\vec{\varphi})_1, \varphi_1 \circ (\text{id}.\mu^{\mathbf{R}}), \text{id})$. The upper trapezium commutes because $\text{id}.\mu^{\mathbf{R}} \circ \text{id}.\eta^{\mathbf{R}} = \text{id}$. Again we obtain the dashed arrow $\text{extend}(\vec{\psi})_1$ which factors over $\Psi = \text{Fac } \mathbf{L}RR\vec{s}$.

The middle vertical surface is a hybrid from the two sides: the upper line is taken from the right, the lower trapezium is taken from the left. This constitutes a morphism $\theta : \vec{k} \rightarrow \vec{t}$ from some damped arrow

$$\vec{k} = (\Gamma.C.S \xrightarrow{\text{id}.\eta^{\mathbf{R}} \circ \eta^{\mathbf{R}}} \Gamma.C.\mathbf{R}_C^2.S \xrightarrow{\pi \circ \pi} \Gamma).$$

which is *not* in the image of \mathbf{L} . It is still a left map, however (since $\eta^{\mathbf{R}} = \ell$ is a left map and left maps are closed under composition as is clear from the characterization as maps lifting right maps) so it still has an algebra structure $\kappa : \ell \rightarrow \mathbf{L}\ell$ with $\kappa_1 : \Gamma.C.\mathbf{R}_C^2.S \rightarrow \Theta = \text{Fac } \vec{\ell}$.

We complete the **front and back surfaces** with obvious choices of identities, units and multiplications. These diagrams commute by the monad laws. The diagrams in the back are moreover morphisms of left arrows, as can be seen by the fact that they lift right arrows compatibly. On the left, this follows immediately from the fact that the middle arrow is a composite. On the right, it follows from being a composite in combination with a monad law.

Functoriality of Fac yields arrows $\Theta \rightarrow \Psi$, $\Theta \rightarrow \Phi$, and the fact that the diagrams in the back are morphisms of left arrows, means that the morphisms from Θ will respect the algebra structures.

Thus, everything commutes.

Then we see that

$$\text{extend}(\vec{\varphi})_1 \circ (\text{id}.\mu^{\mathbf{R}}) = \text{Fac } \vec{\theta} \circ \kappa_1,$$

$$(\text{id}.\mathbf{R}\eta^{\mathbf{R}}) \circ \text{Fac } \vec{\theta} \circ \kappa_1 = \text{extend}(\vec{\psi})_1.$$

Postcomposing the second line with $\text{id}.\mu^{\mathbf{R}}$, we get

$$\text{extend}(\vec{\varphi})_1 \circ (\text{id}.\mu^{\mathbf{R}}) = \text{Fac } \vec{\theta} \circ \kappa_1 = \text{id}.\mu^{\mathbf{R}} \circ \text{extend}(\vec{\psi})_1.$$

Translating to type theory, the left hand side becomes

$$\text{extend}(\vec{\varphi})_1 \circ (\text{id}.\mu^{\mathbf{R}}) \rightsquigarrow (c, s'') \mapsto \mathbf{R}' e f c (\mu^{\mathbf{R}} c s'').$$

Then $\text{extend}(\vec{\psi})_1$ becomes

$$\text{extend}(\vec{\psi})_1 \rightsquigarrow (c, s'') \mapsto \mathbf{R}' (\lambda u. \lambda x''. e u (\mu^{\mathbf{R}} u x'')) (\mathbf{R}' e f) c s''.$$

Then the second line becomes

$$\text{id.}\mu^{\mathbf{R}} \circ \text{extend}(\vec{\psi})_1 \rightsquigarrow (c, s'') \mapsto \\ \mu^{\mathbf{R}} (ec(\mu^{\mathbf{R}} cs'')) (\mathbf{R}' (\lambda u. \lambda x''. eu(\mu^{\mathbf{R}} ux''))) (\mathbf{R}' ef) cs'').$$

This proves the rule DLC:EXTEND:MULT.

The rule LC:EXTEND:MULT is proven analogously but should be simpler. \square

Chapter 9

Degrees of Relatedness

Preamble In this chapter, we discuss the work

[ND18a] A. Nuyts and D. Devriese. “Degrees of Relatedness: A Unified Framework for Parametricity, Irrelevance, Ad Hoc Polymorphism, Intersections, Unions and Algebra in Dependent Type Theory”. In: *Logic in Computer Science (LICS) 2018, Oxford, UK, July 09-12, 2018*. 2018, pp. 779–788. DOI: 10.1145/3209108.3209119. URL: <https://doi.org/10.1145/3209108.3209119>

which presents a type system that we will refer to as RelDTT for short. This work builds on prior work

[NVD17a] A. Nuyts, A. Vezzosi, and D. Devriese. “Parametric quantifiers for dependent type theory”. In: *PACMPL 1.ICFP (2017)*, 32:1–32:29. DOI: 10.1145/3110276. URL: <http://doi.acm.org/10.1145/3110276>

which presents the type system ParamDTT and which we consider to be largely superseded by RelDTT. ParamDTT was accompanied by a technical report [Nuy17] which was extended to become a technical report of RelDTT [Nuy18a].

Instead of including here the full papers themselves, I will include an application section, written by myself, from the technical report on MTT:

[Gra+20a] D. Gratzer, A. Kavvos, A. Nuyts, and L. Birkedal. “Type Theory à la Mode”. Pre-print. 2020. URL: <https://anuyts.github.io/files/mtt-techreport.pdf>

which discusses at a high level the motivation behind and the model of these two papers, and also puts them in a more ‘contemporary’ perspective by explaining how they are (almost) instances of MTT. The text is taken almost verbatim, only theorems 9.2.7 and 9.3.6 and their proofs have been slightly integrated with the content of chapters 6 and 8.

In an additional section 9.5, which is written on the occasion of this thesis, I discuss a number of parametricity features and what their requirements are in terms of complexity of the type system and its model. In particular, I want to waylay the misconception that I seem to have created over the past few years through imprudent communication, that dependently typed parametricity would always require modalities. This is not the case: a parametricity modality is only needed if we want identity extension for large types.

Personal contributions In ParamDTT [NVD17a], Andrea Vezzosi contributed important insights regarding the open problems in Atkey et al.’s work [AGJ14] and some cornerstone ideas to model an improved system, the idea to use `Glue` as a parametricity operator, the implementation of `agda-parametric`, and everything related to the application to sized types. Dominique Devriese’s contributions are limited to indispensable general supervision and writing support. I contributed in particular the precise type system, the results on Church encoding and the fully elaborated model [Nuy17]. In RelDTT [ND18a], again, Devriese’s contributions are in the area of general supervision and writing. Additional insights in this section are entirely my contribution.

Introduction Of all type systems present in the literature, the most similar to MTT is probably that of Degrees of Relatedness [ND18a]. In section 9.1, we discuss at a conceptual level how Reynolds’ original formulation of parametricity [Rey83] was gradually generalized to dependent types. In section 9.2, we explain how modalities can help to validate the identity extension lemma for large types [NVD17a]. In section 9.3, we discuss Degrees of Relatedness proper, and in section 9.4, we consider how MTT can serve as an internal language in which one could build a model of Degrees of Relatedness.

9.1 Parametricity, from System F to DTT

We discuss parametricity in System F [Rey83], System $F\omega$ [Atk12], and dependent type theory [AGJ14; BCM15; Mou16].

9.1.1 System F

System F is relationally parametric [Rey83]. If we think of proof-irrelevant relations $R : \text{Rel}(A, B)$ as notions of heterogeneous equality between elements of A and elements of B , and write $a \simeq_R b$ for $R(a, b)$ in order to emphasize this perspective, then we can conceptually describe proof-relevant relational parametricity as follows:

- Type-level operations $F : * \rightarrow *$ preserve (meta-theoretical) equality,¹
- Type-level operations $F : * \rightarrow *$ preserve relations,
 - sending the equality relation on X to the equality relation on FX (this is the identity extension lemma),
- Parametric functions $f : \forall X. FX$ send relations to proofs of heterogeneous equality,
- Term-level operations $hX : FX \rightarrow GX$ preserve heterogeneous equality.

The identity extension lemma asserts that our use of the name ‘heterogeneous equality’ is sensible: in the homogeneous case, it boils down to mathematical equality.

We can represent this diagrammatically as follows:

$$\begin{array}{ccc}
 A = B & \xrightarrow{F=} & FA = FB \\
 \text{Eq} \downarrow & & \text{Eq} \downarrow \\
 \text{Rel}(A, B) & \xrightarrow{F_{\text{Rel}}} & \text{Rel}(FA, FB)
 \end{array}$$

$$\begin{array}{ccc}
 A = B & & fA \simeq_{F_{\text{Rel}}R} fB \\
 \text{Eq} \downarrow & \nearrow f_{\text{Rel}} & \\
 R : \text{Rel}(A, B) & &
 \end{array}$$

$$a \simeq_{F_{\text{Rel}}R} b \xrightarrow{h_{\text{Rel}}R} hAa \simeq_{G_{\text{Rel}}R} hBb$$

These diagrams are a bit awkward in the sense that some of their nodes are meta-theoretic propositions whereas others are meta-theoretic sets. For example,

¹A sane meta-theory will not allow the creation of anything that doesn’t.

the arrow $\text{Eq} : A = B \rightarrow \text{Rel}(A, B)$ is to be read as: if A and B are really the same thing, then Eq will pick out a relation Eq_A between A and B , namely the identity relation. Set theorists who do not balk at dealing with large objects, may prefer to write this as $\text{Eq} \in \prod_A \text{Rel}(A, A)$. Commutativity of the diagram simply means that $F_{\text{Rel}}\text{Eq}_A = \text{Eq}_{FA}$.

The arrow $f_{\text{Rel}} : (R : \text{Rel}(A, B)) \rightarrow fA \simeq_{F_{\text{Rel}}R} fB$ means: for any relation $R : \text{Rel}(A, B)$, the relation $F_{\text{Rel}}R$ will relate fA and fB . This would be more typically written as $\forall (R \in \text{Rel}(A, B)).(fA, fB) \in F_{\text{Rel}}R$.

An alternative way to make sense of the diagram is by translating every proposition P to the subsingleton $\{*\mid P\}$.

9.1.2 System $F\omega$

Atkey [Atk12] extends Reynolds' ideas to System $F\omega$. Every kind κ is equipped with a 'native' proof-relevant relation $\frown_{\kappa} : \kappa \times \kappa \rightarrow \mathbf{Set}$, such that $\frown_* = \text{Rel}$.² We say that $K_1, K_2 : \kappa$ are **related** if we can give an element of $K_1 \frown_{\kappa} K_2$.³ Similarly, for every $K : \kappa$, we get a proof $\text{refl}(K) : K \frown_{\kappa} K$ such that for $X : *$ we get $\text{refl}(X) = \text{Eq}_X : \text{Rel}(X, X)$. We can then generalize our description of relational parametricity:

- Type-level operations $F : \theta \rightarrow \kappa$ preserve equality,
- Type-level operations $F : \theta \rightarrow \kappa$ preserve relatedness,
 - sending $\text{refl}(X)$ to $\text{refl}(FX)$ (this is the identity extension lemma),
- Parametric functions $f : \forall (X : \kappa).FX$ send related types to heterogeneously equal terms,
- Term-level operations $hX : FX \rightarrow GX$ preserve heterogeneous equality.

²We ignore size issues in this introductory exposition.

³Note that any two types $T_1, T_2 : *$ are related. However, as \frown_* is a proof-relevant relation, we care not only for the truth value (whether types are related) but also for the particular proof we choose to give (the relation $R : T_1 \frown_* T_2$ that we consider between T_1 and T_2).

Diagrammatically (the same interpretation remarks apply as for the System F diagrams above):

$$\begin{array}{ccc}
 A =_{\theta} B & \xrightarrow{F=} & FA =_{\kappa} FB \\
 \text{refl} \downarrow & & \text{refl} \downarrow \\
 A \frown_{\theta} B & \xrightarrow{F_{\frown}} & FA \frown_{\kappa} FB
 \end{array}$$

$$\begin{array}{ccc}
 A =_{\kappa} B & & fA \frown_{F_{\frown}R} fB \\
 \text{refl} \downarrow & \nearrow f_{\frown} & \\
 R : A \frown_{\kappa} B & &
 \end{array}$$

$$a \frown_{F_{\frown}R} b \xrightarrow{h_{\frown}R} hAa \frown_{G_{\frown}R} hBb$$

Following Robinson and Rosolini [RR94] and Hasegawa [Has94a; Has94b], Atkey structured all of this in a reflexive graph model. A reflexive graph Γ is a (contravariant) presheaf over the category RG generated by the following diagram, subject to the following equations:⁴

$$\begin{array}{ccc}
 \mathbb{N} & \begin{array}{c} \xrightarrow{s} \\ \xleftarrow{r} \\ \xrightarrow{t} \end{array} & \mathbb{I} \\
 & & \mathbf{r} \circ \mathbf{s} = 1_{\mathbb{N}}, \\
 & & \mathbf{r} \circ \mathbf{t} = 1_{\mathbb{N}}.
 \end{array}$$

The idea is that $\mathbb{N} \Rightarrow \Gamma$ is the set of nodes, $\mathbb{I} \Rightarrow \Gamma$ is the set of edges, and that $(\sqcup \circ \mathbf{s}), (\sqcup \circ \mathbf{t}) : (\mathbb{I} \Rightarrow \Gamma) \rightarrow (\mathbb{N} \Rightarrow \Gamma)$ extract the source and target of an edge, whereas $(\sqcup \circ \mathbf{r}) : (\mathbb{N} \Rightarrow \Gamma) \rightarrow (\mathbb{I} \Rightarrow \Gamma)$ produces the reflexive edge on a node. The equations assert that the edge $\gamma \circ \mathbf{r}$ really goes from γ to γ .

In this reflexive graph model, kinds κ are interpreted as large reflexive graphs $\llbracket \kappa \rrbracket$. The nodes in $\mathbb{N} \Rightarrow \llbracket \kappa \rrbracket$ are the semantic elements of κ , whereas the edges in $\mathbb{I} \Rightarrow \llbracket \kappa \rrbracket$ can be seen as a triple of two elements $K_1, K_2 : \kappa$ wrapped up with a proof of $K_1 \frown_{\kappa} K_2$. The kind $*$ specifically is interpreted as the reflexive graph $\llbracket * \rrbracket$ whose nodes are small sets and whose edges are proof-irrelevant relations, the reflexive edges being the equality relations. An open type $\Gamma \vdash K : \kappa$ is then a reflexive graph morphism (i.e. a presheaf morphism) $\llbracket K \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \kappa \rrbracket$. The

⁴Readers who expected the opposite of RG are likely thinking of *covariant* functors to **Set**, whereas we take presheaves to be contravariant functors to **Set**.

fact that these preserve reflexive edges (for $*$ this means the equality relation), expresses the identity extension lemma.

This means that a closed type $\cdot \vdash T : *$ is essentially a small discrete reflexive graph, i.e. a small reflexive graph whose only edges are the reflexive ones. To see this, note that the empty context is interpreted as the terminal reflexive graph $\llbracket \cdot \rrbracket$, having a single node $()$ and a single reflexive edge $() \circ \mathbf{r}$. This node $()$ is then mapped to a small set $\llbracket T \rrbracket \circ ()$, and the edge $() \circ \mathbf{r}$ to a relation $\llbracket T \rrbracket \circ ((\) \circ \mathbf{r})$ on that set. However, since graph morphisms map reflexive edges to reflexive edges, and reflexive edges in $\llbracket * \rrbracket$ are the equality relations, we see that

$$\llbracket T \rrbracket \circ ((\) \circ \mathbf{r}) = (\llbracket T \rrbracket \circ ()) \circ \mathbf{r} = \text{Eq}_{\llbracket T \rrbracket \circ ()}$$

i.e. $\llbracket T \rrbracket$ is a set equipped with its equality relation.

A general (open) type $\Gamma \vdash T : *$ can be reorganized to be seen as a reflexive graph $\llbracket \Gamma | T \rrbracket \rightarrow \llbracket \Gamma \rrbracket$ over $\llbracket \Gamma \rrbracket$ that lifts reflexive edges (i.e. edges over reflexive edges are reflexive, this is again the identity extension lemma; see example 8.1.5), and equality of edges (i.e. edges over the same edge in $\llbracket \Gamma \rrbracket$ are necessarily equal, expressing proof irrelevance). A term $\Gamma \mid \Theta \vdash t : T$ is then interpreted as a morphism from $\llbracket \Gamma | \Theta \rrbracket \rightarrow \llbracket \Gamma | T \rrbracket$ in the slice category over $\llbracket \Gamma \rrbracket$; in particular a closed term $\Gamma \mid \cdot \vdash t : T$ is a section of $\llbracket T \rrbracket$.

9.1.3 Dependent Type Theory

As dependent type theory is not just a programming language but also a logic, we can distinguish *three* approaches to parametricity:

- In the external approach, we state *and* prove parametricity theorems in the meta-theory. This is the only possible approach in System F and $F\omega$.
- In the admissible approach, we state the parametricity theorems in some very similar (ideally the same) type system, and we give a metatheoretic proof that every program is parametric. That is, we give a meta-theoretic function that maps program derivation trees to derivation trees of proofs of the statement that the program is parametric.
- In the internal approach, we have an internal operator that essentially inhabits the theorem “every program is parametric”. This operator will again have type dependencies, and self-application should prove that it is parametric. This phenomenon is called iterated parametricity, and generally needs to be modelled in higher-dimensional reflexive graphs, i.e. cubical sets.

External parametricity, with identity extension only for small types Atkey, Ghani, and Johann [AGJ14] have reorganized Atkey’s model [Atk12] to a model of dependent type theory. Essentially, they start from the standard presheaf model of dependent type theory in reflexive graphs [Hof97, Ch. 4] (chapter 4). The idea is that nodes of large types (kinds) represent their elements, whereas edges represent proofs of relatedness (\curvearrowright_κ). For small types, nodes are again elements, but edges should be proofs of heterogeneous equality (\simeq_R , where R is the corresponding edge between the types).

The desired identity extension lemma can now be rephrased as: homogeneous edges (edges living above reflexive edges in the context) should be reflexive. In order to validate this lemma, we could naively require all internal types to be discrete, i.e. to satisfy this condition. However, the problem is that the universe does *not* satisfy it. Indeed, a homogeneous edge in the universe is like a proof of $A \curvearrowright_* B$ in System $F\omega$, which is essentially a relation between A and B . Surely, the existence of a relation $R : \text{Rel}(A, B)$ does not imply that $A = B$ and $R = \text{Eq}_A$. So the universe is not discrete as it has non-reflexive homogeneous edges.

For this reason, Atkey, Ghani, and Johann [AGJ14] only adapt the Hofmann-Streicher universe of small types [HS97] by restricting it to small discrete proof-irrelevant⁵ types. Types in general are allowed to be non-discrete, and hence identity extension is only proven for small types. Proof-irrelevance is required in order to model function types: for function types to be discrete, we either need to work in proof-irrelevant graphs, or we need higher-dimensional structure (cubical sets) in order to reason about equality of functions’ actions on edges.

Writing $e : x \dot{\simeq}_A y$ for a homogeneous edge in type A (which generalizes both $e : x \curvearrowright_A y$ and $e : x \simeq_A y$), and $e : x \dot{\simeq}_R y$ for a heterogeneous edge where $R : A \dot{\simeq}_U B$, we can summarize the behaviour of dependent functions $f : (x : A) \rightarrow B(x)$ in Atkey, Ghani, and Johann [AGJ14] in a single diagram:⁶

$$\begin{array}{ccc}
 x =_A y & \xrightarrow{f=} & f(x) =_{B(x)} f(y) \\
 \sqcup(\mathbf{r}) \downarrow & & \downarrow \sqcup(\mathbf{r}) \\
 e : x \dot{\simeq}_A y & \xrightarrow{f\dot{=}} & f(x) \dot{\simeq}_{B_\pm(e)} f(y)
 \end{array}$$

⁵In the sense of Atkey, Ghani, and Johann [AGJ14]: having a proof-irrelevant edge relation.

⁶The bottom arrow in this diagram can be generalized to act on heterogeneous edges (by replacing A with an edge in the universe); however then the left side of the diagram would be ill-typed. Dependent diagrams are always a bit awkward.

In this diagram, $=_A$ denotes mathematical equality. E.g. the arrow $(\sqcup\langle\mathbf{r}\rangle) : x =_A y \rightarrow x \dot{\div}_A y$ means: if x and y are really the same node of A , then $x\langle\mathbf{r}\rangle$ is an edge of A whose source $x\langle\mathbf{r}\rangle\langle\mathbf{s}\rangle$ equals x and whose target $x\langle\mathbf{r}\rangle\langle\mathbf{t}\rangle$ equals y .

Admissible parametricity The work on admissible parametricity generally uses different techniques and is in this sense much less relevant in this historical resume. We cite some important works for completeness:

- Takeuti [Tak01] gives a parametric translation from every system in the Lambda Cube to a richer system in the Lambda Cube, and proves soundness of identity extension (calling it the “axiom of parametricity”) for small types.
- Bernardy, Jansson, and Paterson [BJP12] give a parametric translation from a general pure type system to (in general) a different pure type system. Identity extension is not considered.
- Keller and Lasson [KL12] give a parametric translation from a variation of the calculus of inductive constructions to itself. They use this as a basis to implement the ParamCoq plugin for Coq [Kel+]. Identity extension is not considered.

Internal parametricity, without identity extension Bernardy, Coquand, and Moulin [BCM15] and Moulin [Mou16] have introduced internal operators that allow the creation of proof terms for parametricity theorems, and provide a model in (unary) cubical sets.

Their system is about unary parametricity and hence cannot feature identity extension, but it can be converted to a binary system straightforwardly in which we could either postulate the identity extension lemma for small types as an axiom⁷, or create a universe of types that satisfy the lemma and is closed under small type formers.

(Binary) cubical sets can be seen as higher-dimensional graphs, which feature not just nodes and edges, but also squares, cubes, and higher-dimensional cubes. This higher-dimensional structure is necessary to model iterated parametricity (see above), as well as to prove the identity extension lemma for the function type if you want to allow parametricity to be applied to proof-relevant relations.

⁷This axiom would be partly justified by a cubical generalization of Atkey, Ghani, and Johann’s model [AGJ14], but Moulin’s model [Mou16] is more subtle in that it uses *refined* presheaves (based on I -sets) to strictify certain isomorphisms related to the internal parametricity operators. To our knowledge no one has created a refined presheaf model of identity extension.

9.2 Parametric Quantifiers: Internal Parametricity with Identity Extension

9.2.1 Motivation

In order to validate the identity extension lemma for all types, rather than just small types, Nuyts, Vezzosi, and Devriese [NVD17a] create a type system ParamDTT that uses modalities to distinguish between parametric, continuous and pointwise functions. These modalities differ in how they act on different flavours of edges:

- **Paths** $p : x \asymp y$ generalize equality of types and heterogeneous equality of terms in System $F\omega$,
- **Bridges** $b : x \frown y$ generalize relatedness of types.

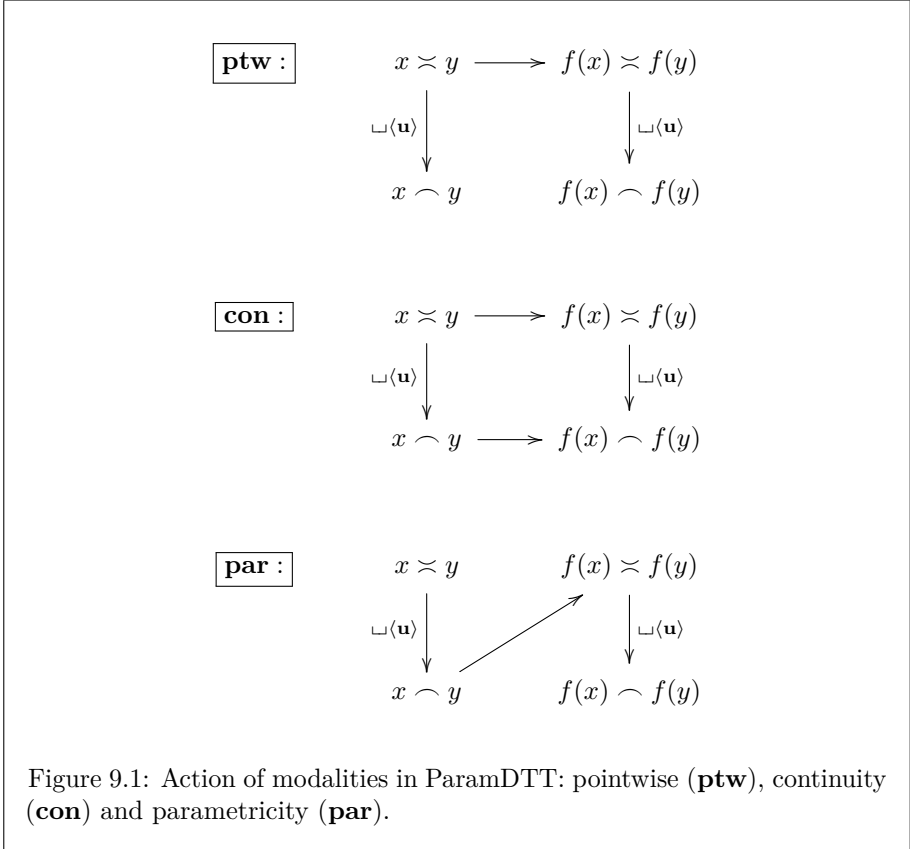
One might hope to give a model in bridge/path reflexive graphs, which would be presheaves over the category BPRG generated by the following diagram and equations:

$$\begin{array}{ccc}
 & \xrightarrow{\mathbf{s}} & \\
 \mathbb{N} & \xleftarrow{\mathbf{r}} \mathbb{P} \xleftarrow{\mathbf{u}} & \mathbb{B} \\
 & \xleftarrow{\mathbf{t}} &
 \end{array}
 \qquad
 \begin{array}{l}
 \mathbf{r} \circ \mathbf{u} \circ \mathbf{s} = 1_{\mathbb{N}}, \\
 \mathbf{r} \circ \mathbf{u} \circ \mathbf{t} = 1_{\mathbb{N}}.
 \end{array}$$

Here, \mathbf{r} expresses that every node is path-equal to itself, \mathbf{u} expresses that when things are path-equal, they are also bridge-related, and \mathbf{s} and \mathbf{t} extract source and target from a bridge. By composing with \mathbf{u} , we can extract source and target of a path, or obtain reflexive bridges.

However, because the bridges in the universe — which will be relations between types — are inherently proof-relevant, we need a model that accommodates proof-relevant parametricity. Furthermore, because the aim is to provide internal parametricity operators, it is desirable to accommodate iterated parametricity. For these two reasons, we need a cubical model. Indeed, ParamDTT is modelled in bridge/path cubical sets, which are presheaves over the category \mathbf{BPCube} which is the free cartesian monoidal category over BPRG with the same terminal object \mathbb{N} . In other words, the objects of \mathbf{BPCube} are finite products of \mathbb{B} and \mathbb{P} and the morphisms are generated by weakening ($\mathbf{r} : \mathbb{P} \rightarrow ()$), exchange ($v \times w \rightarrow w \times v$), contraction ($w \rightarrow w \times w$) and $\mathbf{u} : \mathbb{B} \rightarrow \mathbb{P}$.

Functions $f : (x : A) \rightarrow B(x)$ in ParamDTT are then classified according to how they act on bridges and paths (fig. 9.1).



A **pointwise** function $f : (\text{ptw} \mid x : A) \rightarrow B(x)$ maps path-connected inputs $p : x \asymp y$ to path-connected outputs $f_{\asymp}(p) : f(x) \asymp f(y)$, witnessing that it preserves heterogeneous equality. However, it has no action on bridges $b : x \frown y$, meaning that bridge-related inputs may be mapped to arbitrary outputs. In particular, pointwise quantification over types has no action on relations between types. The only way to assert bridge-connected outputs from a pointwise function, is by feeding it path-connected inputs; then $f_{\asymp}(p)\langle \mathbf{u} \rangle$ is the desired bridge. The pointwise modality may be used to soundly assume the law of excluded middle:

$$(\text{ptw} \mid X : \mathbf{U}) \rightarrow X \uplus (X \rightarrow \text{Empty}).$$

Stating it with the parametric modality would imply that either all types are inhabited or all types are empty.

A **continuous** function $f : (\text{con} \mid x : A) \rightarrow B(x)$ sends path-connected inputs $p : x \asymp y$ to path-connected outputs $f_{\asymp}(p) : f(x) \asymp f(y)$, and bridge-connected inputs $b : x \frown y$ to bridge-connected outputs $f_{\frown}(b) : f(x) \frown f(y)$. Thus, it preserves heterogeneous equality *and* relatedness. This corresponds to the behaviour of a type-level operation in System F ω .

A **parametric** function $f : (\text{par} \mid x : A) \rightarrow B(x)$ sends bridge-connected inputs $b : x \frown y$ to path-connected outputs $f_{\frown}(b) : f(x) \asymp f(y)$. Hence, it also sends paths $p : x \asymp y$ to paths $f_{\frown}(p(\mathbf{u})) : f(x) \asymp f(y)$ and bridges $b : x \frown y$ to bridges $f_{\frown}(b)\mathbf{u} : f(x) \frown f(y)$. In particular, a function $f : (\text{ptw} \mid X : \mathbf{U}) \rightarrow T(X)$ sends a relation $B : X \frown Y$ to a proof $f_{\frown}(B) : f(X) \asymp f(Y)$ that the instantiations $f(X)$ and $f(Y)$ are heterogeneously equal according to the relation $T_{\frown}(B) : T(X) \frown T(Y)$.⁸

Remark 9.2.1. We remark that Vezzosi’s ParamDTT implementation `agda-parametric` [NVD17a] features three additional and at the time experimental modalities, for which we need to include a trivially satisfied relation sending x and y to the singleton \top (fig. 9.2): irrelevance (**irr**), shape-irrelevance (**shi**) [AVW17], and the join of shape-irrelevance and parametricity (**shi** \vee **par**).

9.2.2 The Mode Theory and the Corresponding Instance of MTT

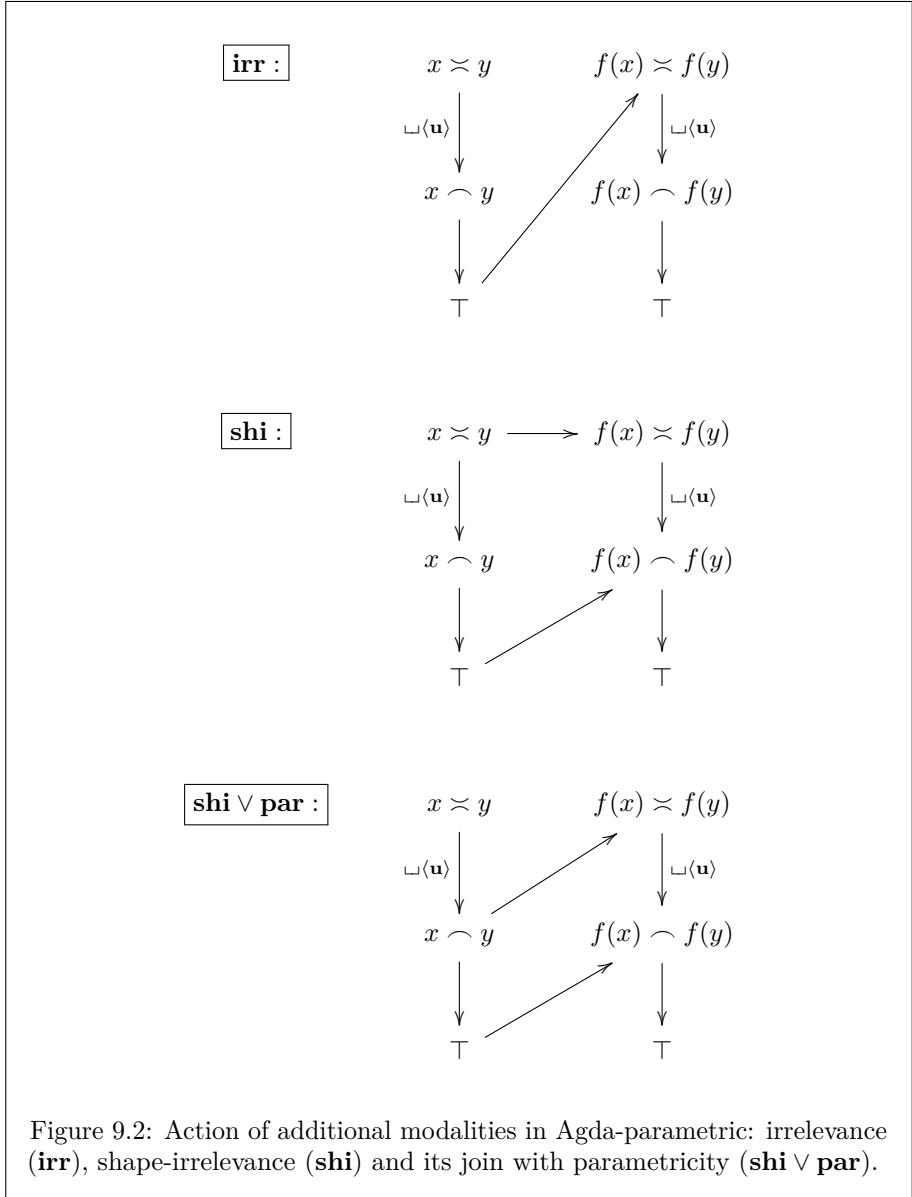
Definition 9.2.2. The mode theory for ParamDTT is the poset-enriched category

- that has a single object $*$,
- such that $\text{Hom}(*, *) = \{\text{ptw} < \text{con} < \text{par}\}$,
- where **con** is the identity and composition is given by

$\downarrow \circ \rightarrow$	ptw	con	par
ptw	ptw	ptw	par
con	ptw	con	par
par	ptw	par	par

It is clear that the identity function is continuous. The modality of a composite function, can be found by pasting together the above diagrams, which yields the above composition table.

⁸The codomain T is required to be continuous for the parametric function type to be well-formed.



Note also that, using $\sqcup\langle \mathbf{u} \rangle$, we can prove that all parametric functions are continuous. All continuous functions are clearly also pointwise (as we can forget the action on bridges), which confirms the postulated order on modalities.

Theorem 9.2.3. The instantiation of MTT with the mode theory for ParamDTT yields a type system ParamMTT which can be modelled in the category $\text{Psh}(\text{BPCube})$ as an instance of theorem 5.3.3. ParamMTT is *not* the system ParamDTT [NVD17a].

Remark 9.2.4. ParamDTT deviates from MTT in two important respects:

- It uses eager left division $\mu \setminus \Gamma$, rather than lazy locks Γ, \mathbf{a}_μ (see section 5.3.7),
- It features a parametric type decoding rule

$$\frac{\mathbf{par} \setminus \Gamma \vdash T : \mathbf{U}}{\Gamma \vdash T \text{ type}} \tag{9.1}$$

which has the effect of making variables available in a term and its type (or more precisely its type’s code) by a different modality (e.g. parametric functions have continuous type).

Furthermore, ParamMTT lacks all system-specific features, such as internal parametricity operators.

Lemma 9.2.5. We have three adjoint functors $\int \dashv \mathbf{b} \dashv \mathbf{\sharp} : \text{BPCube} \rightarrow \text{BPCube}$ which are the cartesian monoidal functors such that:

$$\begin{aligned} \int \mathbb{P} &= (), & \mathbf{b} \mathbb{P} &= \mathbb{B}, & \mathbf{\sharp} \mathbb{P} &= \mathbb{P}, \\ \int \mathbb{B} &= \mathbb{B}, & \mathbf{b} \mathbb{B} &= \mathbb{B}, & \mathbf{\sharp} \mathbb{B} &= \mathbb{P}. \end{aligned}$$

Proof. Left as an exercise to the reader, but note that these functors are definable on BPRG and that the adjunctions can be proven there and carry over. □

Proof of theorem 9.2.3. We need to find a strict 2-functor $J : \mathcal{M} \rightarrow \text{Cat}$ where \mathcal{M} is the mode theory for ParamDTT, such that $J(\mu)_*$ is a good interpretation of the dependent right adjoint. Indeed, then the locks are interpreted by $J(\mu)^*$ which will also be a strict 2-functor $\mathcal{M} \rightarrow \text{Cat}$. Clearly, we will take $J(*) = \text{BPCube}$.

Before we define the action of J on morphisms, we will define $K : \mathcal{M}^{\text{coop}} \rightarrow \text{Cat}$, and then we will construct J so that $J(\mu) \dashv K(\mu)$. This means that $K(\mu)^*$ will be naturally isomorphic to $J(\mu)_*$. Of course all of this is only well typed assuming we take $K(*) = J(*) = \text{BPCube}$.

In general, $K(\mu)\mathbb{B}$ should be the weakest relation (represented by an object of \mathbf{BPCube}) such that a μ -modal function will send $K(\mu)\mathbb{B}$ -related inputs to bridge-related outputs. Similarly, $K(\mu)\mathbb{P}$ should be the weakest relation such that a μ -modal function will send $K(\mu)\mathbb{P}$ -related inputs to path-equal outputs.

For **con**, which is the identity modality, this means $K(\mathbf{con}) = 1$. For parametricity, a bridge in the domain is sufficient to guarantee either a path or a bridge in the codomain, so we take $K(\mathbf{par}) = \flat$. For pointwise functions, we need a path in the domain to guarantee either a path or a bridge in the codomain, so we take $K(\mathbf{ptw}) = \sharp$. This is immediately seen to reverse 2-cells.

For J then, we simply take the left adjoints:

$$J(\mathbf{con}) = 1, \quad J(\mathbf{par}) = \flat, \quad J(\mathbf{ptw}) = \sharp. \quad \square$$

Let us now map concepts from System $F\omega$ to those of ParamDTT by looking for similarities between the corresponding diagrams. Type level operators in $F\omega$ become continuous functions in ParamDTT. Parametric functions in System $F\omega$ become parametric functions in ParamDTT. One can imagine a modal extension of System $F\omega$ that allows ad hoc polymorphism, so that we can have a **typecase** operator or postulate a non-parametric law of excluded middle. The latter is sound in ParamDTT.

When we consider term level functions in System $F\omega$, we notice an awkward aspect of the model of ParamDTT, namely that small types, too, come equipped with a path (\simeq) and a bridge (\frown) relation. In System $F\omega$ on the other hand, we could only consider heterogeneous equality (\simeq) for elements of small types. In fact, we have no need for these two relations, and unless we allow HITs with bridge constructors, all small closed types will be bridge-discrete, meaning essentially that $\sqsubset(\mathbf{u})$ is an isomorphism. An immediate consequence is that if a function's domain is a small closed type, then its modality does not matter. However, the type system does distinguish between the corresponding function types and has no way of coercing upstream against the order on the modality monoid. This shortcoming is addressed in Nuyts and Devriese [ND18a] (section 9.3) by having a separate mode for types that have no bridge relation, thus conflating the different modalities.

Remark 9.2.6. If we add **shi**, **irr** and **shi** \vee **par** (remark 9.2.1), then the inequality relation is given by

$$\mathbf{ptw} < \mathbf{con} < \begin{array}{c} \mathbf{par} \\ \mathbf{shi} \end{array} < (\mathbf{shi} \vee \mathbf{par}) < \mathbf{irr}, \quad (9.2)$$

and **shi** and **par** are incomparable. Composition is given by

$\downarrow \circ \rightarrow$	ptw	con	par	shi	shi \vee par	irr
ptw	ptw	ptw	par	ptw	par	irr
con	ptw	con	par	shi	shi \vee par	irr
par	ptw	par	par	irr	irr	irr
shi	shi	shi	shi \vee par	shi	shi \vee par	irr
shi \vee par	shi	shi \vee par	shi \vee par	irr	irr	irr
irr	irr	irr	irr	irr	irr	irr

Since **ptw** \circ **shi** = **ptw** < **con** and **con** < **shi** = **shi** \circ **ptw**, we see that **ptw** \dashv **shi**. Furthermore, **shi** \vee **par** = **shi** \circ **par** and **irr** = **par** \circ **shi**. These observations inspire us to extend the semantics from theorem 9.2.3 with:

$$J(\mathbf{shi}) = \sharp, \quad J(\mathbf{shi} \vee \mathbf{par}) = \sharp \circ \int, \quad J(\mathbf{irr}) = \int \circ \sharp.$$

Together, these are all 6 ‘relation shifting’ modalities whose modal functions still preserve path-equality. If we want to also classify functions that do not preserve path-equality, then we get 4 more modalities, but their locks cannot be interpreted as central liftings (rather they are left liftings, which do not constitute a *strict* 2-functor), so we would have to rely on our strictification theorem [Gra+20a] to build a model.

9.2.3 Extending the MTT instance to ParamDTT \blacktriangleleft

While ParamMTT is not ParamDTT, we can extend it soundly and come pretty close. The main remaining differences will be:

- The use of locks,
- That face restrictions on the context will have a modality annotation and be subject to locks (see section 6.5), which we consider an improvement over ParamDTT proper.

Theorem 9.2.7. We can soundly extend ParamMTT to a system ParamDTT \blacktriangleleft by adding:

1. Bridge interval variables, face propositions, Glue- and Weld-types [NVD17a],
2. A judgement form for discrete types $\Gamma \vdash T \text{dtype}_\ell @ *$ which is closed under discreteness-preserving type formers with modality annotations as in MTT and such that

$$\frac{\Gamma \vdash T \text{dtype}_\ell @ *}{\Gamma \vdash T \text{type}_\ell @ *} \quad (9.3)$$

3. The degeneracy axiom, stating that homogeneous paths in discrete types are constant,⁹
4. Parametric existential quantifiers,
5. A universe $\vdash \mathbf{U}_\ell^{\text{DD}} \text{dtype}_{\ell+1} @ *$ which is closed under discreteness-preserving type formers with modality annotations as in ParamDTT, which features a parametric decoding rule

$$\frac{\Gamma, \mathbf{U}_{\text{par}} \vdash T : \mathbf{U}_\ell^{\text{DD}} @ *}{\Gamma \vdash \text{El } T \text{ type}_\ell @ *}. \quad (9.4)$$

Note that discreteness-preserving type formers most notably exclude the Hofmann-Streicher universe and $\langle \text{par} \mid \perp \rangle$,¹⁰ which is why parametric existentials are explicitly listed as a separate addition.

Proof. 1. The bridge interval is simply interpreted by $\mathbf{y}(\mathbb{B})$. Glue and Weld exist in any presheaf category. We refer to the original work [NVD17a; Nuy18a] for details, to example 6.1.6 for a sketch of how to type-check and to section 6.5 for the interaction with modalities.

2. The semantics of this judgement is simply a type which satisfies the degeneracy axiom.

Alternatively, we may observe that discreteness is the right class of a robust non-damped OFS [Nuy18a], internalize the (idempotent) discrete replacement monad \mathfrak{f} (RR), and define a discrete type to be an algebra for \mathfrak{f} (section 8.6), i.e. a type $T \cong \mathfrak{f}T$.

3. This is then trivial (using the first option in the previous point) or follows if we postulate the degeneracy axiom for the discrete replacement.
4. We can simply take the Σ -type over $\langle \text{par} \mid A \rangle$ and then take the discrete replacement of that.
5. Using standard techniques, either in the model or as an instance of \mathbf{U}^{NFF} (section 8.6), we obtain a closed type \mathbf{U}^{NDD} that is a classifier for the discrete typing judgement but is itself not discrete and still has a continuous decoding rule [Nuy18a]. It's symbol stands for 'non-discrete universe of discrete types'. Then we define $\mathbf{U}^{\text{DD}} := (\mathfrak{f})^* \mathbf{U}^{\text{NDD}}$, as motivated below. \square

Remark 9.2.8 (Construction of \mathbf{U}^{DD}). The non-discrete universe \mathbf{U}^{NDD} behaves like the Hofmann-Streicher universe, only it classifies discrete types.

⁹This is the internalization of the identity extension lemma.

¹⁰as well as $\langle \text{shi} \mid \perp \rangle$, $\langle \text{shi} \vee \text{par} \mid \perp \rangle$ and $\langle \text{irr} \mid \perp \rangle$.

A bridge $B : X \frown_{\text{UNDD}} Y$ then encodes a notion of heterogeneous bridges $(x : X) \frown_B (y : Y)$, and path $P : X \succ_{\text{UNDD}} Y$ encodes notions of paths $(x : X) \succ_P (y : Y)$ and also a notion of bridges $(x : X) \frown_{P(\mathbf{u})} (y : Y)$. The latter is inevitable, as P must contain all information needed to form the bridge $P(\mathbf{u}) : X \frown_{\text{UNDD}} Y$. It is immediately clear that the existence of a P does not assert that $X = Y$, i.e. UNDD is not itself discrete.

In the desired discrete universe of discrete types U^{DD} , all paths are reflexive, i.e. we would like to have $(\mathbb{P} \triangleright \text{U}^{\text{DD}}[_]) = (()\triangleright \text{U}^{\text{DD}}[_])$.

Meanwhile, we want to model the parametric decoding rule by making sure that there exists a function $\text{El} : \langle \text{par} \mid \text{U}^{\text{DD}} \rangle \rightarrow \text{UNDD}$. This means that a bridge in U^{DD} must be (at least) a path in UNDD . We can achieve this if $(\mathbb{B} \triangleright \text{U}^{\text{DD}}[_]) = (\mathbb{P} \triangleright \text{UNDD}[_])$:

$$\begin{array}{ccccc}
 X \succ_{\text{U}^{\text{DD}}} Y & \equiv & X =_{\text{UNDD}} Y & & X \succ_{\text{UNDD}} Y \\
 \sqsubset(\mathbf{u}) \downarrow & & \sqsubset(\mathbf{r}) \downarrow & \nearrow \text{El} \sim & \downarrow \sqsubset(\mathbf{u}) \\
 X \frown_{\text{U}^{\text{DD}}} Y & \equiv & X \succ_{\text{UNDD}} Y & \boxed{\text{par}} & X \frown_{\text{UNDD}} Y
 \end{array}$$

Both equations are satisfied by taking $\text{U}^{\text{DD}} = (\#\int)^* \text{UNDD}$, since

$$\#\int \mathbb{P} = \#\int() = (), \quad \#\int \mathbb{B} = \#\int \mathbb{P} = \mathbb{P}, \tag{9.5}$$

(and $\#\int$ is the unique cartesian monoidal functor respecting these equations).

The attentive reader might wonder why we found it appropriate to discard the bridge relation \frown_{UNDD} when building U^{DD} . The unsatisfactory answer is that the path relation \succ_{UNDD} contains more information and that we ran out of slots so we had to discard something. An issue that can be traced back to this discarding, is that the internal parametricity operators of ParamDTT have an extremely contagious pointwise dependency that essentially renders proofs of parametricity theorems non-parametric themselves, getting in the way of iterated parametricity despite having a cubical model.

9.2.4 Wrapping up

In ParamDTT, ParamMTT and ParamDTT \blacktriangle , we see two important causes of discomfort: we have too many relation slots in small types (which feature an unnecessary bridge relation), and we have one too few in the universe. In Degrees of Relatedness (section 9.3), small types are equipped with just a single

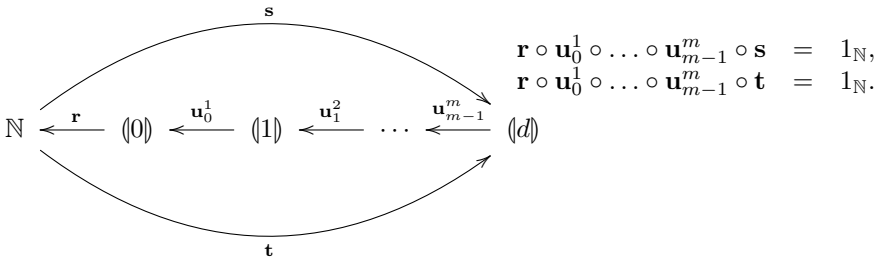
relation, and every universe has one relation slot more than the types that it classifies.

9.3 Degrees of Relatedness

In comparison to ParamDTT [NVD17a], the type system RelDTT [ND18a] makes two improvements:

- It officially contains modalities that interact with the trivially and uniquely provable relation \top (which were already available in `agda-parametric` but not in ParamDTT or its model),
- It addresses the aforementioned shortcomings of ParamDTT by moving to a multimode system in which types come equipped with a different number of relations, depending on their mode.

We focus on the second improvement. The modes of RelDTT (called **depths**) are integers starting from -1 and types of mode m are equipped with $m + 1$ relations called \frown_0 through \frown_m . The mode m segment of the type system is modelled in ‘depth m cubical sets’, which are presheaves over DCube_m , the free cartesian monoidal category (with same terminal object) over RG_m , which is generated by:



By convention, DCube_{-1} is the point category.

The modalities $\mu : m \rightarrow n$ will be, essentially, all diagrams from m ordered relations (and \top) to n ordered relations (and \top) such that a 0-edge in the domain always gives rise to a 0-edge in the codomain, and such that we can also map from \top to \top .

A succinct way to denote such a diagram is by answering, for all $i = 0 \dots n$, the question: how related do the arguments need to be, if I want the results to be i -related? This gives rise to a nondecreasing function $\{0 < 1 < \dots < n\} \rightarrow \{0 < 1 < \dots < m < \top\}$. Hence, we define:

Definition 9.3.1. The mode theory for RelDTT is the poset-enriched category

- whose objects are integers starting from -1 ,
- such that $\text{Hom}(m, n)$ is the set of nondecreasing functions

$$\mu : \{0 < 1 < \dots < n\} \rightarrow \{0 < 1 < \dots < m < \top\} : i \mapsto i \cdot \mu,$$

also denoted $\langle 0 \cdot \mu, \dots, n \cdot \mu \rangle$,

- where the identity modality **con** is given by $i \cdot \mathbf{con} = i$ and composition is given by

$$i \cdot (\nu \circ \mu) = \begin{cases} (i \cdot \nu) \cdot \mu & \text{if } i \cdot \nu \neq \top, \\ \top & \text{if } i \cdot \nu = \top, \end{cases}$$

where $\mu \leq \nu$ whenever $i \cdot \mu \leq i \cdot \nu$ for all i .

Example 9.3.2. We refer to Nuyts and Devriese [ND18a] for a compendium of interesting modalities. Here, we just mention parametricity $\mathbf{par} : m + 1 \rightarrow m$ for which $i \cdot \mathbf{par} = i + 1$ and its right adjoint **structurality** $\mathbf{str} : m \rightarrow m + 1$ for which $0 \cdot \mathbf{str} = 0$ and $(i + 1) \cdot \mathbf{str} = i$. Their action is depicted diagrammatically in fig. 9.3.

Theorem 9.3.3. The instantiation of MTT with the mode theory for RelDTT yields a type system RelMTT which can be modelled in the categories $\text{Psh}(\text{DCube}_m)$ as an instance of theorem 5.3.3. RelMTT is *not* the system RelDTT [ND18a].

Remark 9.2.4 applies also for RelMTT vs. RelDTT.

Lemma 9.3.4. The modalities $\mu : m \rightarrow n$ are, by Galois connection $(\kappa \dashv \mu)$, in 1-1 correspondence with nondecreasing functions

$$\kappa : \{0 < 1 < \dots < m\} \rightarrow \{(\text{=}) < 0 < 1 < \dots < n\} : j \mapsto j \cdot \kappa,$$

which are called **contramodalities**. □

Proof of theorem 9.3.3. We define the 2-functor $J : \mathcal{M} \rightarrow \text{Cat}$ that sends modes to base categories. Of course, we need $J(m) = \text{DCube}_m$. In order to define $J(\mu)$, let $\kappa \dashv \mu$ be the corresponding contramodality. Since $J(\mu)_*$ is going to be the interpretation of the DRA of μ , we can think of $J(\mu)^*$ as the interpretation of κ . Hence, we define $J(\mu)$ to be the cartesian monoidal functor that sends (i) to $(i \cdot \kappa)$ if $i \cdot \kappa \neq (\text{=})$, and to the terminal object otherwise. □

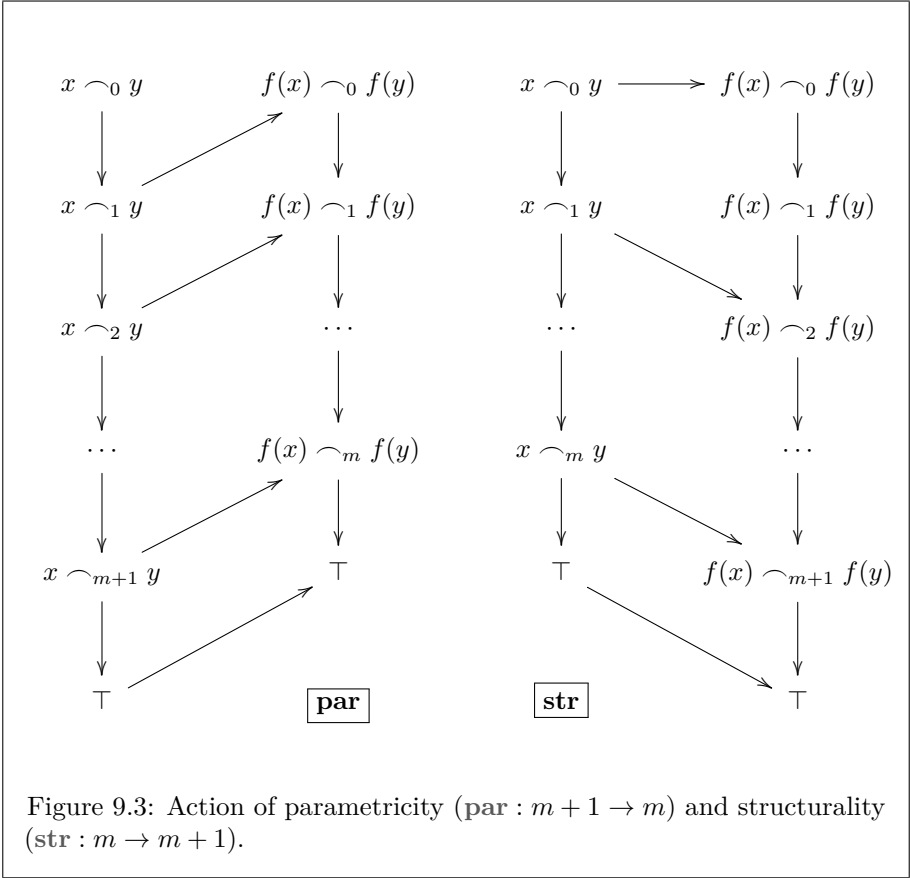


Figure 9.3: Action of parametricity ($\text{par} : m + 1 \rightarrow m$) and structurality ($\text{str} : m \rightarrow m + 1$).

Proposition 9.3.5. ParamMTT is a subsystem of RelMTT with the same semantics. Concretely, we have a functor $I : \mathcal{M}_{\text{ParamDTT}} \rightarrow \mathcal{M}_{\text{RelDTT}}$, invertible on Hom-posets, from the mode theory of ParamDTT to the mode theory of RelDTT such that the following diagram commutes if we identify $\text{BPCube} = \text{DCube}_1$:

$$\begin{array}{ccc}
 \mathcal{M}_{\text{ParamDTT}} & \xrightarrow{I} & \mathcal{M}_{\text{RelDTT}} \\
 & \searrow J & \downarrow J \\
 & & \text{Cat}
 \end{array}$$

This still works if we include the modalities from remark 9.2.6.

Proof. The following definition of I does the job:

$$I(*) = 1$$

$$I(\mathbf{ptw}) = \langle 0, 0 \rangle, \quad I(\mathbf{con}) = \langle 0, 1 \rangle, \quad I(\mathbf{par}) = \langle 1, 1 \rangle, \quad \square$$

$$I(\mathbf{shi}) = \langle 0, \top \rangle, \quad I(\mathbf{shi} \vee \mathbf{par}) = \langle 1, \top \rangle, \quad I(\mathbf{irr}) = \langle \top, \top \rangle.$$

Again, we can extend RelMTT to something that differs from RelDTT mainly in the use of locks vs. left division:

Theorem 9.3.6. We can soundly extend RelMTT to a system RelDTT $_{\blacksquare}$ by adding:

1. Interval variables, face propositions, Glue- and Weld-types,
2. A judgement form for discrete types $\Gamma \vdash T \text{dtype}_{\ell} @ m$ which is closed under discreteness-preserving type formers with modality annotations as in MTT and such that

$$\frac{\Gamma \vdash T \text{dtype}_{\ell} @ m}{\Gamma \vdash T \text{type}_{\ell} @ m} \quad (9.6)$$

3. The degeneracy axiom, stating that homogeneous 0-edges in discrete types are constant,¹¹ or at mode -1 that elements of the same discrete type are equal,
4. Modal existential quantifiers for modalities μ such that $0 \cdot \mu \neq 0$,
5. A universe $\vdash \mathbf{U}_{\ell}^{\text{DD}} \text{dtype}_{\ell+1} @ m + 1$ which is closed under discreteness-preserving type formers with modality annotations as in RelDTT, which features a parametric decoding rule

$$\frac{\Gamma, \blacksquare_{\text{par}} \vdash T : \mathbf{U}_{\ell}^{\text{DD}} @ m + 1}{\Gamma \vdash \text{El } T \text{dtype}_{\ell} @ m}. \quad (9.7)$$

Note that discreteness-preserving type-formers most notably exclude the Hofmann-Streicher universe and $\langle \mu \mid \sqsubset \rangle$ when $0 \cdot \mu \neq 0$, which is why existentials for those modalities are explicitly listed as a separate addition.

Proof. All points but the last are proved as in theorem 9.2.7.

Using standard techniques, we obtain a non-discrete universe of discrete types $\mathbf{U}_{\ell}^{\text{NDD}}$ at every mode m . Then we define $\mathbf{U}_{\ell}^{\text{DD}} := J(\mathbf{par})^* \mathbf{U}_{\ell}^{\text{NDD}}$ (where J is defined as in the proof of theorem 9.2.3), which lives at mode $m + 1$. Note

¹¹This is the internalization of the identity extension lemma.

that $J(\mathbf{par})$ is the interpretation of \mathbf{par} and sends (0) to the terminal object and $(i+1)$ to (i) . Hence, the 0-edges of $\mathbf{U}_\ell^{\text{DD}}$ are the points of $\mathbf{U}_\ell^{\text{NDD}}$ (so it is discrete) and we shove aside all other relations so that the $(i+1)$ -edges of $\mathbf{U}_\ell^{\text{DD}}$ are the i -edges of $\mathbf{U}_\ell^{\text{NDD}}$. This indexation shift allows for a parametric function $\langle \mathbf{par} \mid \mathbf{U}_\ell^{\text{DD}} \rangle \rightarrow \mathbf{U}_\ell^{\text{NDD}}$. \square

In this system, we can think of terms at mode -1 as proofs, at mode 0 as programs, at mode 1 as types, at mode 2 as kinds, etc.

9.4 MTT as an Internal Language of the Model

As mentioned, neither ParamDTT nor RelDTT are themselves instances of MTT, their most stark deviation being the parametric type decoding rule which causes both systems to enforce different modalities for terms and their types (e.g. parametric functions have continuous types and irrelevant functions have shape-irrelevant types).

Fleshing out the semantics of both systems was a major effort and produced a technical report [Nuy18a], some parts of which could be classified as ‘write-only’. It would have been desirable to carry out these proofs in a proof-assistant, i.e. internal to another type system. For the authors, this has the advantage that a lot of tedious bookkeeping could be done automatically, and RelDTT’s end users would of course have more confidence in the system.

As both models start with an instantiation of theorem 5.3.3, MTT seems quite well-suited as a metatheory in which discreteness, \mathbf{U}^{NDD} and \mathbf{U}^{DD} could be defined, and ParamDTT and RelDTT could be shallowly embedded. This is in fact one of the central motivations behind the Menkar project [Nuy19].

There is one important difficulty, namely that the creation of \mathbf{U}^{DD} out of \mathbf{U}^{NDD} needs to insert an equality relation in relation slot 0 , which the internal modalities of ParamDTT and RelDTT are unable to do. A contramodality $\kappa \dashv \mu$ has the capacity to do so, however. If in the above construction, we set $\llbracket \mathbf{par}_\kappa \rrbracket = J(\mu)_!$, then an i -edge in $\langle \kappa \mid A \rangle$ is an $(i \cdot \kappa)$ -edge in A , or an equality proof if $i \cdot \kappa = (=)$.

On the other hand, modalities such as irrelevance and shape-irrelevance interact with \top and as such are not contramodalities. So an ideal metatheory in which to embed RelDTT, has as its weak DRAs both central and right liftings of base category functors, which may compose to weak DRAs that are neither. As such, it becomes difficult to provide semantics that are strictly functorial on

locks, and we need to invoke our strictification theorem [Gra+20a] to model an appropriate metatheory.

Definition 9.4.1. The mode theory for the model of RelDTT is the poset-enriched category

- whose objects are integers starting from -1 ,
- such that $\text{Hom}(m, n)$ is the set of nondecreasing functions

$$\mu : \{0 < 1 < \dots < n\} \rightarrow \{ (=) < 0 < 1 < \dots < m < \top \} : i \mapsto i \cdot \mu,$$

- where the identity modality **con** is given by $i \cdot \mathbf{con} = i$ and composition is given by

$$i \cdot (\nu \circ \mu) = \begin{cases} (i \cdot \nu) \cdot \mu & \text{if } i \cdot \nu \notin \{=, \top\}, \\ (=) & \text{if } i \cdot \nu = (=), \\ \top & \text{if } i \cdot \nu = \top, \end{cases}$$

where $\mu \leq \nu$ whenever $i \cdot \mu \leq i \cdot \nu$ for all i .

Theorem 9.4.2. There is a model in categories equivalent to DCube_m for MTT over the mode theory for the model of RelDTT.

Proof. Every modality of this mode theory can be written as a composite of a modality μ of RelDTT and a contramodality κ of RelDTT. We can interpret $\llbracket \mathbf{a}_\mu \rrbracket = J(\mu)^*$ and $\llbracket \mathbf{a}_\kappa \rrbracket = J(\nu)_!$ where $\kappa \dashv \nu$. One can show that this constitutes a pseudofunctor.¹² As central and right liftings are always DRAs, we can invoke the MTT strictification theorem [Gra+20a]. \square

9.5 Parametricity Features and their Requirements

In this section, we discuss a number of parametricity features and what their requirements are in terms of complexity of the type system and its model.

Parametricity of System F Parametricity of (predicative [Lei91]) System F can be modelled using **Reynolds' original set model** [Rey83], which consists of an object interpretation and a relational interpretation relating 2 (or n) instances of the object interpretation. Reflexivity and the identity extension lemma are proven a posteriori by induction on the type.

¹²E.g. by building a pseudonaturally equivalent strict 2-functor sending modes to categories of presheaves valued not in **Set** but in the category of setoids and functions on equivalence classes. This is possible because the locks for contramodalities are always quotients of central liftings.

Parametricity of System $F\omega$ As already mentioned in section 9.1.2, Reynolds’ model can be structured as a **reflexive graph** model, which allows straightforward generalization to a model of (predicative) System $F\omega$ [Atk12]. Reflexivity is now built into the model, and identity extension follows from the fact that type operators preserve reflexivity (and that reflexivity in the kind $*$ of types sends a node/type T to the edge/relation Eq_T).

Dependently typed parametricity **Proof-irrelevant** parametricity (relational parametricity w.r.t. proof-irrelevant relations) can again be proven by a **reflexive graph** model [AGJ14], which is mostly an instance of the general presheaf model of DTT (chapter 4). Identity extension does however not hold for the universe, or for most types built using the universe. So we only get **identity extension for small types**.

Proof-relevant parametricity In proof-relevant parametricity, we allow the use of proof-relevant relations. This can no longer be modelled in a reflexive graph model, as it does not validate identity extension for Π -types. Indeed, identity extension requires that two functions $f, g : (x : A) \rightarrow B(x)$ are related if and only if they are equal. But equality implies that f and g have the same action on edges. In proof-irrelevant parametricity, this is automatic, because proof-irrelevance requires that there is always at most one well-typed edge. In proof-relevant parametricity, it is instead required that ‘relatedness’ of f and g says something about their action on edges. Thus, we do not only need edges to talk about how points are related, but also squares to talk about how edges are related. Introducing squares, creates a need for cubes to talk about how squares are related (unless we require proof-irrelevance at the square level), which requires 4-cubes, etc. Thus, **proof-relevant** parametricity with **identity extension for small types**, should be modelled in a **cubical set** model, which is again an instance of a presheaf model of DTT (chapter 4). We are unaware of a paper that actually does precisely that, but such a model would be a simplification of that of ParamDTT [NVD17a], a straightforward generalization of that of Atkey, Ghani, and Johann [AGJ14], or just the binary version of the model of Moulin [Mou16] and Bernardy, Coquand, and Moulin [BCM15] on which we actually still have to prove that discrete types are closed under small type formers.

Internal/iterated parametricity In type systems with internal parametricity, there is typically a type $x \frown_T y$ of proofs that x and y are related. Typically, the internal parametricity operators can be applied to the above type (unless the type system takes special measures to prevent it, such as including a pointwise

modality in ParamDTT [NVD17a], which we consider a flaw and not a feature and which we rectified in RelDTT [ND18a]), allowing internal reasoning about relatedness of proofs of relatedness. We call this iterated parametricity.

Actually, internal or iterated parametricity does not add much to the concerns already mentioned: the type $x \curvearrowright_T y$ brings the same challenges as the type $(x : A) \rightarrow B(x)$ when A has a non-trivial edge. As such, internal/iterated parametricity requires a cubical model if you also want it to be proof-relevant, but not otherwise.

Identity extension for large types If we want to attain identity extension for large types in DTT, a deep intervention in how the type system works is appropriate. In order to see why, we consider a simple example. In System F, the type $\forall X.B$ (with B closed) only contains constant functions. This is a parametricity result. In DTT, this type translates to $(X : \mathbb{U}_\ell) \rightarrow B$. Now if we instantiate $B := \mathbb{U}_\ell$, then we can inhabit the type with the identity function $\lambda(X : \mathbb{U}).X$, which is not constant.

There are several perspectives to understand this problem. A first perspective is by observing that the Π -types of DTT generalize both the parametric \forall -type former from System F (which forbids its inhabiting functions to inspect their argument) and the non-parametric type/kind former \rightarrow (which allows its inhabiting functions to inspect their arguments). As such, the Π -type has non-parametric inhabitants coming from the arrow type, violating the parametricity results that held for the \forall -type. In ParamDTT [NVD17a], we rectified this by reintroducing a parametric function type $(\text{par} \mid x : A) \rightarrow B(x)$ containing only parametric functions.

Alternatively, we may consider types' levels. Concretely, we expect inhabitants of $(X : \mathbb{U}_\ell) \rightarrow B$ (with B closed) to be constant *assuming that B has level ℓ* , thus ruling out $B = \mathbb{U}_\ell$ which has level $\ell + 1$. This would follow from a generalization of Atkey, Ghani, and Johann's result [AGJ14], which asserts identity extension for small types. However, if we inspect their model, it turns out that smallness is not really the property they rely on; rather, they rely on the fact that *discrete* types satisfy identity extension and that discrete types are closed under small type formers. It is to be expected that future work on parametricity, which may feature relational higher inductive types (see the HoTT-book for HITs [Uni13]), will provide small type formers which do not preserve discreteness. Thus, it seems wise to distinguish a type's level (a stratification useful for guaranteeing predicativity) from its *depth* [ND18a] (i.e. its relational complexity).

RelDTT [ND18a] cleanly unifies these two perspectives: the function type $(\mu \mid x : A) \rightarrow B(x)$ gets annotated with a modality μ . The choice of possible

modalities $\mu : m \rightarrow n$ depends on the depths m of A and n of B . If A and B have the same depth, then continuity $\text{con} : m \rightarrow m$ becomes an option and we can consider functions that inspect their argument. If B has depth $n = m - 1$, then we can form the type $(\text{par} \mid x : A) \rightarrow B(x)$ whose elements satisfy free theorems [Wad89], but thanks to the wealth of modalities in RelDTT we can still consider ad hoc polymorphic functions $(\text{hoc} \mid x : A) \rightarrow B(x)$ allowing e.g. postulation of the law of excluded middle $(\text{hoc} \mid X : \mathbb{U}) \rightarrow X \uplus (X \rightarrow \text{Empty})$ without breaking parametricity in general.

In summary, if we care for dependently typed parametricity with **identity extension even for large types**, then we should be looking towards modalities or at least a stratification of types based on their relational complexity (which may or may not be decoupled from the level). I would argue that RelDTT is so general that such modal or stratified systems should be almost always explicable as a subsystem of RelDTT, e.g.:

Theorem 9.5.1. There exists a non-trivial model of DTT with Agda-style cumulativity¹³, in which any function $f : \mathbb{U}_\ell \rightarrow A$ where $A : \mathbb{U}_\ell$, is constant.

Proof. DTT translates to a subsystem of RelDTT where types of level ℓ automatically also have depth ℓ and all functions are mediated by $\text{par}^k : m+k \rightarrow m$ or $\text{str}^k : m \rightarrow m+k$ or $\text{con} : m \rightarrow m$, and where $\text{Lift } X := \langle \text{str} \mid X \rangle$.

Under this translation, f ends up having type $(\text{par} \mid \mathbb{U}_\ell) \rightarrow A$. Now we can build a 1-edge between any two types in \mathbb{U}_ℓ (e.g. by welding over the empty type), yielding a 0-edge in A , which is constant by the degeneracy axiom of RelDTT [ND18a]. \square

I hope that this exposition makes it clear that, *in general*, dependently typed parametricity does *not* require modalities, even if we require identity extension for small types.

¹³i.e. if $X : \mathbb{U}_\ell$ then $\text{Lift } X : \mathbb{U}_{\ell+1}$ and $\text{Lift } X \cong X$.

Chapter 10

Conclusion

In this conclusive chapter, we discuss possible future work, the practical implications of this thesis, and a philosophical question that may have arisen.

10.1 Future Work

In section 10.1.1, we discuss how, using the contributions from this thesis, we may proceed to develop a higher-dimensional directed type theory. In section 10.1.2, we address our disregard for syntactical aspects of type theory in the rest of the thesis.

10.1.1 Towards Higher Directed Type Theory

As mentioned in the introductory chapter 1, the common motivation for all contributions in this thesis was the preparation of a higher directed type system with interacting modalities for functoriality and naturality. In this section, I will sketch how such a type system is now starting to take shape, without getting too specific for reasons of space and time.

Modes In RelDTT [ND18a], we have abandoned the idea that types and their kinds should be the same thing, and instead used a multimode system [LS16] to embrace the diversity. This still applies – and perhaps in a more familiar way – to a directed system. There, we may still be concerned with types that

are essentially sets, such as \mathbf{Bool} or \mathbb{N} , and it will be pointless to consider the variance of functions to or from such types. On the other hand, a universe of such sets is of course a category, or indeed a pro-arrow equipped category [nLa20a]. A universe of (pro-arrow equipped) categories, is then a 2-category, or indeed a suitable 2-dimensional generalization of a pro-arrow equipped category. On a 2-category \mathcal{C} , we can reverse 1-arrows (yielding \mathcal{C}^{op}) or 2-arrows (yielding \mathcal{C}^{co}), so that we clearly need more modalities for variance of functors between 2-categories than between 1-categories. A multimode type system instantiating MTT is an excellent answer to this phenomenon.

Pro-arrows In ParamDTT [NVD17a], we considered bridges and paths, and a bridge between types holds a notion of heterogeneous paths between their elements. This was generalized in RelDTT [ND18a], where we considered i -edges, and an $(i + 1)$ -edge between types holds a notion of i -edges between their elements. In category theory, a profunctor between categories can be regarded as holding a notion of heterogeneous morphisms between their objects. These profunctors are abstracted as pro-arrows in the pro-arrow equipped category of categories \mathbf{Cat} . I believe that a directification of Degrees of Relatedness can lead to an interesting theory of higher-dimensional pro-arrow equipments (which to my knowledge is presently non-existent), which seems an excellent setting for combining parametricity, naturality and functoriality in a single system. Interestingly, the existence of companions and conjoinants and their associated squares could be asserted by the presheaf structure via negation-like and connection-like operations.

Hom-abstraction In cubical homotopy type theory [e.g. Coh+17], the path type $\text{Path}_A a b$ is typically implemented or at least modelled via an extension type:

$$\text{Path}_A a b := (i : \mathbb{I}) \rightarrow A \text{ext}\{i \doteq_{\mathbb{I}} 0 ? a \mid i \doteq_{\mathbb{I}} 1 ? b\},$$

so that a path is a function from the interval that equals a at the source and b at the target. In directed type theory, this is difficult, because the Hom -type is contravariant in the source and covariant in the target, so that application to 0 of functions over the interval would have to have different variance than application to 1. Strikingly, the multiplier given by Pinyo and Kraus’s twisted prism functor [PK19] (example 7.4.11) gives us exactly that: it comes with natural transformations $(\text{id}_W, 0) : W^{\text{op}} \rightarrow W \times \mathbb{I}$ and $(\text{id}_W, 1) : W \rightarrow W \times \mathbb{I}$. Specifically, it looks like we can implement $\text{Hom}_A a b$ as the subtype of elements of $\langle \forall (i : \mathbb{I}) \mid \langle \Omega i \mid A \rangle \rangle$ that match the endpoints. Since the twisted prism functor is *not* semicartesian, the weakening modality Ωi can only be used in the empty shape context, as weakening from the empty context is always possible. Another thing to take into account is that an arrow in the above modal type is a

twisted square in A , which should not be a heterogeneous *morphism* between elements of Hom_A , but instead a heterogeneous *equality*, so we need to throw in a parametric/natural modality and take a subtype of $\langle \text{par} \mid \langle \forall(i : \mathbb{I}) \mid \langle \Omega i \mid A \rangle \rangle \rangle$.

Internal naturality Generalizing internal parametricity operators, we would like internal operators that allow us to inhabit naturality squares simply from the knowledge that a function type-checks as natural. From chapter 7, we know that the transpension type and the strictness axiom together give us all the wealth of currently known presheaf operators, which is encouraging.

Fibrancy We are interested in covariant types, which have a mapping like functors do, and Segal types, in which morphisms can be composed. We hope that Segal fibrancy can be defined as covariance of the Hom -type. As these notions of fibrancy are not closed under all the type formers that one would wish, it is recommendable to have a type system that also includes non-fibrant types, and to deal with fibrancy maximally internally. Neither notion of fibrancy is robust, but our findings in Chapter 8 tell us that there is hope for an internal treatment if we consider these notions *contextually*.

10.1.2 Good Syntax

Most contributions in this thesis were primarily focused on creating interesting extensions of DTT that are *sound*. The only place where we prove a syntactic well-behavedness result is in section 5.3: MTT satisfies a canonicity result¹ [Gra+20b]. Here, we highlight where such results are missing and discuss how we could proceed.

Degrees of Relatedness In chapter 9, we explained how ParamDTT [NVD17a] and RelDTT [ND18a] are almost instances of MTT, for which we have a canonicity result. Hence, the primary threat to canonicity is in the extensions. The cubical *Glue* type is known to be unproblematic [Hub19] and we expect similar for *Weld*. The modal Σ -types behave essentially like the weak DRAs in MTT, and are therefore also expected to be unproblematic. Then the main problems are function extensionality and the degeneracy axiom that internalizes the identity extension lemma. Function extensionality can perhaps be handled in the style of Sterling et al.’s XTT [SAG19]. Meanwhile, the degeneracy axiom can likely be made computable in a similar fashion that Kan composition is computable in cubical HoTT [Coh+17; Hub19]: it is ‘implementable’ by

¹by the co-authors.

induction on the type, where ‘implementable’ is between quotes because there is no such thing as internal type induction.

Transpension The type system of chapter 7 is not ready for practical use. In section 7.9 we listed the main challenges that we are still facing.

10.2 Implications in Practice

In this section, we try to answer the question: What are the implications of this thesis for the development of programming languages in the (far) future? Let us (pretend to) take for granted that the increasing importance of software verification will naturally lead to the adoption dependently typed languages as combined programming languages and proof assistants.

Multimode languages A type system like RelDTT [ND18a], with infinitely many modes and modalities, is at first sight likely unappetizing to the practical programmer, even when they are familiar with DTT. However, note how a few modes already appear in a language like Haskell, with programs living at mode 0, types at mode 1 and kinds at mode 2. Adding a mode -1 for proofs does not seem outrageous. As shown in the original paper [ND18a, fig. 2], *all* modalities up to mode 1 can be expressed in terms of modalities that were known prior to RelDTT and/or structurality, the novel modality by which algebras depend on their structure [ND18a].

Haskell unfortunately has completely different languages at mode 0 and 1. A general theory of relational modalities may advise more consistent development in the future. And while full RelDTT provides ω modes, most non-logicians will only need the lowest few which are in fact relatively familiar, and need not be hampered by the existence of others.

An understandable concern is that programming becomes ‘very complicated’ if we have to constantly think about which modality we should annotate our functions with. I suggest to look at this from a different angle. *Supposing* one needs to rely on a ‘free’ parametricity theorem, would programmers rather go back to all code they have been relying on and recursively prove that it satisfies its parametricity predicate², or would they rather put a few modalities here and there to point out to the type-checker that, by non-violation, their program is

²This can of course be automatized [KL12; Kel+; BJP12], but drawbacks remain, such as the impossibility to obtain a parametricity proof for a client-provided function.

parametric? Those who have no need for parametricity theorems, are welcome to annotate all functions as ad hoc polymorphic.

Transpension Work like XTT [SAG19] shows how ideas from cubical type theory involving shape variable abstraction are useful even for users who are not interested in parametricity or HoTT, but maybe are in computational function extensionality. With a transpension type, arcane though it may be, these users can use higher-dimensional pattern matching to find that equal terms in a coproduct type come from the same side. For those interested in internal parametricity, a transpension type can be used in the background to implement Moulin et al.’s parametricity operators [BCM15; Mou16].

Internal fibrancy As mentioned in section 10.1.2, internal fibrancy is useful when trying to give certain axioms – such as Kan composition [Coh+17], the degeneracy axiom (identity extension) [ND18a] and clock-irrelevance [BM18] – computational behaviour. Here, too, the transpension type can occasionally be helpful as argued in section 8.7.

Directed type theory Directed type theory, to which everything in this thesis is relevant as discussed in section 10.1.1, in turn could have an impact on everyday program verification by providing smoothly interacting functoriality-for-free and naturality-by-construction with free naturality theorems.

10.3 Is Information Presheavoidal?

The fact that presheaf models have been successful in modelling so many interesting properties of type theory, may prompt the question whether information is fundamentally presheavoidal and if so, why the base category keeps changing with the application and whether there is something like an ‘ultimate base category’.

I would argue that we have to regard presheaves the same way we should regard complex numbers: they are a useful reasoning tool, but never themselves the object of practical interest. Ultimately, we are interested in the *output* of a computer program for a specific given input, and this output will not be cell of a presheaf but an element of the set of possible outputs, living at RelDTT mode 0. Even when the program takes inputs and we can start drawing commutative diagrams, each side of a diagram is a concrete program of the same type. In the case of RelDTT, it is worth mentioning that closed 0-discrete types of

mode 0 really are modelled by sets. In guarded type theory, if we consider not clock-irrelevance but \ominus_k -irrelevance as in the semantics by Bizjak and Møgelberg [BM18], then closed types are again just sets. In the approach to guarded type theory in section 5.3.6.a, we only briefly visited the topos of trees to construct objects modelled in the category of sets. For HoTT, of course, the situation is different, but HoTT is programming language theory applied to abstract homotopy theory; if it were the other way around, then we would use a multimode theory with a mode n for n -groupoids.

In summary, in each of the practical applications, we may take journeys through the world of presheaves in the middle of an argument, but the start and end of our journey should always be in the world of sets.

List of Publications

Papers

[ND20] A. Nuyts and D. Devriese. “Transpension: The Right Adjoint to the Pi-Type”. Pre-print. 2020. URL: <https://anuyts.github.io/files/transpension-paper.pdf>

[Gra+20b] D. Gratzer, G. A. Kavvos, A. Nuyts, and L. Birkedal. “Multimodal Dependent Type Theory”. In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by H. Hermanns, L. Zhang, N. Kobayashi, and D. Miller. ACM, 2020, pp. 492–506. DOI: 10.1145/3373718.3394736. URL: <https://doi.org/10.1145/3373718.3394736>

[Tsa+20] S. Tsampas, A. Nuyts, D. Devriese, and F. Piessens. “A categorical approach to secure compilation”. In: *Coalgebraic Methods in Computer Science (CMCS)*. 2020

[JND19] K. Jacobs, A. Nuyts, and D. Devriese. “How to do proofs: practically proving properties about effectful programs’ results (functional pearl)”. In: *Proceedings of TyDe@ICFP 2019, Berlin, Germany, August 18, 2019*. Ed. by D. Darais and J. Gibbons. ACM, 2019, pp. 1–13. DOI: 10.1145/3331554.3342603. URL: <https://doi.org/10.1145/3331554.3342603>

[Tsa+19] S. Tsampas, A. Nuyts, D. Devriese, and F. Piessens. “Categorical Contextual Reasoning”. In: *Applied Category Theory (ACT)*. 2019

[ND18a] A. Nuyts and D. Devriese. “Degrees of Relatedness: A Unified Framework for Parametricity, Irrelevance, Ad Hoc Polymorphism, Intersections, Unions and Algebra in Dependent Type Theory”. In: *Logic in Computer Science (LICS) 2018, Oxford, UK, July 09-12, 2018*. 2018, pp. 779–788. DOI: 10.1145/3209108.3209119. URL: <https://doi.org/10.1145/3209108.3209119>

[NVD17a] A. Nuyts, A. Vezzosi, and D. Devriese. “Parametric quantifiers for dependent type theory”. In: *PACMPL* 1.ICFP (2017), 32:1–32:29. DOI: 10.1145/3110276. URL: <http://doi.acm.org/10.1145/3110276>

Technical reports and master thesis

- [Nuy20] A. Nuyts. “The Transpension Type: Technical Report”. Pre-print. 2020. URL: <https://anuyts.github.io/files/transpension-techreport.pdf>
- [Gra+20a] D. Gratzer, A. Kavvos, A. Nuyts, and L. Birkedal. “Type Theory à la Mode”. Pre-print. 2020. URL: <https://anuyts.github.io/files/mtt-techreport.pdf>
- [Nuy18a] A. Nuyts. “Presheaf Models of Relational Modalities in Dependent Type Theory”. In: *CoRR* abs/1805.08684 (2018). arXiv: 1805.08684
- [Nuy17] A. Nuyts. *A Model of Parametric Dependent Type Theory in Bridge/Path Cubical Sets*. Tech. rep. Subsumed in [Nuy18a]. Belgium: KU Leuven, 2017. URL: <https://arxiv.org/abs/1706.04383>
- [Nuy15] A. Nuyts. “Towards a Directed Homotopy Type Theory based on 4 Kinds of Variance”. Master’s thesis. Belgium: KU Leuven, 2015. URL: <https://anuyts.github.io/files/mathesis.pdf>

Abstracts

- [CND19] J. Ceulemans, A. Nuyts, and D. Devriese. “Reasoning about Effect Parametricity Using Dependent Types”. In: *Type-Driven Development (TyDe)*. 2019. URL: <http://tydeworkshop.org/2019-abstracts/paper11.pdf>
- [ND19a] A. Nuyts and D. Devriese. “Dependable Atomicity in Type Theory”. In: *TYPES*. 2019
- [ND19b] A. Nuyts and D. Devriese. “Menkar: Towards a Multimode Presheaf Proof Assistant”. In: *TYPES*. 2019
- [Nuy18b] A. Nuyts. “Robust Notions of Contextual Fibrancy”. In: *Workshop on Homotopy Type Theory / Univalent Foundations*. 2018. URL: https://hott-uf.github.io/2018/abstracts/HoTTUF18_paper_2.pdf
- [ND18b] A. Nuyts and D. Devriese. “Internalizing Presheaf Semantics: Charting the Design Space”. In: *Workshop on Homotopy Type Theory / Univalent Foundations*. 2018. URL: https://hott-uf.github.io/2018/abstracts/HoTTUF18_paper_1.pdf
- [NVD17b] A. Nuyts, A. Vezzosi, and D. Devriese. “Parametric quantifiers for dependent types”. In: *TYPES*. 2017. URL: <https://lirias.kuleuven.be/1656707>

Projects

- [Nuy19] A. Nuyts. *Menkar*. <https://github.com/anuyts/menkar>. 2019

List of Figures

3.1	DTT: Contexts and substitutions	79
3.2	DTT: Structural rules	80
3.3	DTT: Π -types	87
3.4	DTT: Σ -types	88
3.5	DTT: Unit type	89
3.6	DTT: Coproduct type	89
3.7	DTT: Empty type	90
3.8	DTT: Identity type	91
3.9	DTT: Natural numbers (elimination rule only)	94
3.10	DTT: Universes	95
5.1	Internalizing a functor or CwF morphism	114
5.2	Internalizing a natural transformation	116
5.3	Internalizing an adjunction	120
5.4	Dependent right adjoints (DRAs)	123
5.5	MTT: Structural rules	135
5.6	MTT: Modal types (weak DRAs)	136
5.7	MTT: Π -types	138

6.1	Propositions: Structural rules	159
6.2	Propositions: Universe	160
6.3	Propositions: Logical operators and proof elimination	162
6.4	Propositions: Equality	164
6.5	Propositions: Proof types	167
6.6	Propositions: Extension types	168
6.7	Presheaves: Strictness type	170
6.8	Presheaves: Glue type	171
6.9	Presheaves: Pushout type	172
6.10	Presheaves: Weld type	174
6.11	Presheaf MTT	177
7.1	Transpension: Naïve typing rules	189
7.2	Transpension: Examples of multipliers	194
7.3	Transpension: Informal notation: Locks and modal types	199
7.4	Transpension: Informal notation: Projections	201
7.5	Transpension: Informal notation: 2-cells	202
7.6	Transpension: Pattern matching	205
7.7	Known operators: The Φ -rule	207
7.8	Known operators: The Ψ -type	209
8.1	Strictly stable NWFS: Fibrant replacement	238
8.2	Strictly stable damped NWFS: Fibrant replacement	239
8.3	Strictly stable NWFS: Left coreplacement	241
8.4	Strictly stable damped NWFS: Left coreplacement	242
8.5	Semantics of LC:EXTEND:MULT	253
9.1	ParamDTT: Action of modalities	266

9.2	ParamDTT: Action of additional modalities in Agda-parametric	268
9.3	RelDTT: Action of parametricity and structurality	276

List of Mathematical Symbols

For abbreviations in text, we refer to the index.

\sqsubset A gap, e.g. $f(\sqsubset)$ means f .

Delimiters

$\langle \dots \rangle$ Angular brackets can mean:

$t\langle\varphi\rangle$ Restriction of a type cell $W \triangleright t : T$ by a morphism $\varphi : V \rightarrow W$ in the base category (section 4.1.2).

$i\langle\varphi\rangle$ Contravariant morphism application in a base category such as Cube or Clock.

$\langle R \mid T \rangle$ Application of the DRA R to the type T (definition 5.2.1, DRA). If R is actually a right adjoint CwF morphism, then this means $(RT)[\eta]$ (lemma 5.2.3).

$\langle \mu \mid^m T \rangle$ Weak DRA (WDRA), which binds a tick m .

$[\dots]$ Brackets can mean:

$T[\sigma]$ Type substitution (see (\dots) for the substitution itself) (TY:SUB).

$t[\sigma]$ Term substitution (see (\dots) for the substitution itself) (TM:SUB).

$[f_1, \dots, f_n]$ Morphism out of a coproduct or colimit.

$[\varphi]$ Proof type (PROOF).

$[n]$ n -dimensional simplex object in Simplex.

$[\dots]$ A ket can mean:

$t[\gamma]$ Action of the semantic term $\Gamma \vdash t : T$ on a cell $\gamma : W \Rightarrow \Gamma$ (section 4.1.2).

$T[\gamma]$ See \triangleright at Judgements/Turnstiles.

$\llbracket \dots \rrbracket$ Interpretation in a model.

$\{ \dots \}$ Curly braces can mean:

$\{a_1, \dots, a_n\}$ Set containing a_1, \dots, a_n .

$\{x \mid P(x)\}$ Set containing all x satisfying $P(x)$.

$\{\varphi ? t_1 \mid \chi ? t_2\}$ Eliminator of a proposition (FALSEHOOD:ELIM, OR:ELIM, EQ:J, notation 6.1.5).

(\dots) Parentheses can mean:

(\dots) Overrides precedence rules.

$f(x)$ Sometimes used for function application, especially in set theory, although we also use juxtaposition (see there).

(f_1, \dots, f_n) Morphism to a product or limit.

(a, b) Pair constructor of the Σ -type.

(a_1, \dots, a_n) Element of a product or limit in **Set**, i.e. a tuple.

$()$ can mean:

$()$ Empty context.

$()$ Substitution to the empty context.

$()_F$ Substitution to the image $F()$ of the empty context under a weak CwF morphism F .

(φ, χ) Morphism in the (twisted) arrow category.

- (σ, t) Substitution to an extended context (in variable-free notation) (CTX-EXT:INTRO).
- $(\sigma, t/x)$ Idem (in variable notation).
- $(\sigma, t)_F$ Substitution to the image $F(\Gamma.T)$ of an extended context (in variable-free notation) under a weak CwF morphism F (IMG:CTX-EXT:INTRO).
- $(\sigma, t/Fx)_F$ Idem (in variable notation).
- (σ, ok) Substitution to a restricted context (CTX-RESTR:INTRO).
- $(\sigma, \alpha \downarrow_n^m)$ Functorial action of $\mathbf{!}$, applied to $\sigma : \Gamma \rightarrow \Gamma'$ (covariantly) and $\alpha : \mu \Rightarrow \nu$ (contravariantly), see LOCK:FMAP.
- (σ, \downarrow_n^m) Short for $(\text{id}, \downarrow_n^m)$.
- $(\in \Gamma)$ See ‘element’.
- (ℓ) ℓ -edge interval in DCube_d .
- $\lceil \dots \rceil$ Encode; turn a type/proposition into an element of a universe.
- $\lceil \dots \rceil$ Identity function $\text{Ty}_\ell(\mathbf{y}W) \rightarrow (W \triangleright \mathbf{U}_\ell)$.

Arrows

\rightarrow can mean:

- $A \rightarrow B$ Set of functions from A to B .
- $A \rightarrow B$ Type of non-dependent functions (of identity modality) from A to B .
- $(x : A) \rightarrow B$ Type of dependent functions (of identity modality).
- $a \rightarrow b$ Set of morphisms from a to b (definition 2.2.1).
- $F \rightarrow G$ Set of natural transformations from F to G (definition 2.2.6).
- $(x \xrightarrow{\varphi} y)$ Object of the arrow category \mathcal{C}^\uparrow .
- $(x \xrightarrow{\varphi} y)$ Object of the twisted arrow category $\text{Tw}(\mathcal{C})$ (definition 2.2.16).
- $\sigma : \Gamma \rightarrow \Delta$ Substitution judgement.
- $\sigma : \Sigma \rightarrow \Sigma'$ Substitution metajudgement (definition 3.1.8).
- \uparrow The walking arrow.

- \mathcal{C}^\uparrow Arrow category of \mathcal{C} .
- $\uparrow \searrow$ The walking damped arrow.
- $\mathcal{C}^{\uparrow \searrow}$ Damped arrow category of \mathcal{C} .
- x^\uparrow Morphism of $\vec{x} = (x_0 \xrightarrow{x^\uparrow} x_1) \in \mathcal{C}^\uparrow$
or $\vec{x} = (x_0 \xrightarrow{x^\uparrow} x_1 \xrightarrow{x^\searrow} x_2) \in \mathcal{C}^{\uparrow \searrow}$.
- x^\searrow Damping of $\vec{x} = (x_0 \xrightarrow{x^\uparrow} x_1 \xrightarrow{x^\searrow} x_2) \in \mathcal{C}^{\uparrow \searrow}$.
- \vec{x} See ‘vec’.
- $x \alpha \downarrow_m$ The variable x whose usage under $\mathbf{!}_\mu^m$ is justified by 2-cell α (CTX-MODEXT:VAR).
- $x \downarrow_m$ Short for $x \downarrow_m$.
- $\alpha \downarrow_n^m$ Short for $(\text{id}, \alpha \downarrow_n^m)$, see (...) at Delimiters.
- \downarrow_n^m Short for $(\text{id}, \downarrow_n^m)$, see (...) at Delimiters.
- \Rightarrow can mean:
- $P \Rightarrow Q$ P implies Q .
- $W \Rightarrow \Gamma$ Application of presheaf Γ to base category object W (notation 2.3.2).
- $|_i^j$ The unique morphism $i \rightarrow j$ in a poset used as a category.
- $x \mapsto a$ x maps to a .
- \uparrow See \mathfrak{U} (‘uplus’).

Judgements/Turnstiles

- \dashv See ‘dashv’.
- \perp See ‘bot’.
- \top See ‘top’.
- \Vdash Metajudgement, i.e. judgement in a GAT (definition 3.1.8).
- $\Sigma \Vdash S$ sort Expression S is a sort in metacontext Σ .
- $\Sigma \Vdash t : S$ Expression t is a term of sort S in metacontext Σ .
- \vdash Judgement:
- $J \text{jud}$ J is a judgement form (section 3.1.5).
- Γctx Γ is a context (JUD:CTX).
- $\Gamma \vdash T$ type T is a type in context Γ (JUD:TY).
- $\Gamma \vdash T$ type $_\ell$ T is a type of size ℓ in context Γ .

- $\Gamma \vdash t : T$ t is a term of type T in context Γ (JUD:TM).
 $\Gamma \vdash T$ The set $\{t \mid \Gamma \vdash t : T\}$.
 $\Gamma \vdash \varphi$ **prop** φ is a proposition in context Γ (JUD:PROP).
 $\Gamma \vdash \varphi$ Proposition φ holds in context Γ (JUD:ASSERT).
- \triangleright can mean:
 $W \triangleright T[\gamma]$ Set of presheaf cells over γ of the semantic type T (section 4.1.2).
 $W \triangleright t : T[\gamma]$ means $t \in (W \triangleright T[\gamma])$.
- ## Greek Letters
- The alphabet: $\alpha\beta\gamma\delta\varepsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi\omicron\pi\rho\sigma\tau\nu\varphi\chi\psi\omega$.
- β can mean:
 β Some morphism or natural transformation, defined locally.
 β -rule See the index.
- δ can mean:
 δ Comonadic duplication (definition 2.2.46).
 δ Some morphism or natural transformation, defined locally.
 $\vec{\delta}^{\mathbf{L}}$ Comonadic duplication of \mathbf{L} .
 $\delta^{\mathbf{L}}$ Codomain of $\vec{\delta}^{\mathbf{L}} = (\text{id}, \delta^{\mathbf{L}})$
- Δ can mean:
 Δ Some presheaf or context, defined locally.
 Δ_n Synonym for $\mathbf{y}[n]$, the n -dimensional simplex.
- ε can mean:
 ε Unit of an adjunction (definition 2.2.41).
 ε Unit of a monad (definition 2.2.46).
 ε Some morphism or natural transformation, defined locally.
 $\vec{\varepsilon}^{\mathbf{L}}$ Co-unit of $U_{\mathbf{L}} \dashv C_{\mathbf{L}}$; hence of \mathbf{L} .
 $\varepsilon^{\mathbf{R}}$ Co-unit of $F_{\mathbf{R}} \dashv U_{\mathbf{R}}$.
 $\varepsilon^{\mathbf{R}}$ Domain of $U_{\mathbf{R}} \varepsilon^{\mathbf{R}} = (\varepsilon^{\mathbf{R}}, \text{id})$.
- η can mean:
 η Co-unit of an adjunction (definition 2.2.41).
 η Co-unit of a comonad (definition 2.2.46).
- η Some morphism or natural transformation, defined locally.
 η -rule See the index.
 $\eta^{\mathbf{L}}$ Unit of $U_{\mathbf{L}} \dashv C_{\mathbf{L}}$.
 $\eta^{\mathbf{L}}$ Codomain of $U_{\mathbf{L}} \eta^{\mathbf{L}} = (\text{id}, \eta^{\mathbf{L}})$.
 $\eta^{\mathbf{R}}$ Unit of $F_{\mathbf{R}} \dashv U_{\mathbf{R}}$; hence of \mathbf{R} .
- λ can mean:
 $\lambda(x : A).b$ λ -abstraction.
 $\lambda x.b$ λ -abstraction (with domain annotation suppressed).
 λb λ -abstraction in variable-free notation.
 $\lambda u.a$ Informal notation for $\text{mod}_{\forall u} a$ or $\text{mod}_{\Pi u} a$ (section 7.5).
- Λ can mean:
 Λ Some presheaf or context, defined locally.
 Λ_n A simplicial set consisting of n composable arrows; the subpresheaf of $\mathbf{y}[n]$ containing just the Hamiltonian path.
 $\Lambda X.t$ Type abstraction in System F or F ω .
- μ can mean:
 μ Monadic multiplication (definition 2.2.46).
 μ Some morphism or natural transformation, defined locally.
 $\vec{\mu}^{\mathbf{R}}$ Comonadic duplication of \mathbf{R} .
 $\mu^{\mathbf{R}}$ Codomain of $\vec{\mu}^{\mathbf{R}} = (\text{id}, \mu^{\mathbf{R}})$
- ξ Last variable (in variable-free notation).
- π can mean:
 π Weakening by the last variable (in variable-free notation) (CTX-EXT:WKN).
 π_x Weakening by variable x (in variable notation) (CTX-EXT:WKN).
 π Weakening by the last proposition (CTX-RESTR:WKN).
 π_i The i th projection from a cartesian product.
- Π can mean:
 ΠAB Π -type in variable-free notation.
 $\Pi(u : \mathbb{U}, i = 0)$ Π -modality, right adjoint to $\Omega(u : \mathbb{U}, i = 0)$ (section 7.4.4).
 $\Pi\sigma$ Π -modality, right adjoint to $\Omega\sigma$ (section 7.4.4).
 Πu Shorthand for $\Pi(u : \mathbb{U})$.

- $\Pi(u : \mathbb{U}, i = 0).A$ Informal notation for $\langle \Pi(u : \mathbb{U}, i = 0) \mid A \rangle$ (section 7.5).
- \prod Cartesian product.
- \coprod Coproduct.
- Σ can mean:
- ΣAB Σ -type in variable-free notation.
 - Σ_U Forgetful functor from the slice category \mathcal{W}/U (definition 2.2.36).
 - Σ_Γ Forgetful functor from the category of elements \mathcal{W}/Γ (definition 2.2.37).
 - Σ/v Functorial action of the slice category (definition 2.2.36).
 - Σ/σ Functorial action of the category of elements (definition 2.2.37).
 - Σu Shorthand for $\Sigma(u : \mathbb{U})$.
 - $\Sigma(u : \mathbb{U}, i = 0).\Gamma$ Informal notation for $\Gamma, \mathbf{\Delta}_{\Omega(u:\mathbb{U},i=0)}$ (section 7.5).
- τ can mean:
- τ Representable morphism $\tau : \widetilde{\text{Ty}} \rightarrow \text{Ty}$ of a natural model.
 - τ Some substitution.
- Φ The Φ -rule (PHI) [BCM15; Mou16], a.k.a. extent [CH20].
- Ψ The Ψ -type (PSI) [BCM15; Mou16], a.k.a. Gel [CH20].
- Ω can mean:
- $\Omega(u : \mathbb{U}, i = 0)$ Weakening/substitution modality, in this case it weakens over u while substituting $0/i$ (section 7.4.4).
 - $\Omega\sigma$ Weakening/substitution modality (section 7.4.4).
 - Ωu Shorthand for $\Omega(u : \mathbb{U})$.
 - $\Omega(u : \mathbb{U}, i = 0).A$ Informal notation for $\langle \Omega(u : \mathbb{U}, i = 0) \mid A \rangle$ (section 7.5).
 - $\Omega(u : \mathbb{U}, i = 0).\Gamma$ Informal notation for $\Gamma, \mathbf{\Delta}_{\Pi(u:\mathbb{U},i=0)}$ (section 7.5).

Numbers

- 0 Zero constructor of Nat .
- 0 Domain object of the walking (damped) arrow.

- x_0 Domain of $\vec{x} = (x_0 \xrightarrow{x\uparrow} x_1) \in \mathcal{C}^\uparrow$ or $\vec{x} = (x_0 \xrightarrow{x\uparrow} x_1 \xrightarrow{x\downarrow} x_2) \in \mathcal{C}^{\uparrow\downarrow}$.
- 1 Codomain object of the walking (damped) arrow.
- 1 Identity modality or 2-cell.
- x_1 Domain of $\vec{x} = (x_0 \xrightarrow{x\uparrow} x_1) \in \mathcal{C}^\uparrow$ or $\vec{x} = (x_0 \xrightarrow{x\uparrow} x_1 \xrightarrow{x\downarrow} x_2) \in \mathcal{C}^{\uparrow\downarrow}$.
- 2 Damping object of the walking damped arrow.
- x_2 Damping object of $\vec{x} = (x_0 \xrightarrow{x\uparrow} x_1 \xrightarrow{x\downarrow} x_2) \in \mathcal{C}^{\uparrow\downarrow}$.

Alphabetically

We order other symbols alphabetically by plausible pronunciations and/or by their $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ command.

A

- \forall can mean:
- $\forall x.P(x)$ For all x , $P(x)$ holds.
 - $\forall i.F(i, i)$ End of the functor F (definition 2.2.26).
 - $\forall i.F(i \xrightarrow{\text{id}} i)$ Dependent end of the functor F (definition 2.2.31).
 - $\forall X.A$ Type quantification in System F or $\text{F}\omega$.
 - $\forall(u : \mathbb{U})$ \forall -modality (section 7.4.5).
 - $\forall u$ Shorthand for $\forall(u : \mathbb{U})$.
 - $\forall(u : \mathbb{U}).A$ Informal notation for $\langle \forall(u : \mathbb{U}) \mid A \rangle$ (section 7.5).
 - $\forall(u : \mathbb{U}).\Gamma$ Informal notation for $\Gamma, \mathbf{\Delta}_{\forall(u:\mathbb{U})}$ (section 7.5).
 - $\Gamma, \forall(u : \mathbb{U}).\Delta$ Informal notation for $\Gamma, \mathbf{\Delta}_{\forall(u:\mathbb{U})}, \Delta, \mathbf{\Delta}_{\forall(u:\mathbb{U})}$ when \mathbb{U} is cancellative and affine (section 7.5).

$\acute{\ell}$ can mean:

- $\acute{\ell}$ Object of \mathcal{L} , the category of left maps.

- \acute{g} Shorthand for $I_{\mathcal{G}} \acute{g}$.

$\acute{\lambda}$ Morphism of \mathcal{L} , the category of left maps.

A can mean:

- A** Default symbol for transposition (definition 2.2.41).

- A** The identity function ($LyW \vdash T[\sigma] \rightarrow (W \triangleright (RT)[\mathbf{A}(\sigma)])$) for the adjunction $\mathbf{A} : L \dashv R$.
- A_F** Transposition for $F_! \dashv F^*$.
- A_F** The identity function ($FV \Rightarrow \Gamma \rightarrow (V \Rightarrow F^*\Gamma)$).
- A_F** The identity function ($FV \triangleright T[\gamma] \rightarrow (V \triangleright (F^*T)[\mathbf{A}_F(\gamma)])$).
- \wedge See ‘wedge’.
- app_u** Co-unit of $\exists u \dashv \forall u$ (section 7.4.5) or of $\Omega u \dashv \Pi u$ (section 7.4.4).
- *** The kind of types in System F ω .
- ⊗** Applicative functor application, especially for weak DRAs (section 5.3.3.a).
- @** can mean:
- @ C** This judgement takes place in the CwF \mathcal{C} .
 - @ p** This judgement takes place at mode p .
- B**
- B** can mean:
- B** Transposition for some adjunction.
 - B_F** Transposition for $F^* \dashv F_*$.
 - B_F** The identity function ($F^*\mathbf{y}W \rightarrow \Gamma \rightarrow (W \Rightarrow F_*\Gamma)$).
 - B_F** The identity function ($F^*\mathbf{y}W \vdash T[\sigma] \rightarrow (W \triangleright (F_*T)[\mathbf{B}_F(\sigma)])$).
- ℬ** Bridge interval.
- ∅** See ‘transpension’.
- Bool** Type of booleans.
- ⊥** can mean:
- ⊥** Initial object.
 - ⊥** Logical falsehood.
 - $\lambda \perp \rho$ Orthogonality of morphisms (definition 2.4.1).
- ∂** can mean:
- $\partial\mathbb{U}$ Boundary of a shape (definition 7.4.3).
 - ∂U Boundary of U (definition 7.4.3).
 - $\sqsubset \times \partial U$ Boundary of a multiplier (definition 7.4.3).
 - $u \in \partial\mathbb{U}$ Boundary predicate (BOUNDARY).
- Marks the end of a proof or of a proposition whose proof is left to the reader.
- BPCube** Category of bridge/path cubes. Essentially DCube₁.
- ⌢** See ‘frown’.
- Empty string as a tick for the identity modality.
- C**
- C** can mean:
- C_K** Cofree Eilenberg-Moore coalgebra functor for the comonad K (proposition 2.2.50).
 - C_L** Cofree left map functor in an NWFS.
- case** An eliminator. Look up the pattern that is matched against.
- Cat** Category or 2-category of (small) categories.
- CCHM** Category of CCHM cubes.
- o** can mean:
- $g \circ f$ Composition of functions.
 - $G \circ F$ Composition of functors.
 - $\beta \circ \alpha$ Composition of natural transformations.
 - $\chi \circ \varphi$ Composition of morphisms.
 - $\gamma \circ \varphi$ Restriction of presheaf cell $\gamma : W \Rightarrow \Gamma$ by base morphism $\varphi : V \rightarrow W$ (notation 2.3.2).
 - $\sigma \circ \gamma$ Application of presheaf morphism $\sigma : \Gamma \rightarrow \Delta$ to presheaf cell $\gamma : W \Rightarrow \Gamma$ (notation 2.3.2).
- Clock** Category of clocks.
- ⊙** Clock object, $\ominus = \text{colim}_\ell \mathbf{y}(i : \ominus_\ell)$ (example 2.3.16).
 - ⊙_ℓ** Clock of longevity ℓ .
 - C^{co}** The 2-category \mathcal{C} with 2-cells reversed.
- Cod** Codomain functor from the (twisted) arrow category.
- colim** Colimit (definition 2.2.19).
- $x : A$ Term x has type A . Occasionally also: x is an element of set A , especially for sets of functions, morphisms or presheaf cells.
- $x :: A$ Term x has pseudotype A .
- :=** See ‘equals’.
- ,** can mean:
- $\Gamma, x : T$ Context extension (in variable notation) (CTX-EXT).
 - Γ, φ Context restriction (in variable notation) (CTX-RESTR).

- ... See the surrounding delimiters at Delimiters.
- con** Continuous modality.
- \cong See ‘isomorphic’.
- const_u Unit of $\perp u \dashv \forall u$ (section 7.4.5) or of $\Omega u \dashv \Pi u$ (section 7.4.4).
- $\mathcal{C}^{\text{coop}}$ The 2-category \mathcal{C} with 1- and 2-cells reversed.
- \amalg See Greek letter π .
- cospoil_u Restricts to fresh stuff: $\text{cospoil}_u : \amalg u \Rightarrow \forall u$.
- ctx** See \vdash at Judgements/Turnstiles.
- Cube** can mean:
- ${}^a\text{Cube}$ Category of a -ary cartesian cubes. Default: $a = 2$.
 - Cube_{\square} Category of affine cubes.
 - Cube_{\neg} Category of cubes with an edge involution.
- cut_{φ} Constructor of the extension type (EXTENSION:INTRO).
- D**
- $L \dashv R$ L is left adjoint to R (2.2.41).
- DCube_d Category of depth d cubes.
- \div can mean:
- $a \div_A b$ Type/set of edges from a to b in type A .
 - $a \div_R b$ Type/set of bridges from a to b along edge R .
- $/$ See ‘slash’.
- \check{g} Object of \mathcal{G} , the category of generating left maps.
- Dom** Domain functor from the (twisted) arrow category.
- \cdot can mean:
- $\Gamma.T$ Context extension (in variable-free notation) (CTX-EXT).
 - $\sigma.f$ Functorial action of context extension in variable-free notation, applied to a substitution $\sigma : \Gamma \rightarrow \Gamma'$ and a function $\Gamma \vdash f : T \rightarrow T'[\sigma]$ or similar.
 - $\Gamma.\varphi$ Context restriction (in variable-free notation) (CTX-RESTR).
- $f \cdot^{\text{m}} a$ Modal function application (MODPI:ELIM), which binds a tick m in the argument. Written $f a$ for non-modal functions.
- See ‘bullet’.

E

- \exists can mean:
- $\exists x.P(x)$ There exists some x such that $P(x)$ holds.
 - $\exists i.F(i, i)$ Co-end of the functor F (definition 2.2.26).
 - $\exists i.F(i \xrightarrow{\text{id}} i)$ Dependent co-end of the functor F (definition 2.2.32).
 - $\exists X.A$ Type quantification in System F or $F\omega$.
 - $\forall u$ Shorthand for $\exists(u : \mathbb{U})$.
 - $\exists(u : \mathbb{U}).\Gamma$ Informal notation for $\Gamma, \blacksquare_{\perp(u:\mathbb{U})}$ (section 7.5).
 - $\Gamma, \exists(u : \mathbb{U}).\Delta$ Informal notation for $\Gamma, \blacksquare_{\forall(u:\mathbb{U})}, \Delta, \blacksquare_{\perp(u:\mathbb{U})}$ when \mathbb{U} is cancellative and affine (section 7.5).
- EI** Decode; promote universe element to a type/proposition.
- EI** Identity function ($W \triangleright U_{\ell} \rightarrow \text{Ty}_{\ell}(\mathbf{y}W)$).
- EM** can mean:
- $\text{EM}(M)$ Eilenberg-Moore category for the monad M .
 - $\text{EM}(K)$ Eilenberg-Moore category for the comonad K .
- Empty** Empty type.
- \in can mean:
- $x \in A$ x is an element of set A .
 - $x \in \mathcal{C}$ x is an object of category \mathcal{C} .
 - $(\in \Gamma')$ Proposition (or morphism to **Prop**) corresponding to the subobject $\Gamma' \subseteq \Gamma$ (definition 2.2.58).
 - $(\in \Upsilon)$ Cell of **Prop** (the subobject classifier of a presheaf category) corresponding to the subobject $\Upsilon \subseteq \mathbf{y}W$ (proposition 2.3.25).
- $=$ can mean:
- $a = b$ Judgemental equality.
 - $a = b$ Set theoretic equality.
- $a \dot{=}^{\text{m}} b$ Equality proposition (EQ). The subscript may be omitted.
- $:=$ can mean:
- $\text{not} := \text{def}$ Notation *not* is defined as *def*.

$x := b$ Pattern match on an assertion of the equality proposition (EQ:J, notation 6.1.5).

\cong See ‘isomorphic’.

$a \equiv_A b$ Propositional equality, i.e. identity type (ID). The subscript may be omitted.

\succ can mean:

$a \succ_A b$ Type/set of paths (or proofs of heterogeneous equality) from a to b in type A .

$a \succ_R b$ Type/set of paths (or proofs of heterogeneous equality) from a to b along bridge/relation R .

$A \text{ ext}\{\varphi ? a\}$ Extension type (EXTENSION).

extent See Φ in Greek Letters.

F

F can mean:

F_M Free Eilenberg-Moore algebra functor for the monad M (proposition 2.2.50).

$F_{\mathbf{R}}$ Free right map functor in an NWFS.

\exists can mean:

$\exists(u : \mathbb{U})$ Fresh weakening modality (section 7.4.5).

$\exists u$ Shorthand for $\exists(u : \mathbb{U})$.

$\exists(u : \mathbb{U}).A$ Informal notation for $\langle \exists(u : \mathbb{U}) \mid A \rangle$ (section 7.5).

$\exists(u : \mathbb{U}).\Gamma$ Informal notation for $\Gamma, \mathbf{A}_{\forall(u:\mathbb{U})}$ (section 7.5).

Fac Factorization functor in an FWFS or NWFS.

false Constructor of Bool.

Fib can mean:

$\text{Fib}(T)$ Type of internal fibrancy structures for T over an NWFS.

$\text{Fib}(w.T)$ Type of internal contextual fibrancy structures for T over a damped NWFS.

\forall See ‘A’.

fresh $u a$ Informal notation for $\text{mod}_{\exists u} a$ (section 7.5).

\curvearrowright can mean:

$a \curvearrowright_A b$ Type/set of bridges from a to b in type A .

$S \curvearrowright_{\kappa} T$ Type/set of edges from a to b in kind κ .

$a \curvearrowright_R b$ Type/set of bridges from a to b along bridge/relation R .

fst First projection from Σ -type.

funext Function extensionality axiom.

G

\mathcal{G} Category of generating left maps.

\mathcal{G}_{α} The α th Grothendieck universe, the set of α -sized things (definition 2.1.3).

Gel See Ψ in Greek Letters.

Glue Glue type (GLUE).

glue Constructor of the Glue type (GLUE:INTRO).

\hat{r} Object of \mathcal{R} , the category of right maps.

$\hat{\rho}$ Morphism of \mathcal{R} , the category of right maps.

H

\square See ‘box’.

\uparrow See Arrows.

hoc Ad hoc polymorphism.

$\text{Hom}(x, y)$ Set of morphisms from x to y (definition 2.2.1).

I

\mathbb{I} can mean:

\mathbb{I} Edge object in e.g. \mathbf{RG} .

\mathbb{I} Edge interval in a category of cubes.

\mathbb{I} Interval shape/type in a cubical type theory.

id Identity morphism.

id_x Identity morphism at x .

Id Identity functor.

$I_{\mathcal{G}}$ Inclusion $I_{\mathcal{G}} : \mathcal{G} \rightarrow \mathcal{L}$ of generating left maps in left maps.

\in See ‘element’.

inl Left constructor of the coproduct type.

inr Right constructor of the coproduct type.

$\int_{c \in C} F(c \xrightarrow{\text{id}_c} c)$ Category of elements of F (definition 2.2.37).

irr Irrelevance modality.

$A \cong B$ A is isomorphic to B . As a type: the type of isomorphisms.

J

jud See \vdash at Judgements/Turnstiles.

juxtaposition can mean:

$f x$ Function application.

$f u$ Informal notation for $\text{prmod}_{\forall u} \cdot f$ or $\text{prmod}_{\Pi u} \cdot f$ (section 7.5).

$F x$ Application of functor F to object x .

$F\varphi$ Application of functor F to morphism φ .

$F\alpha$ Whiskering of functor F and natural transformation α (essentially application).

αF Whiskering of natural transformation α and functor F .

nm Concatenated string as a tick for a composite modality.

... Anything also denoted by \circ , see ‘circ’.

L

\mathcal{L} Category or class of left maps.

L Left coreplacement functor/comonad.

ℓ Natural transformation $\text{Dom} \rightarrow \text{Fac}$.

let An eliminator. Look up the pattern that is matched against.

let_{ν}^n A modal eliminator. Look up the pattern that is matched against.

\pitchfork See ‘pitchfork’.

lim Limit (definition 2.2.19).

\mathbf{L}_{μ}^m Lock (LOCK), conceptually the left adjoint to μ , which binds a tick m .

$\text{locks}(\Theta)$ Composite of all the modalities on locks in telescope Θ (fig. 5.5).

\times See ‘times’.

M

merid $u a$ Informal notation for $\text{mod}_{\forall u} a$ (section 7.5).

mill A primitive we once proposed that swaps $\forall(u : \mathbb{U})$ and Weld [ND18b].

$\text{mod}_{\mu}^m x$ Constructor of the weak DRA (WDRA:INTRO), which binds a tick m .

mor The natural transformation $\text{mor} : \text{Dom} \rightarrow \text{Cod}$.

N

\mathbb{N} Set of natural numbers, including 0.

\mathbb{N} Node object e.g. **RG**.

Nat Type of natural numbers.

O

0 See ‘0’ at Numbers.

\otimes See ‘asterisk’.

$\text{Obj}(\mathcal{C})$ Set of objects of category \mathcal{C} (definition 2.2.1).

(σ, ok) See (\dots) at Delimiters.

1 See ‘1’ at Numbers.

\mathcal{C}^{op} Opposite category of \mathcal{C} .

F^{op} Opposite functor of F .

\vee See ‘vee’.

\perp See ‘bot’.

P

\mathbb{P} Path interval for bridge/path cubes.

$\text{paste}\{\varphi ? a | e\}$ Eliminator of the extension type (EXTENSION:ELIM).

\asymp See ‘equals’.

par Parametric modality.

∂ See ‘boundary’.

\perp See ‘bot’.

\vdash can mean:

$\mu \vdash x :^m A$ Modal variable declaration (CTX-MODEXT, MODPI).

$\mu \vdash x$ Idem, omitting the type

$\mu \vdash^m A$ Similar, but non-dependent.

$|$ can mean:

$\Xi | \Gamma$ Separates the shape context (technically the MTT mode) Ξ from the context Γ (section 7.4.1).

$\gamma|_{u=v}$ Reindexing (sections 7.2 and 7.5).

$\{x | P(x)\}$ See $\{\dots\}$ at Delimiters.

$\{\varphi ? t_1 | \chi ? t_2\}$ See $\{\dots\}$ at Delimiters.

$\langle R | A \rangle$ See $\langle \dots \rangle$ at Delimiters.

$\vec{\ell} \pitchfork \vec{r}$ can mean:

- $\vec{\ell} \pitchfork \vec{r}$ and \vec{r} satisfy the lifting property.
 $\vec{\ell} \pitchfork \vec{r}$ Set of lifting operators of $\vec{\ell}$ against \vec{r} .
- $\sigma+$ The substitution $(\sigma \circ \pi, \xi) : \Delta.T[\sigma] \rightarrow \Gamma.T$, where $\sigma : \Delta \rightarrow \Gamma$.
- prmod_μ Projection from $\langle \mu \mid A \rangle$ (proposition 7.3.3).
- \prod See Greek letter π .
- Prop** Subobject classifier; universe of propositions.
- prop** See Judgements/Turnstiles.
- Psh**(\mathcal{W}) Category of presheaves over \mathcal{W} (definition 2.3.1).
- ptw** Pointwise modality.
- Q**
- $\varphi ? t$ Proposition equivalent of $(p : [\varphi]) \mapsto t$.
- R**
- \mathcal{R} Category or class of right maps.
- R** can mean:
- R** Right replacement functor/monad of a (damped) NWFS.
 - R** T Internal fibrant replacement over an NWFS (fig. 8.1).
 - R** f Functorial action of the above (fig. 8.1).
 - R'** f Dependent functorial action of the above (fig. 8.3).
 - R**($w.T$) d Internal contextual fibrant replacement over a damped NWFS (fig. 8.2).
 - R** $e f$ Functorial action of the above (fig. 8.2).
 - R'** $e f$ Dependent functorial action of the above (fig. 8.4).
- r** Natural transformation $\mathbf{Fac} \rightarrow \mathbf{Cod}$.
- r** Reflexivity map in e.g. \mathbf{RG} .
- refl_a Reflexivity constructor of the identity type. The subscript may be omitted.
- reid_{x_u} Unit of $\forall u \neg \exists u$ (section 7.4.5).
- \uparrow See Arrows.
- ${}^a\mathbf{RG}$ Base category of a -ary reflexive graphs. Default: $a = 2$.
- \mathbf{RG}_- Base category of undirected (binary) reflexive graphs.
- S**
- s** Source map in e.g. \mathbf{RG} .
- Set** Category of (small) sets.
- shi** Shape-irrelevance modality.
- Simplex** Category of simplices.
- $/$ can mean:
- U/\mathcal{W} Coslice category (definition 2.2.36).
 - \mathcal{W}/U Slice category (definition 2.2.36).
 - \mathcal{C}/Γ Category of elements of the presheaf Γ (definition 2.2.37).
 - a/x Shorthand for $(\text{id}, a/x)$, see (...) at Delimiters.
 - $(\sigma, a/x)$ See (...) at Delimiters.
 - i/\circlearrowleft Emphasizes that a weakening is happening.
- snd** Second projection from Σ -type.
- \sqsubset A gap, e.g. $f(\sqsubset)$ means f .
- spoil_u Forgets freshness: $\text{spoil}_u : \exists u \Rightarrow \Omega u$.
- \square See ‘box’.
- \surd The amazing right adjoint.
- \star can mean:
- $\beta \star \alpha$ Whiskering of natural transformations (definition 2.2.6).
 - $\alpha \star \varphi$ Application of natural transformation α to morphism φ (definition 2.2.6).
- $*$ See ‘asterisk’.
- \circledast See ‘asterisk’.
- str** Structural modality.
- Strict** Strictness type (\mathbf{STRICT}).
- strict** Strictness isomorphism ($\mathbf{STRICT}:\mathbf{ISO}$).
- \subseteq can mean:
- $A \subseteq B$ A is a subset of B .
 - $\Gamma \subseteq \Delta$ Γ is a subpresheaf (objectwise subset) of Δ .
- suc** Successor constructor of \mathbf{Nat} .

T

\perp See ‘bot’.

t Target map in e.g. RG.

ticks(Θ) Concatenation of all the ticks on locks in telescope Θ (fig. 5.5).

\times can mean:

$A \times B$ Binary cartesian product.

$A \times B$ Type of non-dependent pairs.

$(x : A) \times B$ Type of dependent pairs (of identity modality).

$A \times_C B$ Pullback.

$\Delta \times_{\Gamma} W$ Base pullback (definition 2.3.29).

$W \ltimes U$ The multiplier $\sqsubset \ltimes U$ applied to W .

Tm Presheaf of terms in a CwF.

T can mean:

\top Terminal object.

\top Logical truth.

\checkmark can mean:

$\checkmark(u : \mathbb{U})$ Transpension modality (section 7.4.5).

$\checkmark u$ Shorthand for $\checkmark(u : \mathbb{U})$.

$\checkmark(u : \mathbb{U}).A$ Informal notation for $\langle\langle \checkmark(u : \mathbb{U}) \mid A \rangle\rangle$ (section 7.5).

\tilde{g} See ‘ddot’.

\triangleright See Judgements/Turnstiles.

\pitchfork See ‘pitchfork’.

true Constructor of Bool.

tt Constructor of the proof type (PROOF:INTRO).

Tw(\mathcal{C}) Twisted arrow category of \mathcal{C} (definition 2.2.16).

2 See ‘2’ at Numbers.

Ty Presheaf of types in a CwF or natural model.

$\widetilde{\text{Ty}}$ Presheaf of typed terms in a natural model.

type See Judgements/Turnstiles.

U

U can mean:

U_{ℓ} Universe of level ℓ types.

U Universe of unspecified level.

U_{ℓ}^{DD} Discrete universe of discrete types.

U_{ℓ}^{NDD} Non-discrete universe of discrete types.

U_{ℓ}^{NFF} Non-fibrant universe of fibrant types.

U can mean:

U_M Forgetful functor $U_M : \mathbf{EM}(M) \rightarrow \mathcal{C}$ for the monad $M : \mathcal{C} \rightarrow \mathcal{C}$ (proposition 2.2.50).

U_K Forgetful functor $U_K : \mathbf{EM}(K) \rightarrow \mathcal{C}$ for the comonad $K : \mathcal{C} \rightarrow \mathcal{C}$ (proposition 2.2.50).

$U_{\mathcal{L}}$ Forgetful functor $\mathcal{L} \rightarrow \mathcal{C}^{\uparrow}$ in a (damped) NWFS.

$U_{\mathcal{R}}$ Forgetful functor $\mathcal{R} \rightarrow \mathcal{C}^{\uparrow}$ in a (damped) NWFS.

$U_{\mathcal{G}}$ Forgetful functor $U_{\mathcal{G}} = U_{\mathcal{L}}I_{\mathcal{G}} : \mathcal{G} \rightarrow \mathcal{C}^{\uparrow}$ in a left generated (damped) NWFS.

$\sqsubset \ltimes U$ Some multiplier, defined locally.

\mathbb{U} Some shape.

\ddot{g} See ‘ddot’.

unglue Eliminator of the Glue type (GLUE:ELIM).

Unit Unit type.

unit Constructor of the Unit type.

un λ_x Inverse operation to λ -abstraction over x (proposition 3.2.13).

unmerid can mean:

unmerid($u.t$) Informal notation for $\text{prmod}_{\checkmark u} \cdot t$ (section 7.5).

unmerid $_u$ Unit of $\forall u \dashv \checkmark u$ (section 7.4.5).

\uplus can mean:

$A \uplus B$ Binary coproduct.

$A \uplus_C B$ Pushout.

$A \uplus (y : B \dashv \{\varphi ? a\})$ Pushout type (PUSHOUT).

V

\vdash See Judgements/Turnstiles.

\Vdash See Judgements/Turnstiles.

\vec{x} can mean:

\vec{x} Object of the (twisted/damped) arrow category.

$\vec{\varphi}$ Morphism of the (twisted/damped) arrow category.

\vec{g} Shorthand for $U_{\mathcal{L}}\dot{g} = U_{\mathcal{G}}\ddot{g}$.

$\vec{\ell}$ Shorthand for $U_{\mathbf{L}} \ell$.

\vec{r} Shorthand for $U_{\mathbf{R}} r$.

\vee can mean:

$P \vee Q$ Logical disjunction.

$i \vee j$ Connection in CCHM.

where Used in figures of typing rules to save space. What follows are associated rules whose premises are omitted.

$wkn\ u\ a$ Informal notation for $\text{mod}_{\Omega\ u}\ a$ (section 7.5).

W

\wedge can mean:

$P \wedge Q$ Logical conjunction.

$i \wedge j$ Connection in CCHM.

Weld Weld type (WELD).

weld Constructor of the Weld type (WELD:INTRO).

Y

y Yoneda-embedding (definition 2.3.3).

Z

0 See '0' at Numbers.

Index

- 2-category, **37**
- 2-cell, 129, 192
- 2-functor, **37**

- abstraction, **86**
- acyclic, **56**
- ADJ:LRL, **120**
- ADJ:RLR, **120**
- adjoint, **29**, 119, 124
- adjunction, *see* adjoint
 - internal, **149**
 - laws, **29**
 - walking, **150**
- affine
 - cubical set, *see* cubical set, affine
 - multiplier, **192**
- algebra
 - Eilenberg-Moore, **33**
 - for a functor, **31**
 - for a monad, *see* algebra, Eilenberg-Moore
 - for a pointed functor, **33**
 - linear, *see* vector space
 - of an algebraic theory, 73, 75, 76
- algebraic
 - weak factorization system, *see* factorization system, weak, algebraic
- algebraic theory, *see* theory, algebraic
- amazing right adjoint, **206**
- and, *see also* logical conjunction
- AND, **162**
- AND:ELIM:FST, **162**
- AND:ELIM:SND, **162**
- AND:RED:FST, **162**
- AND:RED:SND, **162**
- anonymous
 - function, *see* abstraction
- application, **86**
- arity
 - in an algebraic theory, 73, 75, 76
- arrow
 - category, **23**
 - damped, **226**
 - walking, *see* walking arrow, *see* walking arrow
- ASSERT:SUB, **159**
- auto-internal, **185**
- axiom
 - degeneracy, **272**, **277**, 282
 - K, *see* uniqueness of identity proofs, *see* functor, applicative
 - of choice, **20**
 - of function extensionality, *see* function extensionality
 - of strictness, *see* strictness type
 - of universes, **20**

- base
 - pullback, *see* pullback, base
- base category, **37**
- β -reduction, *see* rule, β
- β -rule, *see* rule, β
- booleans, **93**
- boundary, **193**
 - predicate, **203**
- BOUNDARY, **203**
- BOUNDARY:INTRO, **203**
- bridge, 10, 265
- bridge/path cubical set, *see* cubical set, bridge/path

- cancellative
 - multiplier, **192**
- canonicity, 93, 286
- cartesian
 - cubical set, *see* cubical set, cartesian
 - multiplier, **192**
 - natural transformation, **25**

- product, *see* product, cartesian
- case, 89
- category, **21**
 - arrow, *see* arrow category, *see* arrow category
 - base category, *see* base category
 - discrete, **25**
 - Eilenberg-Moore, **33**
 - model, *see* model category
 - of clocks, **43**
 - of coslices, *see* coslice category
 - of functors, *see* functor category
 - of presheaves, *see* presheaf, category
 - of slices, *see* slice category
 - opposite, *see* opposite category
 - simplex, **40**
 - sub-, *see* subcategory
 - theory, **76**
 - twisted arrow, *see* twisted arrow category
 - under, *see* coslice category
 - with families, **81**, 82
 - morphism of, *see* CwF morphism
- category, 2-, *see* 2-category
- CCHM cubical set, *see* cubical set, CCHM cell
 - of a presheaf, **37**, 100
- central
 - lifting, *see* lifting, central
- choice, *see* axiom of choice, 98
- classifier
 - subobject, *see* subobject classifier
- clock category, *see* category of clocks
- clock-irrelevant, 69, 216, 223, 233, 250
- co-end, **26**
 - dependent, **27**
- co-inductive type, *see* type, codata
- co-unit
 - of a comonad, **31**
 - of an adjunction, **29**
- co-Yoneda lemma, *see* Yoneda, co-Yoneda lemma
 - dependent, *see* Yoneda, co-Yoneda lemma, dependent
- coalgebra
 - Eilenberg-Moore, **33**
 - for a comonad, *see* coalgebra, Eilenberg-Moore
 - for a copointed functor, **33**
 - for a functor, **31**
- codata type, *see* type, codata
- codiscrete, **53**, 68, 215, 217, 222, 233
- coequalizer, **25**
- cofibration, **56**
- colimit, **24**
- comonad, **31**
 - coalgebra, *see* coalgebra, Eilenberg-Moore
 - laws, **31**
- composition
 - of morphisms, **21**
 - of natural transformations, **22**
- computation
 - rule, *see* rule, computation
- Conduché, 224
- conjunction, *see* logical conjunction
- connection-free, **192**
- constructive, 97
- constructor, **85**
- context, 78
 - empty, 79
 - extension, 80, 82, 84
 - restriction, 159
 - shape, *see* shape context
 - split, 152
- contextually fibrant, *see* fibrant, contextually, *see* fibrant, degenerately
- continuous
 - modality, **266**
- contramodality, **275**
- copointed functor, *see* functor, copointed algebra, *see* coalgebra for a copointed functor
- COPROD, **89**
- COPROD:ELIM, **89**
- COPROD:INL, **89**
- COPROD:INL:BETA, **89**
- COPROD:INR, **89**
- COPROD:INR:BETA, **89**
- coproduct, **25**
 - type, **89**
- coreplacement
 - left, *see* left coreplacement
- coslice category, **28**
- CTX-EXT, **80**
- CTX-EXT:ETA, **80**
- CTX-EXT:INTRO, **80**
- CTX-EXT:VAR, **80**
- CTX-EXT:VAR:BETA, **80**
- CTX-EXT:WKN, **80**
- CTX-EXT:WKN:BETA, **80**
- CTX-MODEXT, **135**
- CTX-MODEXT:ETA, **135**
- CTX-MODEXT:INTRO, **135**
- CTX-MODEXT:VAR, **135**
- CTX-MODEXT:VAR:2CELL, **135**

- CTX-MODEXT:VAR:BARE, **136**
 CTX-MODEXT:VAR:BETA, **135**
 CTX-MODEXT:WKN, **135**
 CTX-MODEXT:WKN:BETA, **135**
 CTX-MODRESTR, **177**
 CTX-MODRESTR:ETA, **177**
 CTX-MODRESTR:INTRO, **177**
 CTX-MODRESTR:VAR, **177**
 CTX-MODRESTR:WKN, **177**
 CTX-MODRESTR:WKN:BETA, **177**
 CTX-RESTR, **159**
 CTX-RESTR:ETA, **159**
 CTX-RESTR:INTRO, **159**
 CTX-RESTR:VAR, **159**
 CTX-RESTR:WKN, **159**
 CTX-RESTR:WKN:BETA, **159**
 cube, *see also* cubical set
 twisted, **195**
 cubical set
 affine, **42**
 bridge/path, **43**
 cartesian, **41**
 CCHM, **42**
 depth, **43**
 CwF, *see* category with families
 morphism, **82**, **113**

 damped
 arrow
 category, *see* arrow category,
 damped
 walking, *see* walking arrow,
 damped
 data type, *see* type, data
 DDTT, *see* directed dependent type
 theory
 decoding
 of a type, **95**
 definitional
 equality, *see* equality, judgemental
 degeneracy axiom, *see* axiom, degeneracy
 degeneracy map, **40**
 democratic, **83**
 dependent
 co-end, *see* co-end, dependent
 co-Yoneda lemma, *see* Yoneda, co-
 Yoneda lemma, dependent
 end, *see* end, dependent
 function, *see* function, dependent
 naturality, **116**
 pair, *see* pair, dependent
 right adjoint, **123**, **132**
 weak, **134**, **143**
 type
 theory, **78**
 Yoneda lemma, *see* Yoneda lemma,
 dependent
 depth, **274**, **281**
 cubical set, *see* cubical set, depth
 destructor, **85**
 diagram, **24**
 dimensionally split, **193**
 directed
 dependent type theory, **1**, **3**
 type theory, **283**
 discrete, **10**, **53**, **54**, **68**, **69**, **215**, **216**, **218**,
 222, **223**, **233**, **251**
 category, *see* category, discrete
 replacement, **12**
 disjunction, *see* logical disjunction
 division, left, *see* left division
 DLC:EXTEND, **242**
 DLC:EXTEND:ASSOC, **242**
 DLC:EXTEND:ETA, **242**
 DLC:EXTEND:MULT, **242**
 DLC:EXTEND:NAT:COD, **242**
 DLC:EXTEND:NAT:DOM, **242**
 DLC:EXTEND:RCOUNT, **242**
 DLC:EXTEND:UNIT, **242**
 double
 negation, *see* negation, double
 DRA, *see* dependent right adjoint
 DRA, **123**
 DRA:BETA, **123**
 DRA:ELIM, **123**
 DRA:ELIM:SUB, **123**
 DRA:ETA, **123**
 DRA:INTRO, **123**
 DRA:INTRO:SUB, **123**
 DRA:SUB, **123**
 DRR, **239**
 DRR:ASSOC, **239**
 DRR:ELIM, **245**
 DRR:FMAP, **239**
 DRR:FMAP:COMP, **239**
 DRR:FMAP:ID, **239**
 DRR:LUNIT, **239**
 DRR:MULT, **239**
 DRR:MULT:BETA, **245**
 DRR:MULT:NAT, **239**
 DRR:RUNIT, **239**
 DRR:UNIT, **239**
 DRR:UNIT:BETA, **245**
 DRR:UNIT:NAT, **239**
 DTT, *see* dependent type theory
 duality, **21**

- duplication
 - comonadic, **31**
- E, *see also* extensionality, **157**
- Eilenberg-Moore
 - algebra, *see* algebra, Eilenberg-Moore
 - category, *see* category, Eilenberg-Moore
 - coalgebra, *see* coalgebra, Eilenberg-Moore
- elimination rule, *see* rule, elimination
- eliminator, **85**
- empty
 - context, *see* context, empty
- EMPTY, **90**
- EMPTY-CTX, **80**
- EMPTY-CTX:ETA, **80**
- EMPTY-CTX:INTRO, **80**
- EMPTY:ELIM, **90**
- encoding
 - of a type, **95**
- end, **25**
 - dependent, **27**
- epi, *see* epimorphism
- epimorphism, **22**, **46**
 - split, **22**, **46**
- EQ, **164**
- EQ:BETA, **164**
- EQ:J, **164**
- EQ:RED, **164**
- EQ:REFLECTION, **164**
- equality, **97**
 - definitional, *see* equality, judgemental
 - judgement, *see* judgement, equality judgemental, **90**
 - proof, **90**
 - propositional, **90**
- equalizer, **25**
- equifibred, *see* cartesian natural transformation
- equipment, **284**
- equivalence
 - weak, *see* weak equivalence
- η -expansion, *see* rule, η
- η -rule, *see* rule, η
- excluded middle, **97**
- existential quantification, *see* quantification, existential
- expansion
 - rule, *see* rule, η
- extension
 - of a context, *see* context extension type, **168**
- EXTENSION, **168**
- EXTENSION:BETA, **168**
- EXTENSION:ELIM, **168**
- EXTENSION:ETA, **168**
- EXTENSION:INTRO, **168**
- EXTENSION:STRICT, **168**
- extensionality, **92**, **157**
 - function, *see* function extensionality
- face map, **40**
- factorization system
 - orthogonal, **53**
 - weak, **55**, **56**
 - algebraic, **55**
 - functorial, **55**, **57**
 - natural, **55**, **60**, **228**
 - natural, Grandis-Tholen, **62**, **229**
- faithful, **23**
 - fully, *see* fully faithful
- false, *see also* logical falsehood
- falsehood, *see* logical falsehood
- FALSEHOOD, **162**
- FALSEHOOD:ELIM, **162**
- FALSEHOOD:ETA, **162**
- family
 - of types, *see* type family
- fibrant, **215**
 - contextually, **228**
 - degenerately, **228**
 - replacement, **217**
- fibration, **56**
- field, *see* eliminator, *see* type, codata
- filler, **52**
- final
 - object, *see* terminal object
- Fitch-style, **131**, **152**
- formation rule, *see* rule, formation
- fresh
 - weakening, **192**
 - modality, **197**
- FTR:COMP:CTX, **114**
- FTR:COMP:SUB, **114**
- FTR:COMP:TM, **114**
- FTR:COMP:TY, **114**
- FTR:CTX, **114**
- FTR:CTX-EXT, **114**
- FTR:CTX-EXT:INTRO, **114**
- FTR:CTX-EXT:VAR, **114**
- FTR:CTX-EXT:WKN, **114**
- FTR:EMPTY-CTX, **114**
- FTR:EMPTY-CTX:INTRO, **114**

- FTR:ID:CTX, **114**
- FTR:ID:SUB, **114**
- FTR:ID:TM, **114**
- FTR:ID:TY, **114**
- FTR:SUB, **114**
- FTR:SUB:COMP, **114**
- FTR:SUB:ID, **114**
- FTR:TM, **114**
- FTR:TY, **114**
- full, **23**
 - subcategory, **23**
- fully faithful, **23**
- function
 - anonymous, *see* abstraction
 - application, *see* application
 - dependent, **86**, **247**
 - modal, **137**
 - extensionality, **93**
 - non-dependent, **87**
- functor, **21**, **113**
 - algebra, *see* algebra for a functor
 - applicative, **139**, **146**, **147**
 - category, **22**
 - coalgebra, *see* coalgebra for a functor
 - contravariant, **21**
 - copointed, **31**
 - covariant, **21**
 - opposite, *see* opposite functor
 - pointed, **31**
- functor, 2-, *see* 2-functor
- functor, pro-, *see* profunctor
- functor, pseudo-, *see* pseudofunctor
- functorial
 - weak factorization system, *see* factorization system, weak, functorial
- FWFS, *see* factorization system, weak, algebraic

- GAT, *see* generalized algebraic theory
- generalized
 - algebraic theory, **71**
- generalized, algebraic theory, **76**
- generating
 - left map, *see* left map, generating
- GLUE, **171**
- glue type, **169**, **211**, **249**
- GLUE:BETA, **171**
- GLUE:ELIM, **171**
- GLUE:ELIM:STRICT, **171**
- GLUE:ETA, **171**
- GLUE:FIB, **249**
- GLUE:INTRO, **171**
- GLUE:STRICT, **171**

- GLUE:STRICT:BETA, **171**
- gluing, **130**, **142**
- Grandis-Tholen
 - NWFS, *see* factorization system, weak, natural, Grandis-Tholen
- graph
 - reflexive, *see* reflexive graph
- Grothendieck universe, **20**
- group, **21**, **74**
- guarded recursion, **146**

- HDPM, *see* higher-dimensional pattern matching
- heterogeneous, *see also* homogeneous, **215**, **217**
- higher dimensional
 - pattern matching, **188**
- Hofmann-Streicher universe, *see* universe, Hofmann-Streicher
- homogeneous, *see also* heterogeneous, **216**, **217**
- homotopy
 - type theory, **3**, **41**, **93**
- HoTT, *see* homotopy type theory

- I, *see also* intensionality, **157**
- ID, **91**
- ID:ETA, **91**
- ID:FIB, **248**
- ID:INTRO, **91**
- ID:J, **91**
- ID:J:BETA, **91**
- ID:REFLECTION, **91**
- ID:UIP, **91**
- idempotent, **35**
- identity
 - extension, **258**, **259**, **260**, **263**, **265**, **272**, **277**, **280**, **281**
 - morphism, **21**
 - type, **90**, **248**
- IEL, *see* identity extension
- IMG:CTX-EXT:ETA, **114**
- IMG:CTX-EXT:INTRO, **114**
- IMG:CTX-EXT:VAR:BETA, **114**
- IMG:CTX-EXT:WKN:BETA, **114**
- IMG:EMPTY-CTX:ETA, **114**
- IMG:EMPTY-CTX:INTRO, **114**
- implication, *see* logical implication
- index
 - of a type family, **92**
- inductive type, *see* type, data
- inhabited, **215**, **217**, **222**
- initial

- object, **25**
- injective
 - presheaf morphism, **46**
- intensionality, **92**, **157**
- internal
 - auto-, *see* auto-internal
 - meta-, *see* meta-internal
- introduction rule, *see* rule, introduction
- irrelevance
 - modality, **267**
- isomorphic, *see* isomorphism
- isomorphism, **95**

- J-rule, *see* rule, J
- JUD:ASSERT, **159**
- JUD:CTX, **79**
- JUD:PROP, **159**
- JUD:SUB, **79**
- JUD:TM, **80**
- JUD:TY, **80**
- judgement, **77**
 - equality, **77**
 - form, **77**
- judgemental
 - equality, *see* equality, judgemental

- Kan, **70**, **217**, **219**, **224**
 - contextually, **225**
- kan, **234**, **251**
 - contextually, **221**, **234**, **252**

- label, **20**
- λ -abstraction, *see* abstraction
- λ -calculus
 - simply typed, **77**
 - untyped, **74**
- large
 - in set theory, **20**
- law
 - adjunction, *see* adjunction laws
 - comonad, *see* comonad laws
 - monad, *see* monad laws
 - of excluded middle, *see* excluded middle
 - middle
- Lawvere theory, *see* theory, Lawvere
- LC:EXTEND, **241**
- LC:EXTEND:ASSOC, **241**
- LC:EXTEND:ETA, **241**
- LC:EXTEND:MULT, **241**
- LC:EXTEND:NAT:COD, **241**
- LC:EXTEND:NAT:DOM, **241**
- LC:EXTEND:RCOUNT, **241**
- LC:EXTEND:UNIT, **241**

- left
 - adjoint, **29**
 - coreplacement, **57**
 - division, **152**
 - generated NWFS, **64**, **229**
 - lifting, *see* lifting, left
 - lifting property, *see* lifting property,
 - see* lifting property
 - map, **53**, **56**
 - generating, **64**, **229**
 - level, *see* size
 - lifting, **52**
 - central, **49**
 - left, **49**
 - problem, **52**
 - property, **55**, **226**
 - right, **49**
 - limit, **24**
 - linear
 - algebra, *see* vector space
 - map, *see* vector space
 - Löb induction, **148**
 - lock, **135**
 - LOCK, **135**
 - LOCK:COMP, **135**
 - LOCK:COMP:FMAP, **135**
 - LOCK:FMAP, **135**
 - LOCK:FMAP:COMP, **135**
 - LOCK:FMAP:ID, **135**
 - LOCK:ID, **135**
 - LOCK:ID:FMAP, **135**
 - LOEB, **148**
 - LOEB:BETA, **148**
 - logical
 - conjunction, **96**, **161**
 - disjunction, **96**, **163**
 - falsehood, **96**, **161**
 - implication, **96**
 - negation, **97**
 - truth, **96**, **161**
 - LSR, *see* [LSR17]

 - meridian, **204**
 - meta-internal, **185**
 - metacontext
 - of an algebraic theory, **76**
 - metavariable, **165**
 - middle
 - excluded, *see* excluded middle
 - mill, **211**
 - modality, **129**, **192**
 - contra-, *see* contramodality
 - mode, **129**, **192**

- theory, **129**, **192**
- model
 - category, **56**
 - natural, *see* natural model
 - of an algebraic theory, **73**, **75**, **76**
 - structure, **56**
- MODPI, **138**
- MODPI:BETA, **138**
- MODPI:ELIM, **138**
- MODPI:ETA, **138**
- MODPI:INTRO, **138**
- monad, **31**
 - algebra, *see* algebra, Eilenberg-Moore
 - laws, **31**
- mono, *see* monomorphism
- monoid, **4**
- monomorphism, **22**, **46**
 - split, **22**
- morphism, **21**
 - of CwFs, *see* CwF morphism
- MTT, **126**
- multimode, **10**, **112**
- multiplication
 - monadic, **31**
- multiplier, **192**
- multisorted
 - algebraic theory, *see* theory, algebraic, multisorted

- NAT:ELIM, **94**
- NATTRANS:COMP:CTX, **116**
- NATTRANS:COMP:TY, **116**
- NATTRANS:CTX, **116**
- NATTRANS:ID:CTX, **116**
- NATTRANS:ID:TY, **116**
- NATTRANS:SUB, **116**
- NATTRANS:TM, **116**
- NATTRANS:TY, **116**
- NATTRANS:TY:COEND, **116**
- NATTRANS:TY:SUB, **116**
- NATTRANS:WHISKER:CTX, **116**
- NATTRANS:WHISKER:TY, **116**
- natural, **22**
 - dependent naturality, *see* dependent naturality
 - isomorphism, **22**
 - model, **84**
 - transformation, **22**, **115**
 - pseudo-, *see* pseudonatural transformation
 - weak factorization system, *see* factorization system, weak, natural
- natural numbers
 - in type theory, **94**
- negation, *see* logical negation
 - double, **97**
- nominal set, **194**
- non-dependent, *see also* weakening
 - function, *see* function, non-dependent
 - pair, *see* pair, non-dependent
- NWFS, *see* factorization system, weak, natural

- object
 - final, *see* terminal object
 - initial, *see* initial object
 - of a category, **21**
 - terminal, *see* terminal object
- OFS, *see* factorization system, orthogonal
- operator
 - of an algebraic theory, **73**, **75**, **76**
- opposite
 - category, **21**
 - functor, **21**
- or, *see also* logical disjunction
- OR, **162**
- OR:ELIM, **162**
- OR:ETA, **162**
- OR:INL:BETA, **162**
- OR:INR:BETA, **162**
- OR:RED:INL, **162**
- OR:RED:INR, **162**
- orthogonal, **52**
 - factorization system, *see* factorization system, orthogonal
- over category, *see* slice category

- pair
 - dependent, **88**, **248**
 - non-dependent, **88**
- ParamDTT, *see also* [NVD17a], **257**
- ParamDTT_♠, **271**
- parameter
 - of a type family, **92**
- parametric
 - modality, **267**, **275**
 - quantifiers, **9**, **265**
- parametricity, **9**, **265**
- ParamMTT, **269**
- partial, **158**
- path, **10**, **265**
- pattern, **90**
- pattern matching
 - higher dimensional, *see* higher-dimensional pattern matching

- PHI, **207**
- PHI:BOUNDARY, **207**
- PHI:SECTION, **207**
- PI, **87**
- II-type, *see* function, dependent
- PI:BETA, **87**
- PI:ELIM, **87**
- PI:ETA, **87**
- PI:FIB, **247**
- PI:FUNEXT, **91**
- PI:INTRO, **87**
- PI:UNLAMBDA, **87**
- PI:UNLAMBDA:BETA, **87**
- PI:UNLAMBDA:ETA, **87**
- point
 - of a presheaf, **38**
- pointed functor, *see* functor, pointed
 - algebra, *see* algebra for a pointed functor
- pointwise
 - modality, **265**
- pole, **204**
- premise, **77**
- presheaf, **37**, **288**
 - category, **37**
- pro-arrow, **284**
- product
 - cartesian, **24**
- profunctor, **37**
- proof
 - irrelevant, **92**
 - type, **167**
- PROOF, **167**
- PROOF:ELIM, **167**
- PROOF:ETA, **167**
- PROOF:INTRO, **167**
- PROP:SUB, **159**
- PROPDRA, **177**
- PROPDRA:ELIM, **177**
- PROPDRA:INTRO, **177**
- PROPDRA:RED, **177**
- propositional
 - equality, *see* equality, propositional
- PROPUNI, **160**
- PROPUNI:BETA, **160**
- PROPUNI:ELIM, **160**
- PROPUNI:ETA, **160**
- PROPUNI:INTRO, **160**
- pseudofunctor, **37**
- pseudonatural transformation, **37**
- pseudotype, **160**
- PSI, **209**
- PSI:BETA, **209**
- PSI:BOUNDARY, **209**
- PSI:ELIM, **209**
- PSI:ETA, **209**
- PSI:INTRO, **209**
- PSI:INTRO:BOUNDARY, **209**
- pullback, **25**
 - base, **48**
- pushout, **25**
- PUSHOUT, **172**
- PUSHOUT:ELIM, **172**
- PUSHOUT:INL, **172**
- PUSHOUT:INL:BETA, **172**
- PUSHOUT:INR, **172**
- PUSHOUT:INR:BETA, **172**
- PUSHOUT:TIP, **172**
- quantifiable, **192**
- quantification
 - existential, **97**
 - universal, **97**
- record type, *see* type, codata
- reduction
 - rule, *see* rule, β
- reflection
 - rule, *see* rule, reflection
- reflexive
 - graph, **39**
- RelDTT, *see also* [ND18a], **257**
- RelDTT_h, **277**
- RelMTT, **275**
- replacement
 - discrete, *see* discrete replacement
 - fibrant, *see* fibrant replacement
 - left co-, *see* left coreplacement
 - right, *see* right replacement
- representable
 - morphism of presheaves, **48**, **84**
 - presheaf, **38**
- restriction
 - of a context, *see* context restriction
 - of a presheaf cell, **38**
- retraction, **22**
- right
 - adjoint, **29**, **124**
 - amazing, *see* amazing right adjoint
 - dependent, *see* dependent right adjoint
 - lifting, *see* lifting, right
 - lifting property, *see* lifting property, *see* lifting property
 - map, **53**, **56**

- replacement, **57**
- robust, **232**
- RR, **238**
- RR:ASSOC, **238**
- RR:ELIM, **246**
- RR:FMAP, **238**
- RR:FMAP:COMP, **238**
- RR:FMAP:ID, **238**
- RR:LUNIT, **238**
- RR:MULT, **238**
- RR:MULT:BETA, **246**
- RR:MULT:NAT, **238**
- RR:RUNIT, **238**
- RR:UNIT, **238**
- RR:UNIT:BETA, **246**
- RR:UNIT:NAT, **238**
- rule
 - β , **86**
 - computation, *see also* rule, β
 - elimination, **85**
 - η , **86**
 - η , **86**
 - expansion, *see* rule, η
 - formation, **85**
 - introduction, **85**
 - J, **91**, **164**
 - reduction, *see* rule, β
 - reflection, **92**, **164**
- Schanuel topos, **194**
- section, **22**
- Segal, **69**, **217**, **219**, **224**
 - contextually, **225**
- segal, **234**, **251**
 - contextually, **221**, **234**
- semicartesian
 - multiplier, **192**
- set
 - nominal, *see* nominal set
- setoid, **93**
- shape, **192**
 - context, **191**
 - of a presheaf cell, **37**
- shape-irrelevance
 - modality, **267**
- Sierpiński topos, **43**
- SIGMA, **88**
- Σ -type, *see* pair, dependent
- SIGMA:ETA, **88**
- SIGMA:FIB, **248**
- SIGMA:FST, **88**
- SIGMA:FST:BETA, **88**
- SIGMA:INTRO, **88**
- SIGMA:SND, **88**
- SIGMA:SND:BETA, **88**
- simple
 - algebraic theory, *see* theory, algebraic, simple
- simplex, *see also* simplicial set, **40**
 - category, *see* category, simplex
- simplicial set, **40**
- simply typed λ -calculus, *see* λ -calculus, simply typed
- simultaneous substitution, *see* substitution
- singleton
 - type, *see* unit type/constructor
- SIP, *see* structure identity principle
- size, **85**
 - in set theory, **20**
 - preserving, **115**
- slice category, **28**
- small
 - in set theory, **20**
 - object argument, **64**, **229**
- solution, **52**
- sort
 - of an algebraic theory, **75**, **76**
- split
 - context, *see* context, split
 - dimensionally, *see* dimensionally split
- split epimorphism, *see* epimorphism, split
- split monomorphism, *see* monomorphism, split
- spooky, **193**
- stable
 - NWFS, **231**
- STLC, *see* λ -calculus, simply typed
- stream, **148**
- strict
 - CwF morphism, *see* CwF morphism
- STRICT, **170**
- strict type, *see* strictness type
- STRICT:BETA, **170**
- STRICT:ELIM, **170**
- STRICT:ETA, **170**
- STRICT:FIB, **249**
- STRICT:INTRO, **170**
- STRICT:ISO, **170**
- STRICT:ISO:STRICT, **170**
- STRICT:STRICT, **170**
- strictness axiom, *see* strictness type
- strictness type, **169**, **249**
- structural
 - modality, **275**

- structure identity principle, **6**
- SUB:ASSOC, **79**
- SUB:COMP, **79**
- SUB:ID, **79**
- SUB:LUNIT, **79**
- SUB:RUNIT, **79**
- subcategory, **23**
 - full, *see* full subcategory
- subobject, **36**
 - classifier, **36**
- subsingleton, **215**, **217**, **222**
- substitution, **78**, **79**
 - in an algebraic theory, **76**
 - modality, **196**
- surjective
 - presheaf morphism, **46**
- syntax
 - of an algebraic theory, **73**, **75**, **76**
- System F, **259**
- System $F\omega$, **260**

- term, **79**
 - of an algebraic theory, **73**, **75**, **76**
 - substitution, *see* substitution
- terminal
 - object, **25**
- theory
 - algebraic
 - generalized, *see* generalized algebraic theory
 - multisorted, **75**
 - simple, **73**
 - category, *see* category theory
 - dependent type, *see* dependent type theory
 - Lawvere, **74**
 - type
 - dependent, *see* dependent type theory
 - homotopy, *see* homotopy type theory
- tick, **133**
- TM:SUB, **80**
- topos
 - Schanuel, *see* Schanuel topos
- topos of trees, **43**
- total, **158**
- TRANSP:ELIM, **205**
- TRANSP:ELIM:POLE, **205**
- TRANSP:ELIM:SECTION, **205**
- transpension
 - modality, **197**
- transpensive, **210**

- TRANSPOSE, **120**
- TRANSPOSE:CANCEL, **120**
- TRANSPOSE:COEND, **120**
- TRANSPOSE:DEF, **120**
- TRANSPOSE:SUB, **120**
- transposition
 - w.r.t. an adjunction, **29**, **119**
- tree, *see also* topos of trees
- triangle
 - identities, *see* adjunction laws
- trivial
 - (co)fibration, **56**
- true, *see also* logical truth
- truth, *see* logical truth
- TRUTH, **162**
- TRUTH:INTRO, **162**
- twisted
 - cube, *see* cube, twisted
- twisted arrow category, **23**
- TY:SUB, **80**
- type, **79**
 - co-inductive, *see* type, codata
 - codata, **86**
 - coproduct, *see* coproduct type
 - data, **86**, **90**
 - extension, *see* extension type
 - family, **92**
 - function, *see* function
 - identity, *see* identity type
 - inductive, *see* type, data
 - of proofs, *see* proof type
 - pair, *see also* pair
 - pseudo-, *see* pseudotype
 - record, *see* type, codata
 - substitution, *see* substitution
- type former, **85**
- typing rule, **77**

- UIP, *see* uniqueness of identity proofs
- under category, *see* coslice category
- UNI, **94**
- UNI:BETA, **94**
- UNI:ELIM, **94**
- UNI:ETA, **94**
- UNI:INTRO, **94**
- uniqueness
 - of identity proofs, **90**
- unit
 - of a monad, **31**
 - of an adjunction, **29**
 - type/constructor, **88**
- UNIT, **89**
- UNIT:ETA, **89**

- UNIT:INTRO, **89**
- universal quantification, *see* quantification, universal
- universe
 - axiom, *see* axiom of universes
 - Grothendieck, *see* Grothendieck universe
 - Hofmann-Streicher, **106**
 - in type theory, **95**
 - level, *see* size
- UNTRANSPOSE, **120**
- UNTRANSPOSE:CANCEL, **120**
- UNTRANSPOSE:COEND, **120**
- UNTRANSPOSE:DEF, **120**
- UNTRANSPOSE:SUB, **120**
- untyped λ -calculus, *see* λ -calculus, untyped
- UT, **162**
- variable, 79
- vector space, 21
- walking
 - arrow, **23**
 - damped, **226**
- WDRA, **136**
- WDRA:BETA, **136**
- WDRA:ELIM, **136**
- WDRA:INTRO, **136**
- weak
 - CwF morphism, *see* CwF morphism
 - DRA, *see* dependent right adjoint, weak
 - equivalence, **56**
 - factorization system, *see* factorization system, weak
 - weakening, 79
 - fresh, *see* fresh weakening
 - modality, **196**
 - WELD, **174**
 - weld type, 169, 211
 - WELD:BETA, **174**
 - WELD:ELIM, **174**
 - WELD:INTRO, **174**
 - WELD:INTRO:STRICT, **174**
 - WELD:STRICT, **174**
 - WELD:STRICT:BETA, **174**
 - WFS, *see* factorization system, weak
 - where, *see* List of Mathematical Symbols
 - whiskering
 - of morphism and natural transformation, **22**
 - of natural transformations, **22**
- XTT, 93
- Yoneda
 - co-Yoneda lemma, **44**
 - dependent, **45**
 - embedding, **38**, 46
 - lemma, **43**
 - dependent, **44**
- ZFC, **20**
- zigzag, **26**

Bibliography

- [Abe06] A. Abel. “A Polymorphic Lambda-Calculus with Sized Higher-Order Types”. PhD thesis. Ludwig-Maximilians-Universität München, 2006. URL: <http://www.cse.chalmers.se/~abela/diss.pdf> (pp. 13, 111, 133, 152).
- [Abe08] A. Abel. “Polarised subtyping for sized types”. In: *Mathematical Structures in Computer Science* 18.5 (2008), pp. 797–822. DOI: 10.1017/S0960129508006853. URL: <https://doi.org/10.1017/S0960129508006853> (pp. 3, 13, 111, 133, 152).
- [AGJ14] R. Atkey, N. Ghani, and P. Johann. “A Relationally Parametric Model of Dependent Type Theory”. In: *Principles of Programming Languages*. 2014. DOI: 10.1145/2535838.2535852 (pp. ix, 9–11, 40, 98, 182, 183, 258, 263, 264, 280, 281).
- [AHH18] C. Angiuli, K.-B. Hou (Favonia), and R. Harper. “Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities”. In: *Computer Science Logic (CSL 2018)*. Ed. by D. Ghica and A. Jung. Vol. 119. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 6:1–6:17. ISBN: 978-3-95977-088-0. DOI: 10.4230/LIPIcs.CSL.2018.6. URL: <http://drops.dagstuhl.de/opus/volltexte/2018/9673> (pp. 69, 145, 182).
- [AK16] T. Altenkirch and A. Kaposi. “Normalisation by Evaluation for Dependent Types”. In: *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*. 2016. DOI: 10.4230/LIPIcs.FSCD.2016.6 (pp. 130, 142).
- [All87] S. F. Allen. “A non-type-theoretic semantics for type-theoretic language”. PhD thesis. Ithaca, NY, USA: Cornell University, 1987 (p. 144).

- [AM13] R. Atkey and C. McBride. “Productive coprogramming with guarded recursion”. In: *ACM SIGPLAN International Conference on Functional Programming, ICFP’13, Boston, MA, USA - September 25 - 27, 2013*. Ed. by G. Morrisett and T. Uustalu. ACM, 2013, pp. 197–208. DOI: 10.1145/2500365.2500597. URL: <https://doi.org/10.1145/2500365.2500597> (pp. 9, 146).
- [Ang19] C. Angiuli. “Computational Semantics of Cartesian Cubical Type Theory”. PhD thesis. Carnegie Mellon University, 2019 (p. 144).
- [AS12] A. Abel and G. Scherer. “On Irrelevance and Algorithmic Equality in Predicative Type Theory”. In: *Logical Methods in Computer Science* 8.1 (2012). TYPES’10 special issue., pp. 1–36. DOI: [http://dx.doi.org/10.2168/LMCS-8\(1:29\)2012](http://dx.doi.org/10.2168/LMCS-8(1:29)2012) (pp. 11, 111, 127, 152).
- [Atk12] R. Atkey. “Relational Parametricity for Higher Kinds”. In: *Computer Science Logic (CSL’12) - 26th International Workshop/21st Annual Conference of the EACSL*. Vol. 16. Leibniz International Proceedings in Informatics (LIPIcs). 2012, pp. 46–61. DOI: 10.4230/LIPIcs.CSL.2012.46 (pp. 258, 260, 261, 263, 280).
- [AVW17] A. Abel, A. Vezzosi, and T. Winterhalter. “Normalization by Evaluation for Sized Dependent Types”. In: *Proc. ACM Program. Lang.* 1.ICFP (Aug. 2017), 33:1–33:30. ISSN: 2475-1421. DOI: 10.1145/3110277. URL: <http://doi.acm.org/10.1145/3110277> (pp. 11, 111, 267).
- [Awo18] S. Awodey. “Natural models of homotopy type theory”. In: *Mathematical Structures in Computer Science* 28.2 (2018), pp. 241–286. DOI: 10.1017/S0960129516000268 (pp. 83–85, 142).
- [BB08] B. Barras and B. Bernardo. “The Implicit Calculus of Constructions as a Programming Language with Dependent Types”. In: *FOSSACS 2008, part of ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 365–379. DOI: 10.1007/978-3-540-78499-9_26 (pp. 11, 111).
- [BCH14] M. Bezem, T. Coquand, and S. Huber. “A Model of Type Theory in Cubical Sets”. In: *19th International Conference on Types for Proofs and Programs (TYPES 2013)*. Vol. 26. Dagstuhl, Germany, 2014, pp. 107–128. ISBN: 978-3-939897-72-9. DOI: 10.4230/LIPIcs.TYPES.2013.107. URL: <http://drops.dagstuhl.de/opus/volltexte/2014/4628> (pp. 9, 42, 69, 182, 189, 194).

- [BCM15] J.-P. Bernardy, T. Coquand, and G. Moulin. “A Presheaf Model of Parametric Type Theory”. In: *Electron. Notes in Theor. Comput. Sci.* 319 (2015), pp. 67–82. DOI: <http://dx.doi.org/10.1016/j.entcs.2015.12.006> (pp. ix, 9–11, 42, 182, 184, 187–189, 194, 197, 206–209, 258, 264, 280, 287, 298).
- [BdP00] G. M. Bierman and V. C. V. de Paiva. “On an Intuitionistic Modal Logic”. In: *Studia Logica* 65.3 (2000). DOI: 10.1023/A:1005291931660 (pp. 142, 153).
- [BGM17] P. Bahr, H. B. Grathwohl, and R. E. Møgelberg. “The clocks are ticking: No more delays!” In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. 2017, pp. 1–12. DOI: 10.1109/LICS.2017.8005097. URL: <https://doi.org/10.1109/LICS.2017.8005097> (pp. 13, 127, 130, 131, 133, 146, 148, 149, 153).
- [BGM19] P. Bahr, C. U. Graulund, and R. E. Møgelberg. “Simply RaTT: A Fitch-style Modal Calculus for Reactive Programming Without Space Leaks”. In: *Proc. ACM Program. Lang.* 3.ICFP (July 2019), 109:1–109:27. ISSN: 2475-1421. DOI: 10.1145/3341713 (p. 153).
- [Bir+12] L. Birkedal, R. Møgelberg, J. Schwinghammer, and K. Støvring. “First steps in synthetic guarded domain theory: step-indexing in the topos of trees”. In: *Logical Methods in Computer Science* 8.4 (2012). Ed. by P. Baillot. DOI: 10.2168/LMCS-8(4:1)2012 (pp. 43, 127, 146, 147).
- [Bir+19] L. Birkedal, A. Bizjak, R. Clouston, H. B. Grathwohl, B. Spitters, and A. Vezzosi. “Guarded Cubical Type Theory”. In: *Journal of Automated Reasoning* 63.2 (Aug. 2019), pp. 211–253. ISSN: 1573-0670. DOI: 10.1007/s10817-018-9471-7. URL: <https://doi.org/10.1007/s10817-018-9471-7> (pp. 158, 182).
- [Bir+20] L. Birkedal, R. Clouston, B. Manna, R. E. Møgelberg, A. M. Pitts, and B. Spitters. “Modal dependent type theory and dependent right adjoints”. In: *Mathematical Structures in Computer Science* 30.2 (2020), pp. 118–138. DOI: 10.1017/S0960129519000197 (pp. 13, 14, 112, 122–125, 128, 130–132, 134, 139, 141, 144, 147, 153, 186, 192, 203, 204).
- [Bir00] L. Birkedal. “Developing Theories of Types and Computability via Realizability”. In: *Electronic Notes in Theoretical Computer Science* 34 (2000) (p. 152).

- [Biz+16] A. Bizjak, H. B. Grathwohl, R. Clouston, R. E. Møgelberg, and L. Birkedal. “Guarded Dependent Type Theory with Coinductive Types”. In: *FOSSACS '16*. 2016. DOI: 10.1007/978-3-662-49630-5_2 (pp. 127, 130, 145, 146, 148, 149, 153, 183).
- [BJP12] J.-P. Bernardy, P. Jansson, and R. Paterson. “Proofs for Free — Parametricity for Dependent Types”. In: *Journal of Functional Programming* 22.02 (2012), pp. 107–152. DOI: 10.1017/S0956796812000056. URL: <https://publications.lib.chalmers.se/cpl/record/index.xhtml?pubid=135303> (pp. 264, 286).
- [BM15] A. Bizjak and R. E. Møgelberg. “A Model of Guarded Recursion With Clock Synchronisation”. In: *Electr. Notes Theor. Comput. Sci.* (2015). DOI: 10.1016/j.entcs.2015.12.007. URL: <https://doi.org/10.1016/j.entcs.2015.12.007> (p. 146).
- [BM18] A. Bizjak and R. E. Møgelberg. “Denotational semantics for guarded dependent type theory”. In: *CoRR* abs/1802.03744 (2018). arXiv: 1802.03744. URL: <http://arxiv.org/abs/1802.03744> (pp. 12, 43, 69, 182–184, 287, 288).
- [BPT17] S. Boulier, P. Pédrot, and N. Tabareau. “The next 700 syntactical models of type theory”. In: *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*. 2017, pp. 182–194. DOI: 10.1145/3018610.3018620. URL: <https://doi.org/10.1145/3018610.3018620> (p. 93).
- [Bra13] E. Brady. “Idris, a general-purpose dependently typed programming language: Design and implementation”. In: *Journal of Functional Programming* 23 (05 2013), pp. 552–593 (p. 2).
- [BT17] S. Boulier and N. Tabareau. “Model structure on the universe in a two level type theory”. Working paper or preprint. 2017. URL: <https://hal.archives-ouvertes.fr/hal-01579822> (pp. 12, 214, 220, 228).
- [BV17] J.-P. Bernardy and A. Vezzosi. *Parametric application*. Private communication. 2017 (p. 188).
- [Car78] J. Cartmell. “Generalised Algebraic Theories and Contextual Categories”. PhD thesis. Oxford University, 1978 (pp. 71, 75).
- [Car86] J. Cartmell. “Generalised algebraic theories and contextual categories”. In: *Ann. Pure Appl. Logic* 32 (1986), pp. 209–243. DOI: 10.1016/0168-0072(86)90053-9. URL: [https://doi.org/10.1016/0168-0072\(86\)90053-9](https://doi.org/10.1016/0168-0072(86)90053-9) (pp. 71, 75, 141, 142, 145).

- [CCD17] S. Castellan, P. Clairambault, and P. Dybjer. “Undecidability of Equality in the Free Locally Cartesian Closed Category”. In: *Logical Methods in Computer Science* 13.4 (2017). DOI: 10.23638/LMCS-13(4:22)2017 (p. 143).
- [CD11] P. Clairambault and P. Dybjer. “The Biequivalence of Locally Cartesian Closed Categories and Martin-Löf Type Theories”. In: *Typed Lambda Calculi and Applications*. Ed. by L. Ong. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 91–106. ISBN: 978-3-642-21691-6. DOI: 10.1007/978-3-642-21691-6_10. URL: https://doi.org/10.1007/978-3-642-21691-6_10 (p. 83).
- [CH19] E. Cavallo and R. Harper. “Parametric Cubical Type Theory”. In: *CoRR* abs/1901.00489 (2019). arXiv: 1901.00489 (p. 11).
- [CH20] E. Cavallo and R. Harper. “Internal Parametricity for Cubical Type Theory”. In: *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*. 2020, 13:1–13:17. DOI: 10.4230/LIPIcs.CSL.2020.13. URL: <https://doi.org/10.4230/LIPIcs.CSL.2020.13> (pp. 182–185, 298).
- [CH88] T. Coquand and G. Huet. “The calculus of constructions”. In: *Information and Computation* 76.2 (1988), pp. 95–120 (p. 78).
- [Che12] J. Cheney. “A dependent nominal type theory”. In: *Log. Methods Comput. Sci.* 8.1 (2012). DOI: 10.2168/LMCS-8(1:8)2012. URL: [https://doi.org/10.2168/LMCS-8\(1:8\)2012](https://doi.org/10.2168/LMCS-8(1:8)2012) (p. 183).
- [CHM18] T. Coquand, S. Huber, and A. Mörtberg. “On Higher Inductive Types in Cubical Type Theory”. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. Ed. by A. Dawar and E. Grädel. ACM, 2018, pp. 255–264. DOI: 10.1145/3209108.3209197. URL: <https://doi.org/10.1145/3209108.3209197> (p. 220).
- [Clo+16] R. Clouston, A. Bizjak, H. B. Grathwohl, and L. Birkedal. “The Guarded Lambda-Calculus: Programming and Reasoning with Guarded Recursion for Coinductive Types”. In: *Logical Methods in Computer Science* 12.3 (2016). DOI: 10.2168/LMCS-12(3:7)2016. URL: [https://doi.org/10.2168/LMCS-12\(3:7\)2016](https://doi.org/10.2168/LMCS-12(3:7)2016) (pp. 145, 146, 149).
- [Clo18] R. Clouston. “Fitch-Style Modal Lambda Calculi”. In: *Foundations of Software Science and Computation Structures*. Ed. by C. Baier and U. Dal Lago. Cham: Springer International Publishing, 2018, pp. 258–275. ISBN: 978-3-319-89366-2 (pp. 131, 153).

- [CMS20] E. Cavallo, A. Mörtberg, and A. W. Swan. “Unifying Cubical Models of Univalent Type Theory”. In: *Computer Science Logic (CSL 2020)*. Ed. by M. Fernández and A. Muscholl. Vol. 152. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 14:1–14:17. ISBN: 978-3-95977-132-0. DOI: 10.4230/LIPIcs.CSL.2020.14. URL: <https://drops.dagstuhl.de/opus/volltexte/2020/11657> (pp. 69, 182).
- [CND19] J. Ceulemans, A. Nuyts, and D. Devriese. “Reasoning about Effect Parametricity Using Dependent Types”. In: *Type-Driven Development (TyDe)*. 2019. URL: <http://tydeworkshop.org/2019-abstracts/paper11.pdf> (p. 290).
- [Coh+17] C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. “Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom”. In: *FLAP 4.10* (2017), pp. 3127–3170. URL: <http://www.cse.chalmers.se/~simonhu/papers/cubicaltt.pdf> (pp. 7, 9–11, 42, 69, 70, 155–158, 160, 165, 167, 169, 182, 184, 185, 187, 189, 206, 210, 224, 284, 285, 287).
- [Coq] T. Coq Development Team. *Coq*. <http://coq.inria.fr/> (p. 157).
- [Coq13] T. Coquand. *Presheaf model of type theory*. 2013. URL: <http://www.cse.chalmers.se/~coquand/presheaf.pdf> (p. 94).
- [Coq14] T. Coq Development Team. *The Coq proof assistant: reference manual*. v8.4, <https://coq.inria.fr/refman/>. 2014 (pp. 2, 157).
- [Coq18] T. Coquand. *Canonicity and normalization for Dependent Type Theory*. 2018. arXiv: 1810.09367. URL: <https://arxiv.org/abs/1810.09367> (pp. 130, 142).
- [dPR15] V. de Paiva and E. Ritter. “Fibrational Modal Type Theory”. In: *Proceedings of the Tenth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2015)*. 2015. DOI: 10.1016/j.entcs.2016.06.010 (pp. 129, 140, 152).
- [Dyb96] P. Dybjer. “Internal type theory”. In: *Types for Proofs and Programs: International Workshop, TYPES ’95 Torino, Italy, June 5–8, 1995 Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 120–134. DOI: 10.1007/3-540-61780-9_66 (pp. 9, 81, 82, 142).
- [Gar07] R. Garner. *Cofibrantly generated natural weak factorisation systems*. 2007. arXiv: math/0702290 [math.CT] (p. 64).

- [Gar09] R. Garner. “Understanding the Small Object Argument”. In: *Applied Categorical Structures* 17.3 (2009), pp. 247–285. ISSN: 1572-9095. DOI: 10.1007/s10485-008-9137-4. URL: <https://doi.org/10.1007/s10485-008-9137-4> (p. 64).
- [Gir64] J. Giraud. “Méthode de la descente.” French. In: *Bull. Soc. Math. Fr., Suppl., Mém.* 2 (1964), p. 115. ISSN: 0583-8665. DOI: 10.24033/msmf.2 (pp. 183, 224).
- [Gra+20a] D. Gratzer, A. Kavvos, A. Nuyts, and L. Birkedal. “Type Theory à la Mode”. Pre-print. 2020. URL: <https://anuyts.github.io/files/mtt-techreport.pdf> (pp. 13, 15, 122, 126, 130, 137, 141, 142, 156, 186, 187, 191, 196, 257, 271, 279, 290).
- [Gra+20b] D. Gratzer, G. A. Kavvos, A. Nuyts, and L. Birkedal. “Multimodal Dependent Type Theory”. In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by H. Hermanns, L. Zhang, N. Kobayashi, and D. Miller. ACM, 2020, pp. 492–506. DOI: 10.1145/3373718.3394736. URL: <https://doi.org/10.1145/3373718.3394736> (pp. 13, 14, 112, 125, 186, 187, 191, 285, 289).
- [Gra13] J. G. Granström. *Treatise on Intuitionistic Type Theory*. Springer Publishing Company, Incorporated, 2013. ISBN: 94-007-3639-8 (p. 141).
- [Gro+17] J. A. Gross, D. R. Licata, M. S. New, J. Paykin, M. Riley, M. Shulman, and F. Wellen. “Differential Cohesive Type Theory (Extended Abstract)”. In: *Extended abstracts for the Workshop "Homotopy Type Theory and Univalent Foundations"*. 2017. URL: <https://hott-uf.github.io/2017/abstracts/cohesivett.pdf> (p. 127).
- [GSB19a] D. Gratzer, J. Sterling, and L. Birkedal. “Implementing a Modal Dependent Type Theory”. In: *Proc. ACM Program. Lang.* (2019), 107:1–107:29. ISSN: 2475-1421. DOI: 10.1145/3341711 (pp. 13, 128, 130–132, 140, 141, 153).
- [GSB19b] D. Gratzer, J. Sterling, and L. Birkedal. *Normalization-by-Evaluation for Modal Dependent Type Theory*. Technical Report for the ICFP paper by the same name. 2019. URL: <https://jozefg.github.io/papers/2019-implementing-modal-dependent-type-theory-tech-report.pdf> (pp. 132, 141).
- [GT06] M. Grandis and W. Tholen. “Natural weak factorization systems”. In: *Archivum Mathematicum* 42.4 (2006). URL: <http://www.math.yorku.ca/~tholen/Natwfs.pdf> (pp. 51, 55, 59, 61).

- [Gua18] A. Guatto. “A Generalized Modality for Recursion”. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '18. ACM, 2018. DOI: 10.1145/3209108.3209148 (pp. 127, 152).
- [Has94a] R. Hasegawa. “Categorical Data Types in Parametric Polymorphism”. In: *Mathematical Structures in Computer Science* 4.1 (1994), pp. 71–109. DOI: 10.1017/S0960129500000372. URL: <https://doi.org/10.1017/S0960129500000372> (p. 261).
- [Has94b] R. Hasegawa. “Relational limits in general polymorphism”. In: *Publications of the Research Institute for Mathematical Sciences* 30 (1994), pp. 535–576 (p. 261).
- [Hof97] M. Hofmann. “Syntax and Semantics of Dependent Types”. In: *Semantics and Logics of Computation*. Cambridge University Press, 1997. Chap. 4, pp. 79–130 (pp. 9, 14, 99, 103, 142, 157, 182, 263).
- [How80] W. A. Howard. “The formulae-as-types notion of construction”. In: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Original manuscript from 1969. Academic press, 1980, pp. 479–490 (p. 2).
- [HS94] M. Hofmann and T. Streicher. “The Groupoid Model Refutes Uniqueness of Identity Proofs”. In: *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*. 1994, pp. 208–212. DOI: 10.1109/LICS.1994.316071. URL: <https://doi.org/10.1109/LICS.1994.316071> (p. 93).
- [HS97] M. Hofmann and T. Streicher. *Lifting Grothendieck Universes*. Unpublished note. 1997. URL: <https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf> (pp. 14, 99, 103, 106, 157, 182, 263).
- [Hub16] S. Huber. “Cubical Interpretations of Type Theory”. PhD thesis. Sweden: University of Gothenburg, 2016. URL: <http://www.cse.chalmers.se/~simonhu/misc/thesis.pdf> (pp. 9, 69, 182).
- [Hub19] S. Huber. “Canonicity for Cubical Type Theory”. In: *J. Autom. Reasoning* 63.2 (2019), pp. 173–210. DOI: 10.1007/s10817-018-9469-1. URL: <https://doi.org/10.1007/s10817-018-9469-1> (pp. 145, 285).
- [Jac99] B. Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics 141. North Holland, 1999 (p. 148).

- [JND19] K. Jacobs, A. Nuyts, and D. Devriese. “How to do proofs: practically proving properties about effectful programs’ results (functional pearl)”. In: *Proceedings of TyDe@ICFP 2019, Berlin, Germany, August 18, 2019*. Ed. by D. Darais and J. Gibbons. ACM, 2019, pp. 1–13. DOI: 10.1145/3331554.3342603. URL: <https://doi.org/10.1145/3331554.3342603> (p. 289).
- [Kav17] G. A. Kavvos. “Dual-context calculi for modal logic”. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2017, pp. 1–12. DOI: 10.1109/LICS.2017.8005089. arXiv: 1602.04860 (pp. 132, 134, 152).
- [Kav19] G. A. Kavvos. “Modalities, Cohesion, and Information Flow”. In: *Proceedings of the ACM on Programming Languages* 3.POPL (Jan. 2019), 20:1–20:29. DOI: 10.1145/3290333 (p. 127).
- [Kel+] C. Keller, M. Lasson, A. Anand, P. Roux, E. J. G. Arias, C. Cohen, and M. Sozeau. *ParamCoq*. 2012–2018. URL: <https://github.com/coq-community/paramcoq> (pp. 264, 286).
- [KHS19] A. Kaposi, S. Huber, and C. Sattler. “Gluing for Type Theory”. In: *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24–30, 2019, Dortmund, Germany*. Ed. by H. Geuvers. Vol. 131. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 25:1–25:19. DOI: 10.4230/LIPIcs.FSCD.2019.25. URL: <https://doi.org/10.4230/LIPIcs.FSCD.2019.25> (pp. 73, 130, 142).
- [KKA19] A. Kaposi, A. Kovács, and T. Altenkirch. “Constructing Quotient Inductive-inductive Types”. In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019), 2:1–2:24. ISSN: 2475-1421. DOI: 10.1145/3290315 (pp. 141, 142, 145).
- [KL12] C. Keller and M. Lasson. “Parametricity in an Impredicative Sort”. In: *Computer Science Logic (CSL’12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3–6, 2012, Fontainebleau, France*. 2012, pp. 381–395. DOI: 10.4230/LIPIcs.CSL.2012.381 (pp. 264, 286).
- [KLV12] C. Kapulkin, P. L. Lumsdaine, and V. Voevodsky. “The Simplicial Model of Univalent Foundations”. In: (2012). Preprint, <http://arxiv.org/abs/1211.2851> (pp. 3, 9, 11, 41, 182, 184).
- [Kun13] K. Kunen. *Set Theory*. Vol. 34. Studies in Logic. College Publications, 2013 (p. 20).
- [Law07] W. F. Lawvere. “Axiomatic Cohesion”. In: *Theory and Applications of Categories* 19 (June 2007), pp. 41–49 (p. 127).

- [Law63] F. W. Lawvere. “Functorial Semantics of Algebraic Theories”. PhD thesis. Columbia University, 1963 (p. 74).
- [Law70] W. Lawvere. “Quantifiers and Sheaves”. In: *Actes Congrès intern. math.* Vol. 1. 1970, pp. 329–334 (p. 36).
- [Lei91] D. Leivant. “Finitely stratified polymorphism”. In: *Information and Computation* 93.1 (1991), pp. 93–113. ISSN: 0890-5401. DOI: [http://dx.doi.org/10.1016/0890-5401\(91\)90053-5](http://dx.doi.org/10.1016/0890-5401(91)90053-5) (p. 279).
- [LH11] D. R. Licata and R. Harper. “2-Dimensional Directed Type Theory”. In: *Electr. Notes Theor. Comput. Sci.* 276 (2011), pp. 263–289. DOI: 10.1016/j.entcs.2011.09.026. URL: <https://doi.org/10.1016/j.entcs.2011.09.026> (pp. 3, 9, 111, 182).
- [Lic+18] D. R. Licata, I. Orton, A. M. Pitts, and B. Spitters. “Internal Universes in Models of Homotopy Type Theory”. In: *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK.* 2018, 22:1–22:17. DOI: 10.4230/LIPIcs.FSCD.2018.22. URL: <https://doi.org/10.4230/LIPIcs.FSCD.2018.22> (pp. 11, 12, 111, 127, 149, 151, 152, 185, 187, 206, 244, 251).
- [LS16] D. R. Licata and M. Shulman. “Adjoint Logic with a 2-Category of Modes”. In: *Logical Foundations of Computer Science: International Symposium, LFCS 2016, Deerfield Beach, FL, USA, January 4-7, 2016. Proceedings.* Springer International Publishing, 2016, pp. 219–235. ISBN: 978-3-319-27683-0. DOI: 10.1007/978-3-319-27683-0_16 (pp. 10, 51, 111, 128, 145, 149, 151, 283).
- [LSR17] D. R. Licata, M. Shulman, and M. Riley. “A Fibrational Framework for Substructural and Modal Logics”. In: *FSCD '17.* 2017, 25:1–25:22. DOI: 10.4230/LIPIcs.FSCD.2017.25. URL: <https://doi.org/10.4230/LIPIcs.FSCD.2017.25> (pp. 128–130, 142, 145, 149, 153, 312).
- [Mar75] P. Martin-Löf. “An intuitionistic theory of types: predicative part”. In: *Logic Colloquium '73, Proceedings of the Logic Colloquium.* Ed. by H. Rose and J. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. North-Holland, 1975, pp. 73–118 (p. 144).
- [Mar82] P. Martin-Löf. “Constructive mathematics and computer programming”. In: *Logic, Methodology and Philosophy of Science VI.* 1982, pp. 153–175 (p. 2).
- [Mar84] P. Martin-Löf. “Intuitionistic type theory”. In: *Studies in proof theory.* Bibliopolis, 1984 (p. 78).

- [Mar92] P. Martin-Löf. *Substitution calculus*. Notes from a lecture given in Göteborg, 1992 (p. 141).
- [Mar98] P. Martin-Löf. “An intuitionistic theory of types”. In: *Twenty-five years of constructive type theory*. Oxford University Press, 1998, pp. 127–172 (p. 2).
- [Miq01] A. Miquel. “The Implicit Calculus of Constructions”. In: *TLCA*. 2001, pp. 344–359. DOI: 10.1007/3-540-45413-6_27. URL: https://doi.org/10.1007/3-540-45413-6_27 (p. 111).
- [Møg14] R. E. Møgelberg. “A Type Theory for Productive Coprogramming via Guarded Recursion”. In: *Computer Science Logic (CSL) at Logic in Computer Science (LICS)*. CSL-LICS ’14. 2014. DOI: 10.1145/2603088.2603132 (p. 146).
- [Mog89] E. Moggi. “Computational lambda-calculus and monads”. In: *4th annual symposium on logic in computer science*. IEEE Press, 1989, pp. 14–23 (p. 4).
- [Mou16] G. Moulin. “Internalizing Parametricity”. PhD thesis. Sweden: Chalmers University of Technology, 2016. URL: publications.lib.chalmers.se/records/fulltext/235758/235758.pdf (pp. 9–11, 42, 184, 185, 187, 188, 197, 206–209, 258, 264, 280, 287, 298).
- [MP08] C. McBride and R. Paterson. “Applicative Programming with Effects”. In: *J. Funct. Program.* 18.1 (Jan. 2008), pp. 1–13. ISSN: 0956-7968. DOI: 10.1017/S0956796807006326. URL: <http://dx.doi.org/10.1017/S0956796807006326> (pp. 139, 146).
- [MS08] N. Mishra-Linger and T. Sheard. “Erasure and Polymorphism in Pure Type Systems”. In: *FOSSACS ’08*. 2008, pp. 350–364. DOI: 10.1007/978-3-540-78499-9_25. URL: https://doi.org/10.1007/978-3-540-78499-9_25 (pp. 11, 111).
- [MS92] J. C. Mitchell and A. Scedrov. “Notes on Scoping and Relators”. In: *Computer Science Logic (CSL) ’92, San Miniato, Italy, September 28 - October 2, 1992, Selected Papers*. Ed. by E. Börger, G. Jäger, H. K. Büning, S. Martini, and M. M. Richter. Vol. 702. Lecture Notes in Computer Science. Springer, 1992, pp. 352–378. DOI: 10.1007/3-540-56992-8_21. URL: https://doi.org/10.1007/3-540-56992-8_21 (p. 144).
- [Nak00] H. Nakano. “A Modality for Recursion”. In: *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*. IEEE Computer Society, 2000, pp. 255–266. DOI: 10.1109/LICS.2000.855774. URL: <https://doi.org/10.1109/LICS.2000.855774> (pp. 145, 146).

- [ND18a] A. Nuyts and D. Devriese. “Degrees of Relatedness: A Unified Framework for Parametricity, Irrelevance, Ad Hoc Polymorphism, Intersections, Unions and Algebra in Dependent Type Theory”. In: *Logic in Computer Science (LICS) 2018, Oxford, UK, July 09-12, 2018*. 2018, pp. 779–788. DOI: 10.1145/3209108.3209119. URL: <https://doi.org/10.1145/3209108.3209119> (pp. 7, 11, 42, 54, 111, 126, 127, 130, 133, 152, 182, 183, 257, 258, 270, 274, 275, 281–287, 289, 314).
- [ND18b] A. Nuyts and D. Devriese. “Internalizing Presheaf Semantics: Charting the Design Space”. In: *Workshop on Homotopy Type Theory / Univalent Foundations*. 2018. URL: https://hott-uf.github.io/2018/abstracts/HotTUUF18_paper_1.pdf (pp. 11, 184, 185, 187, 206, 210, 211, 290, 302).
- [ND19a] A. Nuyts and D. Devriese. “Dependable Atomicity in Type Theory”. In: *TYPES*. 2019 (pp. 188, 290).
- [ND19b] A. Nuyts and D. Devriese. “Menkar: Towards a Multimode Presheaf Proof Assistant”. In: *TYPES*. 2019 (pp. 13, 111, 290).
- [ND20] A. Nuyts and D. Devriese. “Transpension: The Right Adjoint to the Pi-Type”. Pre-print. 2020. URL: <https://anuyts.github.io/files/transpension-paper.pdf> (pp. 181, 289).
- [nLa20a] nLab authors. *2-category equipped with proarrows*. Revision 32. Apr. 2020. URL: <http://ncatlab.org/nlab/show/2-category%20equipped%20with%20proarrows> (p. 284).
- [nLa20b] nLab authors. *Conduché functor*. Revision 20. Apr. 2020. URL: <http://ncatlab.org/nlab/show/Conduch%C3%A9%20functor> (p. 224).
- [nLa20c] nLab authors. *Grothendieck universe*. Revision 52. Apr. 2020. URL: <http://ncatlab.org/nlab/show/Grothendieck%20universe> (p. 20).
- [nLa20d] nLab authors. *Lawvere theory*. Revision 72. Apr. 2020. URL: <http://ncatlab.org/nlab/show/Lawvere%20theory> (p. 74).
- [nLa20e] nLab authors. *model category*. Revision 81. Apr. 2020. URL: <http://ncatlab.org/nlab/show/model%20category> (p. 56).
- [nLa20f] nLab authors. *orthogonal factorization system*. Revision 35. Mar. 2020. URL: <http://ncatlab.org/nlab/show/orthogonal%20factorization%20system> (p. 53).
- [nLa20g] nLab authors. *quasi-category*. Revision 69. Apr. 2020. URL: <http://ncatlab.org/nlab/show/quasi-category> (p. 9).

- [nLa20h] nLab authors. *transfinite construction of free algebras*. Revision 12. Mar. 2020. URL: <http://ncatlab.org/nlab/show/transfinite%20construction%20of%20free%20algebras> (p. 34).
- [nLab] nLab authors. *HomePage*. URL: <http://ncatlab.org/nlab/show/HomePage> (p. 19).
- [Nor09] U. Norell. “Dependently typed programming in Agda”. In: *Advanced Functional Programming*. Springer, 2009, pp. 230–266 (pp. 2, 157).
- [Nor19] P. R. North. “Towards a Directed Homotopy Type Theory”. In: *Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2019, London, UK, June 4-7, 2019* (2019), pp. 223–239. DOI: 10.1016/j.entcs.2019.09.012. URL: <https://doi.org/10.1016/j.entcs.2019.09.012> (pp. 3, 182).
- [NPP08] A. Nanevski, F. Pfenning, and B. Pientka. “Contextual modal type theory”. In: *ACM Transactions Computational Logic* 9 (June 2008). DOI: 10.1145/1352582.1352591 (p. 152).
- [Nuy15] A. Nuyts. “Towards a Directed Homotopy Type Theory based on 4 Kinds of Variance”. Master’s thesis. Belgium: KU Leuven, 2015. URL: <https://anuyts.github.io/files/mathesis.pdf> (pp. 3, 8, 290).
- [Nuy17] A. Nuyts. *A Model of Parametric Dependent Type Theory in Bridge/Path Cubical Sets*. Tech. rep. Subsumed in [Nuy18a]. Belgium: KU Leuven, 2017. URL: <https://arxiv.org/abs/1706.04383> (pp. 9, 10, 12, 43, 51, 112, 114, 116, 120, 122, 155, 156, 218, 257, 258, 290).
- [Nuy18a] A. Nuyts. “Presheaf Models of Relational Modalities in Dependent Type Theory”. In: *CoRR* abs/1805.08684 (2018). arXiv: 1805.08684 (pp. 37, 42, 54, 81, 82, 99, 112, 156, 192, 214, 234, 257, 272, 278, 290, 331).
- [Nuy18b] A. Nuyts. “Robust Notions of Contextual Fibrancy”. In: *Workshop on Homotopy Type Theory / Univalent Foundations*. 2018. URL: https://hott-uf.github.io/2018/abstracts/HoTTUF18_paper_2.pdf (pp. 12, 69, 183, 214, 216, 221, 231, 234, 235, 290).
- [Nuy19] A. Nuyts. *Menkar*. <https://github.com/anuyts/menkar>. 2019 (pp. 13, 278, 290).
- [Nuy20] A. Nuyts. “The Transpension Type: Technical Report”. Pre-print. 2020. URL: <https://anuyts.github.io/files/transpension-techreport.pdf> (pp. 11, 20, 181, 187, 189, 191–193, 197, 204, 206, 252, 290).

- [NVD17a] A. Nuyts, A. Vezzosi, and D. Devriese. “Parametric quantifiers for dependent type theory”. In: *PACMPL* 1.ICFP (2017), 32:1–32:29. DOI: 10.1145/3110276. URL: <http://doi.acm.org/10.1145/3110276> (pp. 9–11, 43, 54, 111, 127, 133, 152, 155, 156, 158, 169, 181–185, 187, 206, 210, 257, 258, 265, 267, 269, 271, 272, 274, 280, 281, 284, 285, 289, 313).
- [NVD17b] A. Nuyts, A. Vezzosi, and D. Devriese. “Parametric quantifiers for dependent types”. In: *TYPES*. 2017. URL: <https://lirias.kuleuven.be/1656707> (p. 290).
- [OP18] I. Orton and A. M. Pitts. “Axioms for Modelling Cubical Type Theory in a Topos”. In: *Logical Methods in Computer Science* 14.4 (2018). DOI: 10.23638/LMCS-14(4:23)2018. URL: [https://doi.org/10.23638/LMCS-14\(4:23\)2018](https://doi.org/10.23638/LMCS-14(4:23)2018) (pp. 69, 155, 156, 158, 169, 172, 182, 185–187, 206, 210, 251).
- [Ort18] I. Orton. “Cubical Models of Homotopy Type Theory - An Internal Approach”. PhD thesis. University of Cambridge, 2018 (pp. 69, 155, 156, 169, 172, 182, 185, 187, 251).
- [PD01] F. Pfenning and R. Davies. “A judgmental reconstruction of modal logic”. In: *Mathematical Structures in Computer Science* 11.4 (2001), pp. 511–540. DOI: 10.1017/S0960129501003322 (pp. 111, 131, 134, 140, 141, 152).
- [Pfe01] F. Pfenning. “Intensionality, Extensionality, and Proof Irrelevance in Modal Type Theory”. In: *LICS '01*. 2001, pp. 221–230. DOI: 10.1109/LICS.2001.932499. URL: <https://doi.org/10.1109/LICS.2001.932499> (pp. 11, 13, 111, 127, 129, 133, 152).
- [Pie+19] B. Pientka, A. Abel, F. Ferreira, D. Thibodeau, and R. Zucchini. “A Type Theory for Defining Logics and Proofs”. In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2019 (p. 152).
- [Pit13] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Vol. 57. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013. ISBN: 9781107017788 (p. 194).
- [Pit14] A. Pitts. “Nominal Sets and Dependent Type Theory”. In: *TYPES*. 2014. URL: <https://www.irif.fr/~letouzey/types2014/slides-inv3.pdf> (pp. 183, 188, 211).
- [PK19] G. Pinyo and N. Kraus. “From Cubes to Twisted Cubes via Graph Morphisms in Type Theory”. In: *CoRR* abs/1902.10820 (2019). arXiv: 1902.10820. URL: <http://arxiv.org/abs/1902.10820> (pp. 195, 251, 284).

- [PMD15] A. M. Pitts, J. Matthiesen, and J. Derikx. “A Dependent Type Theory with Abstractable Names”. In: *Electronic Notes in Theoretical Computer Science* 312 (2015). Ninth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2014), pp. 19–50. ISSN: 1571-0661. DOI: <https://doi.org/10.1016/j.entcs.2015.04.003>. URL: <http://www.sciencedirect.com/science/article/pii/S1571066115000079> (pp. 183, 185, 194).
- [Pra65] D. Prawitz. *Natural Deduction: a proof-theoretical study*. Almqvist and Wiksell, 1965 (p. 141).
- [Ree03] J. Reed. “Extending Higher-Order Unification to Support Proof Irrelevance”. In: *TPHOLs 2003*. 2003, pp. 238–252. DOI: 10.1007/10930755_16. URL: https://doi.org/10.1007/10930755_16 (p. 111).
- [Ree09] J. Reed. “A Judgmental Deconstruction of Modal Logic”. In: (2009). Manuscript. URL: <http://www.cs.cmu.edu/~jcreed/papers/jdml.pdf> (pp. 128, 145, 149).
- [Rey83] J. C. Reynolds. “Types, Abstraction and Parametric Polymorphism.” In: *IFIP Congress*. 1983, pp. 513–523 (pp. 9, 10, 15, 183, 258, 259, 279).
- [Rie08] E. Riehl. *Factorization Systems*. Lecture notes. 2008. URL: <http://www.math.jhu.edu/~eriehl/factorization.pdf> (pp. 51, 56).
- [RR94] E. P. Robinson and G. Rosolini. “Reflexive graphs and parametric polymorphism”. In: *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science*. July 1994, pp. 364–371. DOI: 10.1109/LICS.1994.316053 (p. 261).
- [RS17] E. Riehl and M. Shulman. “A type theory for synthetic ∞ -categories”. In: *ArXiv e-prints* (May 2017). arXiv: 1705.07442 [math.CT] (pp. 3, 12, 168, 182–185).
- [RSS20] E. Rijke, M. Shulman, and B. Spitters. “Modalities in homotopy type theory”. In: *Logical Methods in Computer Science* Volume 16, Issue 1 (2020). URL: <https://lmcs.episciences.org/6015> (p. 127).
- [SAG19] J. Sterling, C. Angiuli, and D. Gratzer. “Cubical Syntax for Reflection-Free Extensional Equality”. In: *CoRR* abs/1904.08562 (2019). arXiv: 1904.08562. URL: <http://arxiv.org/abs/1904.08562> (pp. 93, 285, 287).
- [Sch13] U. Schreiber. “Differential cohomology in a cohesive infinity-topos”. In: *arXiv e-prints*, arXiv:1310.7930 (Oct. 2013), arXiv:1310.7930. arXiv: 1310.7930 [math-ph] (p. 127).

- [Seg68] G. Segal. “Classifying spaces and spectral sequences”. In: *Publications mathématiques de l’IHÉS* 34.1 (Jan. 1968), pp. 105–112. ISSN: 1618-1913. DOI: 10.1007/BF02684591. URL: <https://doi.org/10.1007/BF02684591> (p. 69).
- [Shu14] M. Shulman. “Univalence for inverse diagrams and homotopy canonicity”. In: *Mathematical Structures in Computer Science* (2014). DOI: 10.1017/s0960129514000565 (pp. 130, 142).
- [Shu18] M. Shulman. “Brouwer’s fixed-point theorem in real-cohesive homotopy type theory”. In: *Mathematical Structures in Computer Science* 28.6 (2018), pp. 856–941. DOI: 10.1017/S0960129517000147. URL: <https://doi.org/10.1017/S0960129517000147> (pp. 127, 129, 134, 140, 141, 145, 149, 151, 152).
- [SS14] U. Schreiber and M. Shulman. “Quantum Gauge Field Theory in Cohesive Homotopy Type Theory”. In: *Proceedings 9th Workshop on Quantum Physics and Logic Brussels, Belgium, 10-12 October 2012*. 2014, pp. 109–126. DOI: 10.4204/EPTCS.158.8 (p. 127).
- [SS86] S. Schanuel and R. Street. “The free adjunction”. In: *Cahiers de topologie et géométrie différentielle catégoriques* 27.1 (1986), pp. 81–83 (pp. 129, 150).
- [Sta23] T. Stacks Project Authors. *Stacks Project*. <http://stacks.math.columbia.edu>. Tag 0023. 2020 (p. 48).
- [StaVC] T. Stacks Project Authors. *Stacks Project*. <http://stacks.math.columbia.edu>. Tags 00VC and 00XF. 2019 (pp. 49, 196).
- [Tai67] W. W. Tait. “Intensional Interpretations of Functionals of Finite Type I”. In: *The Journal of Symbolic Logic* 32.2 (1967), pp. 198–212. ISSN: 00224812. URL: <http://www.jstor.org/stable/2271658> (p. 144).
- [Tak01] I. Takeuti. *The Theory of Parametricity in Lambda Cube*. Tech. rep. 1217. Kyoto University, 2001 (p. 264).
- [Tsa+19] S. Tsampas, A. Nuyts, D. Devriese, and F. Piessens. “Categorical Contextual Reasoning”. In: *Applied Category Theory (ACT)*. 2019 (p. 289).
- [Tsa+20] S. Tsampas, A. Nuyts, D. Devriese, and F. Piessens. “A categorical approach to secure compilation”. In: *Coalgebraic Methods in Computer Science (CMCS)*. 2020 (p. 289).
- [Uni13] T. Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. IAS: <http://homotopytypetheory.org/book>, 2013 (pp. 2, 3, 6, 7, 13, 41, 93, 139, 157, 182, 184, 186, 188, 190, 204, 281).

- [VMA19] A. Vezzosi, A. Mörtberg, and A. Abel. “Cubical Agda: a dependently typed programming language with univalence and higher inductive types”. In: *PACMPL* 3.ICFP (2019), 87:1–87:29. DOI: 10.1145/3341691. URL: <https://doi.org/10.1145/3341691> (pp. 156, 158, 212).
- [Vou12] A. Voutas. “The basic theory of monads and their connection to universal algebra”. In: (2012) (p. 31).
- [Wad89] P. Wadler. “Theorems for Free!” In: *FPCA '89*. Imperial College, London, United Kingdom: ACM, 1989, pp. 347–359. ISBN: 0-89791-328-0. DOI: 10.1145/99370.99404 (pp. 9, 282).
- [WL20] M. Z. Weaver and D. R. Licata. “A Constructive Model of Directed Univalence in Bicubical Sets”. In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by H. Hermanns, L. Zhang, N. Kobayashi, and D. Miller. ACM, 2020, pp. 915–928. DOI: 10.1145/3373718.3394794. URL: <https://doi.org/10.1145/3373718.3394794> (pp. 3, 7, 12, 182, 184).
- [XE16] C. Xu and M. Escardó. *Universes in sheaf models*. Unpublished note. 2016. URL: https://cj-xu.github.io/notes/sheaf_universe.pdf (p. 250).
- [Zwa19] C. Zwanziger. “Natural Model Semantics for Comonadic and Adjoint Type Theory: Extended Abstract”. In: *Preproceedings of Applied Category Theory Conference 2019*. 2019 (pp. 145, 152).

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
IMEC-DISTRINET
Celestijnenlaan 200A box 2402
B-3001 Leuven
<https://distrinet.cs.kuleuven.be>

