A Generalized Algebraic Theory of Directed Equality

by

Jacob Neumann

A thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy

School of Computer Science
University of Nottingham
June 2025

Abstract

We develop a directed type theory capable of synthetic reasoning about 1-categories, with straightforward semantics in the category of categories. All our semantic notions are defined as generalized algebraic theories, permitting modular reasoning about different type theories and ensuring the existence of initial syntax models.

We define the category model of type theory—the directed analogue of Hofmann and Steicher's groupoid model—where contexts are interpreted as categories and types as (split opfibrant) displayed categories. Adapting the groupoid model semantics of key type formers, (dependent) functions and identity types in particular, requires us to adopt a modal typing discipline to track the variances of terms. We define a succession of model notions—polarized categories with families, neutral-polarized categories with families, and directed categories with families—to abstractly capture more and more of the type theory of the category model; this includes an axiomatization of how the groupoid model's undirected type theory is situated inside the category model's directed type theory.

The symmetric (i.e. invertible) identity types of the groupoid model become directed hom-types of the category model; we can show by metatheoretic argument that our polarized typing discipline prevents invertibility of hom-types from being provable in our directed type theory. Each type therefore has the structure of a synthetic category and each function a synthetic functor; this is all proved within the theory using the eliminator for hom-types, directed path induction. We expound a new style of category-theoretic reasoning appropriate to this setting, where the universal mapping properties of standard category theory are all phrased as principles of induction.

To support these developments, we outline generalized algebra, the mathematical discipline concerned with generalized algebraic theories. In particular, we highlight the construction of an initial algebra for every GAT (which is what guarantees a syntax model for the notions of model we use) and the fact that every GAT gives rise to a "concrete category with families", a model of type theory whose contexts are algebras and types are displayed algebras. The groupoid and category models can be viewed as a modification of the concrete CwFs of groupoids and categories, respectively, which ensures the fibrancy of their types.

Acknowledgments

When I was younger, I first heard the phrase, "count your blessings." Taking it to heart, I went off to university to learn how to count past infinity. There are countless people for me to thank for making this thesis possible; let me try to name some of them.

I'm so fortunate to have spent the past four years in the Functional Programming Lab at the University of Nottingham, and be a part of the amazing group there. First and foremost, I must thank my advisor, Thorsten Altenkirch, for his constant mentorship, for endless questions (many of them insightful), and for hundreds of conversations about every aspect of this thesis (and more). Many thanks also to Graham Hutton for all his encouragement and for making the FP Lab such an excellent environment for us all; and thanks to the other faculty of the Lab—Ulrik Buchholtz, Nicolai Kraus, and Dan Marsden—for all the ways they've influenced my thinking. I'm also tremendously grateful for Tom de Jong; Tom is someone we all look up to (both physically and aspirationally), and someone I could always trust to provide honest, helpful feedback. And all my love to my fellow (former) PhD students in the Lab: Josh Chen and Stefania Damato were there with me the entire time, and I had some years with Zac Garby, Brandon Hewer, Aref Mohammadzadeh, Stiéphen Pradal, Johannes Schipp von Branitz, Filippo Sestini, Zhili Tian, Mark Williams, and Sky Wilshaw.

Two very formative experiences for the development of this thesis were my short-term scientific missions[1] to Budapest and Tallinn. Many thanks to my hosts, Ambrus Kaposi and Niccolò Veltri, respectively, and to all the students and researchers I worked with on these trips (especially Andrea Laretto) for some excellent conversations. And thanks to Tarmo Uustalu for finding a place for me in Reykjavik (and for reminding me which part of this thesis is most important!).

Some other folks (not otherwise mentioned) whom I've been lucky to know these past four years, and who influenced this thesis include:[2] Cass Alexandru, Carlo Angiuli, Jonathan Arnoult, Fredrik Bakke, Viktor Bense, Rafaël Bocquet, Thea Brick, Evan Cavallo, Fernando Chu, Roy Crole, Bruno da Rocha Paiva, Martin Escardó, Eric Finster, Thiago Felicissimo, Fredrik Nordvall Forsberg, James Gallicchio, Daniel Gratzer, Harrison Grodin, Chris Grossack, Bob Harper, Philipp Joram, Moana Jubert, Thomas Lamiaux, Paul Levy, Axel Ljungström, Rasmus Møgelberg Vincent Moreau, Paige North, Andreas Nuyts, Daniël Otten, Andor Pénzes, Chris Purdy, Till Rampe, Philip Scott, Artem

[1] This publication is based upon work from COST Action EuroProofNet, supported by COST (European Cooperation in Science and Technology, www.cost.eu).

[2] I'm sure I forgot some names!

Shinkarov, Matteo Spadetto, Sam Speight, Ayberk Tosun, Sam Toth, Taichi Uemura, Phil Wadler, Jonathan Weinberger, Kobe Wullaert, Szumi Xie, Errol Yuksel, Max Zeuner, and Colin Zwanziger.

I would not be the researcher (or person) I am, had it not been for the formative time I spent in the Carnegie Mellon University Philosophy Department. Many of the core topics of this thesis are things I learned at CMU: Jonas Frey taught me what a 'CwF' is; I learned HoTT from Egbert Rijke;[3] I learned my first proof assistant, LEAN, from Jeremy Avigad;[4] I didn't really understand the Yoneda Lemma until I worked as Mathieu Anel's teaching assistant; Steve Awodey taught me categorical semantics of type theory;[5] and Steve, Adam Bjorndahl,[6] Kevin Kelly, and Wilfried Sieg taught me everything I knew about mathematical logic. And there are countless others who influenced and shaped me, for whom I'll be forever grateful.

I was also profoundly shaped by the community surrounding CMU's 15-150 *Principles of Functional Programming* course: my time as a 150 TA was absolutely fundamental to my whole approach to teaching, and the two times I taught the course myself were among the greatest and most exciting challenges of my career so far. I'll always be thankful for Jonathan Merrin, Dilsun Kaynar, and Vijay Ramamurthy for seeing potential in me (even when I didn't see it in myself); for the excellent professors I worked for (Dilsun, Steve Brookes, Mike Erdmann, and Frank Pfenning); the dozens of amazing TAs I got to work with; my coinstructors in the offshoot student-taught course *Hype for Types*; and our thousands of students. *Functions are values*, forever.

Many thanks also to everyone who nurtured my nascent love of mathematics and programming when I was in high school, especially Mr. Esparza, Mr. Fornstrom, Dr. Mocanasu, Dr. Schaeffer Fry, and all the YouTube channels I used to teach myself calculus.

Thanks to anyone who's ever said a kind word about my ideas, my teaching, my art, my photography, my videos.

Lots of love to my extended family—in Milwaukee, Missouri, Pittsburgh, San Diego, and elsewhere—thank you for making me who I am. And love to all the friends I've made along the way, and all the communities I've been lucky to be a part of.

And, most of all, thanks to my parents, my sister, and my brother-in-law[7] for their constant love and support. I couldn't have done it without you!

[3] And I am proud to have been in the zeroth generation of guinea pigs for his textbook, [Rij22].

[4] And I remember him being especially patient in explaining Σ-types to me.

[5] And I'll always be grateful for Steve's support in 2021 when my PhD studentship was questionable.

[6] To whom I also owe tremendous thanks for supervising my master's thesis, for getting me started as a researcher in the first place, and for finding work for me in Fall 2020.

[7] And to Fabio, the wisest and most benevolent chihuahua to ever grace the Earth

Contents

Abstract i

Acknowledgments ii

Contents iv

0 Introduction 1
- 0.1 Motivation . . . 1
- 0.2 Research Questions . . . 4
- 0.3 Related Work . . . 9
- 0.4 Contribution . . . 14
- 0.5 Metatheory and Notation . . . 16

List of GATs & Structures 18

1 Type Theory as a Generalized Algebraic Theory 22
- 1.1 Specifying Structures as GATs . . . 25
- 1.2 The Semantics of Type Theory as a GAT . . . 36
  - 1.2.1 Categories with Families . . . 37
  - 1.2.2 Type Formers . . . 44
- 1.3 A Signature Language for GATs . . . 54
  - 1.3.1 Introducing GATs . . . 55
  - 1.3.2 Eliminating GATs . . . 62
  - 1.3.3 Initial Algebras . . . 65
  - 1.3.4 Concrete CwFs . . . 68
- 1.4 Fibrancy and the Looking-Glass Question . . . 75

2 The Polarity Calculus 86
- 2.1 Polarity Structure of the Preorder and Category Models . . . 87
- 2.2 Theory of Polarized CwFs . . . 95
- 2.3 Neutrality . . . 107
  - 2.3.1 Phase Zero . . . 109
  - 2.3.2 Phase One . . . 111
  - 2.3.3 Phase Two+ . . . 124

3 Directed Type Theory 131
- 3.1 Directed Equality Types . . . 133
- 3.2 Observations and Universes . . . 149

3.2.1 Opposites . . . 150
3.2.2 Σ-Types . . . 153
3.2.3 Universes . . . 157

4 Synthetic-Inductive Category Theory 165
4.1 An Introduction to Informal (1,1)-Directed Type Theory . . . 165
4.1.1 Neutral-Polarized Type Theory . . . 166
4.1.2 Directed Type Theory . . . 169
4.1.3 Type Formers . . . 172
4.1.4 Inductive Category Theory . . . 174
4.2 Binary (Co)Products . . . 175
4.2.1 Binary Products—Informal . . . 175
4.2.2 Binary Products—Formal . . . 178
4.2.3 Binary Coproducts . . . 182
4.3 Pullbacks and Pushouts . . . 186
4.4 Functors and Adjoints . . . 193
4.4.1 Functors and Natural Transformations—Informal . . . 193
4.4.2 Functors and Natural Transformations—Formal . . . 197
4.4.3 Adjoints—Informal . . . 203

5 Conclusion 206
5.1 Research Questions Reprised . . . 206
5.2 Some Suggestions for Future Work . . . 209

Bibliography 211

A Generalized Algebraic Theories 219
A.1 Basic structures . . . 219
A.1.1 Sets . . . 219
A.1.2 Pointed Sets . . . 219
A.1.3 Bipointed Sets . . . 220
A.1.4 Natural Number Algebras . . . 220
A.1.5 Even-Odd Algebras . . . 221
A.1.6 Monoids . . . 221
A.1.7 Groups . . . 222
A.2 Quiver-like structures . . . 222
A.2.1 Quivers . . . 222
A.2.2 Reflexive Quivers . . . 223
A.2.3 Preorders . . . 223
A.2.4 Setoids . . . 224
A.2.5 Categories . . . 225
A.2.6 Groupoids . . . 226
A.3 Models of Type Theory . . . 228
A.3.1 Categories with Families (CwFs) . . . 228
A.3.2 CwFs /w unit type . . . 230
A.3.3 CwFs /w bool type . . . 231
A.3.4 CwFs /w Π-type . . . 232

A.4 Models of Polarized and Directed Type Theory 233
A.4.1 Polarized Categories with Families (PCwFs) 233

Chapter 0

Introduction

0.1 Motivation

A pivotal moment in the history of category theory was Saunders Mac Lane's visit to France in the summer of 1954. It was likely during this time that Mac Lane had his now-legendary meeting at Gare du Nord with a young Nobuo Yoneda, and learned of the great lemma which now bears the latter's name [Le 16]. It was also when Mac Lane attended the Bourbaki congress in Murol, and tried to convince its members to embrace the (then new) language of category theory [Le 22]. As Bourbaki's internal newsletter, *La Tribu*, No. 34 [Bou54] relates, he was unsuccessful:

> "Frightened by the disorder of the discussions, some members had brought a world-renowned efficiency expert from Chicago. This one, armed with a hammer, tried hard and with good humor, but without much result. He quickly realized that it was useless, and turned, successfully this time, to photography"

In subsequent decades, Bourbaki persisted in avoiding explicit category-theoretic language, even as the content of subsequent volumes of the *Eléments* adopted ideas from category theory [Cor03, pp. 372–373].

Fundamentally, Bourbaki and Mac Lane had different views of what constituted a "mathematical theory": whether the essential content attached to each notion of "structure" is a notion of *isomorphism* or of *morphism*. For Bourbaki, *isomorphisms* were the fundamental consideration: quoting André Weil,[1]

> "About 1936 the collaborators of N. Bourbaki agreed to adopt the notion of structure, and its associated one of isomorphism, as a fundamental principle for the classification of mathematical theories."

To use category-theoretic parlance,[2] the Bourbaki view was (apparently) that the essence of a mathematical theory constitutes a *groupoid*—a universe of structures and the

[1] Quoted in [Cor03, p. 381].

[2] In case it wasn't clear on which side of the argument this author falls.

structural *iso*morphisms between them—and that homomorphisms were an auxiliary concern. From this viewpoint, the category-theoretic view—that the essence of a mathematical theory is rather a *category*, i.e. a universe of structures and structure-preserving *morphisms*[3]—was evidently perplexing: quoting Weil's letter to Chevalley,[4]

> "As you know, my honorable colleague Mac Lane claims that every notion of structure necessarily implies a notion of homomorphism, which consists in indicating for each data constituting the structure, those which behave covariantly and those which behave contravariantly...What do you think can be gained from this kind of considerations?"

Most contemporary mathematicians would not share Weil's skepticism here. Even if one is not working explicitly with category-theoretic tools, there is undoubtedly *something* to be gained by considering the structure-preserving morphism appropriate to whatever kind of "structure" one is studying; or, at the very least, the notion of "structure-preserving morphism" makes sense. We might regard this as a kind of **central dogma of category theory** for any kind of mathematical object one can define, there is a corresponding notion of morphism. That is, every notion of structure inherently gives rise to a category. As Eilenberg and Mac [EM45] advise us, the formulation of this morphism notion is part and parcel of defining the notion of "structure" itself:

> "whenever new abstract objects are constructed in a specified way out of given ones, it is advisable to regard the construction of the corresponding induced mappings on these new objects as an integral part of their definition."

But let us emphasize that the category-theoretic view is only asserting the *logical priority* of morphisms over isomorphisms, not that the latter be banished. Indeed, it is within the category-theoretic framework that isomorphisms obtain their proper treatment: as *invertible* morphisms. Only when we properly understand the *directed* notion of morphism can we fully understand the derived *symmetric* notion of isomorphism.

Viewed against this historical backdrop, there is a striking irony in the contemporary development of *intensional type theory*, specifically **homotopy type theory**: many of the top category theorists in the world have come together to produce a theory whose central notion—equivalence—is *symmetric*, a (significantly improved) version of Bourbaki-style isomorphism! There's good reason for this, of course: homotopy type theory (or "HoTT") and other related theories descend from the dependent type theory of Martin-Löf [Mar75; Mar82], which includes **identity types**: for any two terms t and t' of the same type, there is a type $\mathsf{Id}(t, t')$ whose terms (if there are any) are *identifications* of t and t', i.e. *proofs* or *witnesses* to the equality of t and t'. In this conception of 'identity'—a **constructive** and **proof-relevant** one—the assertion "t is equal to t'" is not a bare, lifeless judgment that merely is or isn't the case, but a *type of data* with its own internal structure. Homotopy type theory takes Martin-Löf Type Theory (MLTT) to its logical extreme by contemplating types whose identity types are themselves rich data structures, such as the circle type $\mathbb{S}^1$—which has only one element $\mathsf{base} : \mathbb{S}^1$, but whose identity type $\mathsf{Id}(\mathsf{base}, \mathsf{base})$ is equivalent to the group $\mathbb{Z}$ of integers. Similarly to how Eilenberg and Mac Lane described the construction of structure-preserving morphisms as an "integral part of (the) definition" of a structural concept, in HoTT we

[3] And that 'isomorphism' was merely a derived notion

[4] Quoted in [Cor03, p. 376].

regard the identity types as being defined alongside the type itself—the structure of the identity types corresponds to the structure of the type. Formally, this is manifested within the theory as a number of **extensionality principles** characterizing the identity types of the various type-constructors. Chief among these is Voevodsky's **univalence axiom**, which equates the identity type $\mathsf{Id}(X, Y)$ of two *types* X and Y with the type of equivalences (roughly speaking, isomorphisms) between X and Y. This is why it's necessary and fitting that HoTT provides a theory centered around a symmetric notion (equivalences) rather than a directed one: identity is naturally symmetric—from $p : \mathsf{Id}(t, t')$, obtain $p^{-1} : \mathsf{Id}(t', t)$—so identities between types X and Y logically couldn't correspond to, say, the type of functions $X \to Y$.

One important precursor to HoTT, upon whose example we'll largely base the present work, is the **groupoid model of type theory** devised by Hofmann and Streicher[HS95]. The type theory of the groupoid model takes the first step towards HoTT by allowing types like $\mathbb{S}^1$ whose identity types can contain many different elements, but truncates any "higher structure": the identity types *of the identity types* cannot contain multiple elements. More precisely, the principle known as Axiom K [Str93] or the **uniqueness of identity proofs** (UIP) for a given type A states that every identity type $\mathsf{Id}(t, t')$ between terms $t, t' : A$ has *at most one element*. The groupoid model was specifically designed to refute UIP in general—types like $\mathbb{S}^1$ serve as counterexamples—but UIP *does* hold for identity types themselves: given $p, q : \mathsf{Id}(t, t')$, there can be at most one term of type $\mathsf{Id}(p, q)$.

We take inspiration from a few particular aspects of the groupoid model.

- The groupoid model is a *model*, that is, a mathematical structure (specifically a *category with families*, or *CwF*) providing denotational semantics of the formal language of type theory. This provides a means to make *metatheoretic arguments*: since the groupoid model validates all the rules of MLTT but does *not* validate UIP, we can deduce that UIP is not provable from just the rules of MLTT.
- Though it's truncated nature does not permit it to model full HoTT and full univalence, it does have a truncated version. Namely, the groupoid model interprets a *universe of sets*, a type classifying all the *sets* (types that *do* validate UIP) in the theory. The truncated version of univalence—which Hofmann and Streicher call **universe extensionality**—is that the identity type $\mathsf{Id}(X, Y)$ between two sets corresponds to the type of *bijections* between X and Y.
- The groupoid model is so named because it is *built out of groupoids*. In order to interpret a type theory, i.e. fulfill all the different components of a CwF, the groupoid model must provide an interpretation for all the structural components of a type theory, specifically *contexts, substitutions, types, and terms*. The groupoid model uses groupoids for all these: contexts are groupoids, substitutions are groupoid functors, types are families of groupoids, and terms are sections of families of groupoids.
- The groupoid model deserves its name for another reason: its type theory provides a **synthetic theory** of groupoids. Every type A in this type theory has the structure of a groupoid: the objects of the groupoid A are the terms $t : A$ and the morphisms from t to t' are the terms of type $\mathsf{Id}(t, t')$. This is a category because identity is reflexive (for each t, obtain $\mathsf{refl}_t : \mathsf{Id}(t, t)$) and transitive (terms of type $\mathsf{Id}(t, t')$ and $\mathsf{Id}(t', t'')$ can be combined to get a term of type $\mathsf{Id}(t, t'')$); this is a

groupoid because of the aforementioned symmetry of identity: every "morphism" $p : \mathsf{Id}(t, t')$ has an inverse $p^{-1} : \mathsf{Id}(t', t)$. So any type which can be written down in this theory automatically has the structure of a groupoid, any function has the structure of a groupoid functor, and so on.

The combination of these last two points will be of special interest to us: it's a model built *out of groupoids*, whose type theory is a synthetic theory *of groupoids*—somehow the groupoids have gained the ability to talk about themselves!

So, returning to the point: what is the *non-symmetric* version of this story? Can we replace groupoids with categories, and consider a **category model** whose type theory furnishes a **synthetic category theory**? Is there a version of identity types which *isn't* symmetric, and whose univalence principle relates them not to equivalences/isomorphisms/bijections but to *functions*? These question are the impetus for **directed type theory**. Instead of considering identity types $\mathsf{Id}(t, t')$, directed type theory considers **hom-types** $\mathsf{Hom}(t, t')$, a **directed notion of equality**. Now, the reason that directed type theory didn't precede usual, *undirected* type theory is because it's much trickier to get right: in order to, as Weil says, "indicat(e) for each data constituting the structure, those which behave covariantly and those which behave contravariantly," is an exceedingly subtle art. A few proposals were put forward (e.g. [LH11; Nor19; Nuy15; ANvdW23]) for such a **calculus of polarity**, a modal typing discipline for explicitly tracking variances, but no consensus has emerged for how exactly to carry this out. Indeed, the most developed directed type theories to date—particularly ones that seek to give the directed analogue of all the higher structure of HoTT, i.e. provide a language for synthetic ∞-category theory—avoid having a polarity calculus at all,[5] and fashion a directed type theory using techniques from simplicial and cubical type theory. Among those that *do* adopt a polarity calculus, North [Nor19] is (to our knowledge) the only one who defines the category model, and doesn't develop adequate syntax to match all the features of the groupoid model (dependent types, the universe of sets, the universe extensionality/univalence principle) or conduct synthetic category theory. So the task of defining the category model and developing it enough to do synthetic category theory remains unfinished. That is the purpose of the present work.

0.2 Research Questions

We organize our investigation around the following questions, which approximately correspond to the chapters of this thesis.

[5] From [RS17, p. 2]:

> "Moreover, interpreting types directly as (higher) categories runs into various problems, such as the fact that ...there are numerous different kinds of "fibrations" given the various possible functorialities and dimensions of categories appearing as fibers.
>
> There is no reason in principle to think these problems insurmountable, and many possible solutions have been proposed. However, in this paper we pursue a somewhat indirect route to a synthetic theory of higher categories"

Research Question 1. What is an appropriate semantics-driven methodology for developing directed type theory and synthetic category theory?

We take the main task of the present work—the explication of the category model—as an opportunity to perform a deep methodological reflection on the semantics of type theory. In designing a formal language towards a particular purpose (in our case, synthetic category theory), one finds oneself in a precarious situation: there is an ever-present temptation to "just add more axioms" to prove whatever one needs, but, as demonstrated by the issue of *consistency* that bedevils mathematical logic, reckless postulation can get one in serious trouble. A semantics-driven approach, where new postulates must first be checked against a desired model (or class of models), addresses the consistency concern but opens up a new consideration: *initiality*. In many frameworks for the semantics of type theory, it is a difficult task to reify the syntax of the type theory as a *term model* and prove that this term model is the *initial* object in the category of models. So long as the initiality conjecture remains open for a given theory and class of models, we cannot be assured that the syntax and semantics interact in the way we expect.

We therefore highlight the theory of **generalized algebraic theories**, or GATs, as a robust approach for studying type theories and their semantics. The crucial benefit of GATs is that each GAT automatically comes equipped with an initial model. Therefore, if we can express our notion of 'model of type theory' as a GAT—as we can, in the aforementioned notion of CwF—then the initiality conjecture is already solved. Moreover, we can continue to extend the notion of CwF to include more type theoretic constructs and, so long as these extensions still constitute a GAT, the initiality claim is preserved. In particular, this will allow us to extend the GAT of CwFs to a GAT of *directed* CwFs, i.e. models of directed type theory. Since we adopt a semantics-driven approach, only adding to the GAT of directed CwFs those axioms which we can validate in the category model, we have a constant guarantee that we are justified in speaking of the "syntax of directed type theory" as the initial directed CwF.

Some intriguing methodological considerations also come into play when conducting synthetic category theory in the syntax of directed type theory. We will want to introduce category-theoretic notions (like 'product', 'exponential', 'adjunction', etc.) in a format better suited to our language, but reassure ourselves that these concepts coincide to their usual meaning. We'll adopt a two-step approach for doing so: first, we'll demonstrate *synthetically* (that is, within the synthetic category theory) that the usual *universal mapping property* of these concepts can be derived from our formulation, and thus we know our formulation is *at least as strong* as the usual one. Secondly, we'll demonstrate *analytically* that our notion is not any stronger: we'll interpret (by way of the initiality of the syntax in the category of directed CwFs) the statement of our notion into the empty context of the category model—where the types are *literally* categories and the terms are *literally* objects, and show that the usual notion entails ours. This is a unique methodology, unlocked by the category model's auto-synthetic potential: it's only possible to perform this analysis because we're doing synthetic category theory, interpreted in a model built from categories.

Research Question 2. How can the groupoid model's uses of *symmetry* be made explicit in the syntax of type theory?

As mentioned above, the style of directed type theory appropriate to the category model is one which adopts a *polarity calculus*, a modal typing discipline for tracking and enforcing the variance of terms. This is necessary when trying to devise the *directed analogue* of the various constructions in the groupoid model: of course, the groupoid model constructions make free use of the fact that the contexts and types are interpreted in *groupoids* by inverting arrows as needed. Since we are interpreting our type theory into *categories*, we must instead pay careful attention to which "direction" of arrow we need. Our version of dependent type theory will be restricted in various ways, in order to ensure that all the dependencies involved are not just *functorial* (as they are in the groupoid model), but functorial *with the correct variance*.

In chapter 2 and chapter 3, we will encounter a number of these **polarity problems**—obstacles to copying the groupoid model definition directly into the category model. These come in three different varieties.

- A **shallow** polarity problem is one that arises because types in the groupoid model are interpreted as families of *groupoids*. For instance, modifying the formation rule of the Id type former to the formation rule of Hom presents a shallow polarity problem. Shallow polarity problems are, generally speaking, the easiest to address: we just need to annotate the types involved to indicate whether terms of those types are appearing covariantly or contravariantly: in $\mathrm{Hom}_A(t, t')$, t is contravariant and thus gets a *negative* annotation, whereas t' appears covariantly and gets a positive annotation.
- More imposing are the **deep** polarity problems: ones which rely on the fact that *contexts* are interpreted as groupoids in the groupoid model. To solve deep polarity problems, we're required to have polarity annotations on the *context*. The construction of Π-types presents a deep polarity problem: the groupoid model semantics of Π-types makes essential use of the fact that contexts are groupoids (not arbitrary categories). This is why North [Nor19] does not have Π-types but Licata and Harper [LH11] do: the latter adopts a deep polarity calculus with annotations on the contexts (and can therefore handle deep polarity problems), where the former only has shallow polarity.
- Finally, there are some polarity problems which do not seem to admit a satisfactory solution, even with the combined shallow- and deep-polarity calculus—the groupoid model semantics is simply too reliant on the invertibility of morphisms to be successfully rendered in the directed setting. We articulate a solution to many of these problems (in particular, some of the key polarity problems that *must* be solved in order to have synthetic category theory) in section 2.3. Our solution is a partial retreat from the fully directed setting: we work in *groupoid contexts in the category model*, which we abstractly articulate as **neutral contexts**. Neutral contexts ameliorate many of the most frustrating aspects of working in a deeply-polarized type theory, and provide an appropriate setting for synthetic category theory; for instance, it is only in a neutral context that a synthetic category (i.e. a type) has the same objects (terms) as its opposite category. Neutrality does not exhaust the polarity problems (for instance, directed function extensionality

remains at-large), but the present work demonstrates just how far into directed type theory/synthetic category theory it will get us.

Research Question 3. What is *directed equality*, and how does it work?

The purpose of identity types in Martin-Löf Type Theory is to internalize the metatheoretic notion of "sameness", i.e. judgmental equality: the assertion that t and t' are judgmentally equal is not a statement *in* type theory, but the type $\mathsf{Id}(t, t')$ is: we can, for instance, write functions which accept a term of type $\mathsf{Id}(t, t')$ as input or produce one as output. We can accordingly understand Hom types as internalizing some metatheoretic notion of *directed equality*, such as *reduction/evaluation* or *conversion*. Part of our goal in the present work is to understand what directed equality is and means. In section 3.1, we'll adopt a number of interpretations of directed equality: as the directed paths of directed homotopy spaces, as reductions in a rewriting calculus, and as temporal-computational processes. These different perspectives will help give us intuition for working with these types, and guidance for future development.

The (undirected) type theories we're interested in—the theory of the groupoid model and HoTT among them—are *intensional* type theories: the notion of "equality" given by identity types, i.e. *propositional equality*, is coarser than the metatheoretic *judgmental equality*—the type $\mathsf{Id}(t, t')$ can be inhabited without t and t' being *judgmentally* equal. This is in contrast to *extensional* type theories, where terms of the identity type $\mathsf{Id}(t, t')$ can be "reflected" to a judgmental equality $t = t'$; the theory of the groupoid model and HoTT cannot be extensional, as extensionality implies UIP, which these theories explicitly reject. That said, we don't wish to swear off extensionality completely: we wish our *intensional* theories to admit a limited set of **extensionality principles**, which, roughly speaking, relate the structure of the identity types between terms of type A to the structure of A itself. **Function extensionality** is a prime example: if we know that two functions are pointwise (propositionally) equal (i.e. we have $\varphi(x) : \mathsf{Id}(f(x), g(x))$ for arbitrary x), we wish to conclude that the functions themselves are equal, i.e. have a term $\mathsf{funext}(\varphi) : \mathsf{Id}(f, g)$. Univalence, i.e. "universe extensionality", is another such principle. The groupoid model, a product of the "extensionality principles in intensional type theory" tradition, validates both function and universe extensionality. HoTT extends the universe extensionality of the groupoid model to full univalence, which is indeed capable of proving many extensionality principles, including function extensionality [Uni13, Section 4.9] and the characterization of the identity types of Σ-types [Rij22, Section 9.3].

As the directed analogue of the theory of the groupoid model, the directed type theory of the present work will be *directed-intensional*. That is, the propositional directed equality of Hom-types will be coarser than the judgmental directed equality. But we too will want to import some **directed extensionality principles**. Though a thorny polarity problem prevents us from being able to state directed function extensionality (that the hom-terms between functions are synthetic natural transformations), we will have **directed universe extensionality/univalence**: the hom-terms between sets are given by functions between those sets. We also characterize the hom-types of Σ-types, giving us (the beginnings of) a **structure morphism principle** (analogous to the structure *identity* principle of undirected type theory), which tells us that homs

of "structures" are given compositionally in terms of the definition of the structure itself—making the aforementioned "central dogma of category theory" concrete in our synthetic setting.

Research Question 4. What does synthetic category-theoretic reasoning in the directed type theory of the category model look like?

Moving from undirected type theory to directed type theory, Id-types to Hom-types, means replacing the synthetic *groupoid* theory of the groupoid model with the synthetic *category* theory of the category model. In the category model, the types will have the structure of synthetic categories: the 'objects' of the category are the terms of the type, and the 'hom-set' between two such 'objects' t and t' is the hom-type $\mathsf{Hom}(t, t')$. It is a *synthetic* category theory because there's no such thing as proving *that something constitutes a category*—any type we can possibly write down in this theory is automatically is equipped with the structure of a category. Moreover, the functions $f : A \to B$ we can express in this theory are automatically functors on the categories A, B—there's no need to define the morphism part and check functoriality; this comes for free.

However, *having categories* is not the same thing as *doing category theory*: we want to have a framework for discussing the main ideas of (at least basic) category theory—like (co)limits, adjunctions, and representables—in our synthetic setting. This raises the question: *what should that framework look like*? Do we just want to use the available dependent types to express the usual category-theoretic definitions verbatim, or is there something more specialized to our setting which we can do?

The inspiration for our answer comes from directed path induction. Analogously to undirected intentional type theory, the directed equality types of the category model will come equipped with an elimination principle, the J-rule or **directed path induction**. Or, rather, by duality, it will come with two: a *forward*, or *coslice* path induction principle, and a *backward*, *slice* path induction principle. As with undirected path induction, these will allow us to prove statements generically about morphisms by proving them for the identity morphism, refl. Syntactically and semantically, these principles boil down to the fact that the identity morphism is initial in the coslice category and, dually, terminal in the slice category. Here we find the germ of our approach to synthetic category theory: the fact that refl is initial among coslices is not phrased as the standard category-theoretic *universal mapping property* ("for all coslices, there exists a unique map, ...") with the quantifiers replaces by dependent types. No, it's captured here as a *principle of induction*. We carry this throughout basic category theory: we define '(co)products', 'pullbacks', 'pushouts', 'exponentials', 'adjoints', etc. in our theory, as structures equipped with **principles of induction**. As mentioned above, we're able to verify that these articulations do indeed correspond to the standard notions by (a) verifying that the universal mapping property can be derived from the principle of induction within the synthetic category theory, and (b) seeing that the principles of induction are validated by the usual structures in the empty context of the category model.

0.3 Related Work

Models of Type Theory

Category-theoretic semantics for dependent type theory have been developed from a variety of different approaches, with each emphasizing different aspects and permitting generalization/weakening in different directions. We refer the reader to recent work by Ahrens, Lumsdaine, and North [ALN25] for a more exhaustive survey of what they (aptly) call a "veritable zoo" of different concepts. The notion of 'model of type theory' which we'll adopt has been articulated in numerous equivalent ways, including *categories of attributes* [Car78, Section 3.2],[Mog91]; *discrete comprehension categories* [Jac93]; *natural models* [Awo18]; and *categories with families* [Dyb95; Hof97]. We prefer the latter formulation, categories with families (CwFs): among these equivalent notions, CwFs provide the closest link to the syntax of type theory, are the most readily-expressible as a generalized algebraic theory, and are particularly suitable for formalization [BD08].

We develop models of type theory to permit *metatheoretic* reasoning. In particular, they serve two purposes: demonstrating the *independence* of certain statements in the theory (i.e. that the rules *cannot* prove said statements), and *soundness* results (that a given axiom/construct can be added to the theory without causing inconsistency or paradox). As discussed above, a celebrated example of the former is Hofmann and Streicher's use of the groupoid model to demonstrate the independence of UIP from the laws of Martin-Löf Type Theory [HS95]. We draw inspiration from this, using the *category* model to prove *symmetry* independent of our system of directed type theory. We also take inspiration from several soundness results, such as the *setoid model* of type theory[6]—whose original purpose was to demonstrate the soundness of adding quotient types [Hof95a] and function extensionality [Alt99] to intensional type theory— and the use of *presheaf models* [Hof97, Section 4] to justify cubical type theory [BCH14; CCHM18]. We develop the directed analogue of the setoid model, the *preorder model*, including a statement of the *fibrancy data* inherent to its types, analogous to the one given in [ABK+21] for the setoid model. Though our main investigation doesn't make use of presheaf models, we rehearse them in section 1.2 as particularly nice examples of CwFs, including their semantics for type universes [HS99].

Generalized Algebraic Theories

Generalized algebraic theories (GATs) were introduced by Cartmell [Car78; Car86] to study equational theories with *dependent sorts*. A paradigm example ([Car86, pp. 212–213], reproduced here as Example 1.1.11) of a theory requiring such dependency is that of *categories*, which have a sort of objects and a family of sorts—the hom-sets—doubly indexed over objects. We follow [KKA19] and [BCDE20] in modifying Cartmell's notion of GATs to (a) exclude the possibility of equations between sorts (instead only allowing equations between *elements* of a given sort); and (b) be articulated in an *intrinsically well-*

[6] We use Altenkirch's formulation, which takes "setoids" to be sets equipped with equivalence relations, rather than Hofmann's formulation, which uses *partial* equivalence relations, i.e. doesn't assume reflexivity.

formed manner (as opposed to Cartmell's definition, which defines GATs by restricting *pre-syntax* by a well-formedness relation).

section 1.3 and subsection 1.3.4 are largely a re-presentation of the results of [KKA19]: we give a signature language for GATs which is a fragment of their signature language of quotient inductive-inductive types, and use their definitions of algebra, homomorphism, displayed algebra (a generalization of the displayed *categories* of [AL19]), etc. This signature language is itself a quotient inductive-inductive type [Dij17] in the metatheory, following [AK16; Kap17]. Most significantly, we follow [KKA19] in using this language to define an *initial algebra* for every GAT; when the GAT in question is (extensions of) the GAT of CwFs, then this is the construction of a *syntax model*.

Though it doesn't play an explicit role in the present work, several of our developments—the results of section 2.2 in particular—anticipate a treatment of this style of directed type theory as a *second-order* generalized algebraic theory (SOGAT). SOGATs were introduced by Uemura [Uem23; Uem21] as a general framework for studying type theories; they have the benefit of automating much of the bureaucracy of GAT presentations of formal theories [KX24] by operating in a *higher-order abstract syntax*, and can be interpreted in presheaf models [Hof99]. At present, it remains an open problem to study substructural type systems in the SOGAT setting, though the technique used in [ACKS24] to study parametricity in the SOGAT setting may provide a solution.

Extensionality

A recurring theme across the past few decades of intensional type theory research has been the quest to incorporate extensional principles within intensional type theory, thereby obtaining the benefits of extensional type theory without its considerable downside (undecidable typechecking). This includes the aforementioned use of the setoid and groupoid models to justify the addition of function extensionality and "universe extensionality" to intensional type theory [Hof95b; Alt99; Hof95a; HS95; Alt21] as well as the *univalence axiom* [Voe14a; Voe14b] characteristic of homotopy type theory [Uni13] and its constructive metatheoretic justification in cubical type theory [BCH14; CCHM18]. The standard formulation of these extensionality principles assert them as equivalences—identities of functions are *equivalent* to pointwise identities, identities of types are *equivalent* to equivalences, etc.—but there is a branch of type theory—*observational type theory*—which goes further and makes these extensionality principles into *judgmental equalities*, i.e. the *definition* of identity types. This is done for the truncated setting (i.e. where UIP is accepted) in [AMS07], but there has recently been research [Shu22; AKS22; ACKS24] into the prospect of a *higher observational type theory*, i.e. a version of HoTT where univalence is not an axiom (as it is in "Book HoTT" [Uni13]) nor a theorem (as it is in cubical type theory [BCH14; CCHM18]), but a *definition*. Though we do not include any explicit treatment of (directed) higher observational type theory in the present work, we *do* work out the first-order theory theory here in anticipation of a hypothetical higher-order treatment in future work.

Directed Type Theory

Existing approaches to directed type theory can be placed into several classes. The most developed branch of directed type theory is *simplicial homotopy type theory (sHoTT)*. This approach was initiated by Riehl and Shulman [RS17], and provides a synthetic language for ∞-categories (analogously to how undirected HoTT provides a synthetic language for ∞-groupoids [VG11]). The fundamental idea of sHoTT is to extend ordinary HoTT with a *directed interval* and, instead of taking Hom-types as primitive (as we do here), defines them by functions out of the directed interval; this approach allows sHoTT to avoid the "polarity problems" inherent to other directed type theories. The task of developing ∞-category theory in sHoTT was taken up in particular by Weinberger [Wei22a; Wei24b; Wei22b; Wei24a]; the original system of Riehl-Shulman has recently been refined by Gratzer, et. al. [GWB24; GWB25] with the incorporation of *multi-modal type theory* [GKNB20]; and sHoTT has been implemented in an experimental proof assistant, RZK [KRW23]. A related, but distinct, approach is *bicubical* type theory [WL20], which, also implements hom-types as maps from the directed interval. A primary difference between sHoTT and bicubical type theory is their model theory—the former is modeled in *bisimplicial* sets (which are generally nonconstructive) whereas the latter is modeled in constructive *bicubical* sets; we refer the reader to Weaver [Wea24] for a more detailed comparison between these systems.

Among directed type theories which *do* adopt a typing discipline to track *polarities*, one prevalent tradition has been so-called *two-dimensional theories*. Licata and Harper's theory [LH11] developed a formal type theory interpreted in the 2-category of categories, with contexts interpreted by categories, substitutions interpreted by functors, and *directed reductions* (a directed, proof relevant replacement for *judgmental equality*) interpreted by natural transformations—this latter construct is the reason for the moniker "two-dimensional" (the present theory would therefore be "one dimensional", as we accept the usual undirected judgmental equality of Martin-Löf Type Theory). The Licata-Harper type theory is notable for interpreting Π-types in the category of categories; this is possible because they have a negation operator on *contexts* interpreted by the *opposite category* construction, and are thus able to solve the "polarity problem" we discuss in section 2.2. This theory was developed further by Nuyts [Nuy15], who added further kinds of variance to account for variables which appear neither negatively nor positively. However, the semantics of these theories are either somewhat *ad-hoc* (Licata-Harper) or not addressed (Nuyts), as noted in [ANvdW23]; this latter work establishes a more principled, general semantic framework for two-dimensional directed type theories with the notion of *comprehension bicategories*. However, a development of hom-types and directed type theory proper remains to be done in this setting.

Several approaches to synthetic category theory make central use of *profunctors* (functors with both a positive and negative component) both syntactically and semantically. In particular, this is useful to address the mixed variance of the term t in the hom-type $\mathsf{Hom}(t, t)$. One abstract category-theoretic treatment of profunctors is in the notion of *double categories*—categories with "vertical" and "horizontal" notions of morphisms, such as the double category of categories, which has categories as objects but *functors* and *profunctors* as two distinct notion of *morphism*. Naturally, proposals have been made for directed type theories based on (virtual) double-categorical struc-

ture, such as the *virtual equipment type theory (VETT)* of New and Licata [NL23] and recent reformulations by Nasu [Nas25; Nas24]. These theories have proved capable of an impressive amount of synthetic category theory, including a theory of natural transformations (which eludes the present work); however, semantic reasoning about such theories seems much more involved than for the present theory, as the mathematical structures interpreting the former—hyperdoctrines on split fibrant virtual double categories—are significantly more complicated. Let us also mention another profunctor-based approach, the recent theory of Laretto et al. [LLV24], which interprets terms as difunctors (endo-profunctors) and entailments as *dinatural transformations*. Though this theory is not exactly a *directed type theory* in the sense used here—Hom is a predicate, not a type-former—it does provide a compelling type-theoretic syntax for phrasing category-theoretic arguments, particularly from the (co)end calculus [Lor23].

The present theory has the greatest overlap with the directed type theory of North [Nor19]. Like North's theory, ours will be "one-dimensional" (in contrast to the aforementioned "two-dimensional" theories) in that we maintain *judgmental equality* as an undirected, metatheoretic notion; our "shallow polarity" (the polarity annotations on types) works the same as North's (and has the same semantics); and we adopt the same hom-formation rule. Also, our basic semantics are the same: our *category model* is the same as interpretation of contexts, substitutions, types, and terms given by North. However, our theory departs from North's in several key ways.

- Syntactically, we adopt different approaches for solving the "polarity problem" posed by refl: since our Hom-formation rule demands its domain to be a term of type A^- and its codomain a term of type A, how can a single term t serve as both the domain and the codomain, as it must in refl_t? North solves this by introducing a *core* modality on types: for each type A, a type A^0, along with coercion operators i, i^{op} sending terms of A^0 to terms of A and A^-, respectively. Thus, the term refl_t can only be asserted when t is of type A^0, and be of type $\mathsf{Hom}(i^{\mathrm{op}}\ t, i\ t)$. This has the advantage of working in an arbitrary context; however, the resulting principles of directed path induction consequently only apply when one endpoint is of a core type. Given that North doesn't include any constructors for producing terms of core types, it's unclear if there's any way to make proofs about *arbitrary* hom-terms. Our solution, on the other hand, is to restrict the *context* to a "neutral context", where there's a general coercion between terms of A and A^-. This eliminates the need for core types,[7] and is quite useful for solving other polarity problems. We suspect our directed path induction to be equivalent to North's in a setting with neutral contexts *and* core types, but don't introduce core types explicitly in the present work.
- North's theory does not include what we call "deep polarity", i.e Licata-Harper-style negation operators on the *contexts* (internalizing the fact that contexts are categories, and hence have opposites). We regard deep polarity as prerequisite to a development of dependent types (Π-types specifically) in directed type theory,

[7] The category model does support core types, with precisely the semantics North gives; they may be added to a future version of the our theory, as taking the core of a synthetic category is still a useful operation to consider. It's unclear if there is a model of the present theory which expressly *does not* have core types, so we don't regard the elimination of core types from the syntax as something likely to result in a broader class of models.

and, moreover, it is needed in the present work to be able to state what a "neutral" context even is.

We're aware of ongoing efforts [CMN24; Man24] to extend North's theory to be able to support dependent types and do meaningful synthetic category theory. Our preliminary understanding is that such extensions evidently involve extending the negation and core operations to contexts after all, and that the resulting polarity calculus seems more expressive than the one expounded here. However, we defer more detailed comparison until a more definitive statement of such a theory is given.

An abbreviated version of the present work is available in preprint form as [NA24], to appear as [NA25]. That work ("the TYPES paper") differs from the present work in several ways, including the following.

- The present work includes a detailed development (chapter 1) of 'generalized algebra', our methodology of studying structures using generalized algebraic theories (including the signature languages, ONEGAT and NOUGAT; the construction of initial algebras and concrete CwFs, and fibrancy).
- The definition of "NPCwF" given in the TYPES paper corresponds to the "unary NPCwFs" of the present work (Definition 2.3.25), and does not include the facilities for multi-variable functions incorporated into the full definition of "NPCwF" given here (Definition 2.3.44). The definition of NPCwF given in the TYPES paper also takes both the ee isomorphisms *and* term-negation as primitive (and axiomatizes their interaction), whereas the present definition defines term-negation using ee and includes sufficient axioms to prove the behavior of term-negation (specifically its involutive property).
- We develop Σ-types in the present work, use the neutral-context operations to characterize $(\Sigma AB)^-$, and assert a characterization of the hom-types of Σ-types.
- The present work includes a development of transport in the directed setting, defined using directed path induction.
- The present work asserts that $\mathsf{Hom}_{A^-}(t', t) = \mathsf{Hom}_A(t, t')$, and develops the principle of *slice path induction* dual to the usual, "coslice", principle of directed path induction (that appears in the TYPES paper).
- The application of functions in informal directed type theory is done using the app^+ operator in the present work, so a function $A \to B$ is applied to an argument of type A; in the TYPES paper, some of the *contravariant* structure is kept around in the informal syntax, so a function $A \to B$ is applied to a term of type A^-. The choice used here is much more convenient, e.g. for defining map.
- The methodology of "inductive category theory", i.e. phrasing universal mapping properties as induction principles, does not appear in the TYPES paper. We also go much further in developing synthetic category theory in the present work: the notions of (co)product, pullback, pushout, and adjunction do not appear in the TYPES paper.
- The TYPES paper concludes that natural transformations cannot be treated component-wise in the present framework. In the present work, we show that hom-terms between functions *can* be treated as natural transformations and *applied* component-wise; but we still cannot *write* natural transformations component-wise.

0.4 Contribution

> "So my aim here is not to teach the method that everyone must follow for the right conduct of his reason, but only to show in what way I have tried to conduct mine."
>
> René Descartes, *A Discourse on the Method*

> "Two roads diverged in a wood, and I—
> I took the one less traveled by,
> And that has made all the difference."
>
> Robert Frost, *The Road Not Taken*

The purpose of the present work is not, so to speak, to make type theory grow *taller*, but rather to see its roots grow *stronger* and *deeper*. Perhaps in the future some version of this theory could serve as the foundation for a synthetic theory of higher- and ∞-categories, and truly provide a directed *homotopy* type theory. But in this thesis, we only aspire to parity with the groupoid model and its type theory, that is, to expound a synthetic theory of 1-categories with semantics in 1-categories. But it is our sincere hope that a reflection on this more modest mathematical topic might still provide useful insights to type- and category-theorists of all stripes; moreover, we intend for this work to serve as a reference point for the "polar style" of directed type theory, a signpost to guide future investigations. The contributions of this thesis are as follows; the most significant contributions (by our estimation) are in **bold**.

Chapter 1: Type Theory as a Generalized Algebraic Theory

- Interpretation of the material from "Constructing QIITs" by Kaposi, Kovács, and Altenkirch [KKA19] as a mathematical discipline—"generalized algebra"—for specifying GATs and calculating with them
  This includes the construction of what we dub "concrete CwFs"—models of type theory whose contexts are algebras (of a given GAT), types are displayed algebras, etc.
- Precise specification of the typical informal style of introducing GATs with a syntactic sugar language, nouGAT, which compiles down to the less-human-readable signature language, oneGAT (a fragment of the QIIT signature language from [KKA19])
- Explicit development and comparison of fibrancy data for displayed preorders, groupoids, and categories (adapting notions from Ahrens-Lumsdaine [AL19], expanding on the presentation from [ABK+21]). We articulate this as a restriction of these concrete CwFs to "fibrant concrete CwFs."
- Posing of a general question: which GATs are models for a synthetic theory of themselves (e.g. the groupoid model gives a synthetic theory of groupoids)? This seems to be essentially the same question as, "for which GATs can we restrict their concrete CwF to a fibrant concrete CwF?"

Chapter 2: The Polarity Calculus

- Definition of the category model (collecting the definition from [Nor19] into a more elementary and explicit presentation) and the preorder model, the directed analogues of the groupoid model and the setoid model, respectively
- **Articulation of a type theory which combines the "shallow polarity" (negation operator on types) from North [Nor19] with the "deep polarity" (negation on contexts and context extension) of Licata-Harper [LH11] in order to solve a variety of "polarity problems"**
- Definition of *Polarized CwFs*, the abstract model notion appropriate to this type theory
- Exploration of how each PCwF contains within it two type theories—one "positive" and one "negative."
  - In the category model, this corresponds to the CwF structures where types are (split) opfibrations and fibrations, respectively.
- Development of dependent types in polarized type theory (following Licata-Harper)
- A demonstration that dependent types are poorly-behaved in arbitrary contexts, and **development of a methodology of *neutral contexts* within polarized type theory**, where the theory works better
- Abstraction of the salient features of neutral contexts in the notion of *Neutral-Polarized CwF*

Chapter 3: Directed Type Theory

- Statement of directed path induction in neutral contexts
- An account of Hom-types vs. Identity-types, and **explanation of how our J-rule can prove symmetry for the latter but not the former**
- Definition of *directed CwFs*, a family of model concepts parametrized by the two "dimensions" of "truncation": how many "levels up" before all parallel morphisms become equal, and how many "levels up" before all morphisms become invertible
- Explicit treatment of the dual statements of directed path induction as "coslice path induction" and "slice path induction", which are inter-derivable (modulo an observational law concerning the hom-types of opposite categories)
- Development of directed universes, particularly the universe of sets in the category model
- Observational rules characterizing the Hom-types of important type constructors, particularly Σ-types and the universe of sets (giving a rule of *directed univalence*)
- Metatheoretic reasoning with category model (proving that our J-rule cannot prove symmetry for non-neutral types), in the tradition of Hofmann-Streicher

Chapter 4: Synthetic-Inductive Category Theory

- Discussion of the syntax model of $(1, 1)$-directed type theory, and how to work informally in a neutral context
- Development of the synthetic category structure of types

- **A novel presentation of basic category theory, where the *universal mapping properties* are *principles of induction***
- A methodology for verifying that our inductive presentation of category-theoretic concepts corresponds to the standard notions
- A proof—the directed analogue of the construction of ap—that every function expressible in (1,1)-directed type theory is automatically a *synthetic functor*
- A synthetic-inductive treatment of products, coproducts, pullbacks, pushouts, adjunctions, and (co)limits

0.5 Metatheory and Notation

We work in a constructive type-theoretic metatheory. We'll write $t = t'$ to indicate that t and t' are judgmentally equal, i.e. equal in the metatheory; we generally treat this notion of equality as intensional, but we do assume the uniqueness of identity proofs and function extensionality. We'll generally write := to make definitions. We assume a type Prop of h-propositions and a type Set of h-sets. We generally ignore issues of size, and don't make universe levels explicit at any point. To define the language ONEGAT in section 1.3, we assume the existence of a specific quotient inductive-inductive type with the expected elimination principle.

Throughout, we use a notational style similar to AGDA, which we call PSEUDOAGDA. We write the dependent function and product types as

$$(x : X) \to Y(x) \qquad \text{and} \qquad (x : X) \times Y(x),$$

respectively, and write $\lambda x \to e$ for lambda-abstraction. For multivariable functions, we don't distinguish between "curried" and "uncurried" functions, writing the arguments separated by either spaces or commas as convenient. We use curly-braces to indicate arguments are implicit; any variable appearing free is also assumed to be an implicit argument.

We introduce dependent sums with named components using **record** notation: for instance, if we wrote

PSEUDOAGDA

```
record 𝔑–Alg : Set where
  N : Set
  z : N
  s : N → N
```

then, for some $\Gamma : \mathfrak{N}$-Alg, we could say Γ.N, Γ.z, and Γ.s for the respective components, or we could introduce the components with their own names by writing (foo, bar, bat) : $\mathfrak{N}$-Alg to, e.g. indicate that bat : foo → foo. On the other hand, we'll sometimes refer to the components of a dependent sum by indices: if, say, we had some

$$\Phi : X \times Y \times Z,$$

then we might write $\Phi_{\#1} : X$, $\Phi_{\#2} : Y$, and $\Phi_{\#3} : Z$; we may even omit the numbered subscripts if it's clear from context which component of Φ is being used. We'll name the components with #'s in the **record** notation to make them anonymous, e.g. in

pseudoAgda

```
record Tm (Γ : Con) (A : Ty Γ) : Set where
  #1 : (x : |Γ|) ⟶ |A x|
  #2 : (x₀₁ : x₀ ~Γ x₁) ⟶ A x₀₁ (#1 x₀) ~A(x₁) (#1 x₁)
```

We make extensive use of category-theoretic notions. Given a category $\mathcal{C}$, we'll write $|\mathcal{C}|$ to indicate the type (set) of objects of $\mathcal{C}$, and write $\mathcal{C}\ [I, J]$ for the set of $\mathcal{C}$-morphisms from I to J. Somewhat unusually, we will use the notation $F : \mathcal{C} \Rightarrow \mathcal{D}$ to indicate that F is a functor from $\mathcal{C}$ to $\mathcal{D}$. We assume "functor extensionality": that if both the object- and morphism-parts of a two functors are pointwise equal, then the functors themselves are equal. We write Set for the category of sets and Cat for the category of categories. We only treat the latter as a 1-category.

In chapter 1, we introduce a signature language onеGAT for specifying generalized algebraic theories, along with a human-readable syntactic sugar nouGAT that is supposed to compile down to onеGAT. The compiler is mostly implemented in Lean4, though not all of the syntax used in this thesis is implemented yet.

Later on, we state a number of constructions in the syntax of directed type theory. These are still presented using some pseudoAgda syntax (such as **where** clauses), but this is intended as just syntactic sugar for the raw syntax presented in the text. An implementation of this syntax in Agda using rewrite rules is in progress.

List of GATs & Structures

Sets

$\mathfrak{Set}$ . . . 25
$\mathfrak{Set}$ (oneGAT) . . . 57
. . . 219
Set ($\mathfrak{Set}$-algebra) . . . 219
$\mathfrak{Set}$ (Initial $\mathfrak{Set}$-algebra) . . . 67
. . . 72
Set (category of $\mathfrak{Set}$-algebras) . . . 26

Pointed Sets

$\mathfrak{P}$ . . . 26
$\mathfrak{P}$ (oneGAT) . . . 58
. . . 219
Pointed set ($\mathfrak{P}$-algebra) . . . 219
$\mathbb{1}$ (Initial $\mathfrak{P}$-algebra) . . . 65
. . . 67
. . . 72
$\mathrm{Set}_\bullet$ (category of $\mathfrak{P}$-algebras) . . . 26

Bipointed Sets

$\mathfrak{B}$ . . . 26
$\mathfrak{B}$ (oneGAT) . . . 58
. . . 220
$\mathfrak{B}$-algebra . . . 26
. . . 220

Natural Number Algebras

$\mathfrak{N}$ . . . 26
$\mathfrak{N}$ (oneGAT) . . . 58
. . . 220
Nat-algebra ($\mathfrak{N}$-algebra) . . . 22
. . . 220
$\mathbb{N}$ (Initial $\mathfrak{N}$-algebra) . . . 23
. . . 65
. . . 67
. . . 73
NatAlg (category of $\mathfrak{N}$-algebras) . . . 27

Displayed $\mathfrak{N}$-algebra . . . 68
Even-Odd Algebras
$\mathfrak{EO}$ . . . 27
$\mathfrak{EO}$ (oneGAT) . . . 58
. . . 221
$\mathfrak{EO}$-algebra . . . 27
. . . 221
Monoids
$\mathfrak{Mon}$ . . . 30
$\mathfrak{Mon}$ (oneGAT) . . . 221
Monoid ($\mathfrak{Mon}$-algebra) . . . 221
Mon (category of $\mathfrak{Mon}$-algebras) . . . 30
Groups
$\mathfrak{Grp}$ . . . 30
$\mathfrak{Grp}$ (oneGAT) . . . 59
. . . 222
Group ($\mathfrak{Grp}$-algebra) . . . 222
Grp (category of $\mathfrak{Grp}$-algebras) . . . 30
Quivers
$\mathfrak{Quiv}$ . . . 27
$\mathfrak{Quiv}$ (oneGAT) . . . 59
. . . 222
Quiver ($\mathfrak{Quiv}$-algebra) . . . 222
Quiv (category of $\mathfrak{Quiv}$-algebras) . . . 27
Reflexive Quivers
$\mathfrak{rQuiv}$ . . . 29
$\mathfrak{rQuiv}$ (plain nouGAT) . . . 28
$\mathfrak{rQuiv}$ (oneGAT) . . . 59
. . . 223
Reflexive Quiver ($\mathfrak{rQuiv}$-algebra) . . . 223
Preorders
$\mathfrak{PreOrd}$ . . . 33
$\mathfrak{PreOrd}$ (oneGAT) . . . 223
Preorder ($\mathfrak{PreOrd}$-algebra) . . . 223
Monotone maps (morphism of $\mathfrak{PreOrd}$-algebras) . . . 33
PreOrd (category of $\mathfrak{PreOrd}$-algebras) . . . 33
Setoids
$\mathfrak{Setoid}$ . . . 33
$\mathfrak{Setoid}$ (oneGAT) . . . 224
$\mathfrak{Setoid}$-algebra . . . 34
. . . 224
Displayed Setoid (Displayed $\mathfrak{Setoid}$-algebra) . . . 75
Section of a displayed $\mathfrak{Setoid}$-algebra . . . 76
Categories
$\mathfrak{Cat}$ . . . 31
$\mathfrak{Cat}$ (oneGAT) . . . 225

Category ($\mathfrak{Cat}$-algebra) . . . 31
. . . 225
Functor (morphism of $\mathfrak{Cat}$-algebras) . . . 32
Cat (category of $\mathfrak{Cat}$-algebras) . . . 32
Groupoids
$\mathfrak{Grpd}$ . . . 33
$\mathfrak{Grpd}$ (oneGAT) . . . 226
$\mathfrak{Grpd}$-algebra . . . 33
. . . 226
Categories with Families
$\mathfrak{CwF}$ . . . 37
$\mathfrak{CwF}$ (oneGAT) . . . 61
. . . 228
CwF ($\mathfrak{CwF}$-algebra) . . . 39
. . . 229
CwFs /w unit type
$\mathfrak{CwF}_1$ . . . 46
$\mathfrak{CwF}_1$ (oneGAT) . . . 230
CwF /w unit type ($\mathfrak{CwF}_1$-algebra) . . . 44
CwFs /w bool type
$\mathfrak{CwF}_2$ . . . 47
$\mathfrak{CwF}_2$ (oneGAT) . . . 231
CwF /w bool type ($\mathfrak{CwF}_2$-algebra) . . . 44
CwFs /w Π-types
$\mathfrak{CwF}_\Pi$ . . . 49
$\mathfrak{CwF}_\Pi$ (oneGAT) . . . 232
CwF /w Π-types ($\mathfrak{CwF}_\Pi$-algebra) . . . 48
CwFs /w Σ-types
$\mathfrak{CwF}_\Sigma$ . . . 50
CwF /w Σ-types ($\mathfrak{CwF}_\Sigma$-algebra) . . . 48
CwFs /w Extensional identity types
CwF /w Extensional identity types ($\mathfrak{CwF}_{E=}$-algebra) . . . 54
CwFs /w Intensional identity types
CwF /w Intensional identity types ($\mathfrak{CwF}_{I=}$-algebra) . . . 53
Polarized Categories with Families
$\mathfrak{PCwF}$ . . . 94
$\mathfrak{PCwF}$ (oneGAT) . . . 233
Polarized CwF ($\mathfrak{PCwF}$-algebra) . . . 93
. . . 233
Zero-ary Neutral-Polarized CwFs
$\mathfrak{N}_0\mathfrak{PCwF}$ . . . 112
Zero-ary NPCwF ($\mathfrak{N}_0\mathfrak{PCwF}$-algebra) . . . 110
Directed CwFs
Directed CwF ($\mathfrak{DCwF}$-algebra) . . . 141
(1,1)-Directed CwFs
(1,1)-Directed CwF ($(1,1)$-$\mathfrak{DCwF}$-algebra) . . . 147

(0,1)-Directed CwFs
(0,1)-Directed CwF ($(0,1)$-$\mathfrak{DCwF}$-algebra) 147
(1,0)-Directed CwFs
(1,0)-Directed CwF ($(1,0)$-$\mathfrak{DCwF}$-algebra) 147
(0,0)-Directed CwFs
(0,0)-Directed CwF ($(0,0)$-$\mathfrak{DCwF}$-algebra) 147

Chapter 1

Type Theory as a Generalized Algebraic Theory

> "With reference to this freeing the elements from every other content (abstraction) we are justified in calling numbers a *free creation of the human mind*....The relations or laws which are derived entirely from [the natural number axioms]...form the first object of the *science of numbers* or *arithmetic*."
>
> Richard Dedekind, *Was sind und was sollen die Zahlen*

We take inspiration for our methodology from the development of the central construct of mathematics, *the natural numbers*. The mathematical community's understanding of the natural numbers began to crystallize towards the end of the nineteenth century, and it became clear that $\mathbb{N}$ could be apprehended from two different perspectives. On the one hand, the natural numbers constitute a *structure*: $\mathbb{N}$ is a *set equipped with stuff*, namely a chosen element *zero* and an endofunction *succ*. We can abstractly articulate this kind of structure into the concept of a *nat-algebra*; in the present work, we'll write $\mathfrak{N}$-Alg for the type of nat-algebras.

pseudoAgda

```
record 𝔑–Alg : Set where
  N : Set
  z : N
  s : N ⟶ N
```

The natural numbers can be articulated as a special nat-algebra, namely one where the endofunction s is injective and has every $n : N$ in its image besides z; Dedekind [Ded63] recognized that any two such systems are canonically isomorphic, hence any one suffices as "*the*" natural numbers.

pseudoAgda

```
data ℕ : Set where
  zero : ℕ
  succ : ℕ → ℕ
```

Figure 1.1: pseudoAgda declaration of the type $\mathbb{N}$, the initial $\mathfrak{N}$-algebra.

On the other hand, it was recognized that the science of natural number arithmetic was a candidate for *axiomatization*, following in Euclid's esteemed example. That is, we can understand the natural numbers as a purely-formal symbolic calculus, governed only by those rules which we impose. The germ of this thinking can be seen in modern-day *inductive definitions* of $\mathbb{N}$, such as in Figure 1.1. The **data** keyword tells us that $\mathbb{N}$ is *freely generated* from the constructors zero and succ: there are no natural numbers (no closed terms of type $\mathbb{N}$) besides those that can be obtained from well-typed combinations of these constructors, and there are no equations that hold between natural numbers besides those equations which logically *have to* hold.[1] In this articulation, the natural numbers are purely *syntactic* creatures: just those combinations of symbols that constitute a well-typed term of type $\mathbb{N}$ per the above specification.

Of course, it's no coincidence that the definition of $\mathfrak{N}$-Alg and the inductive specification of $\mathbb{N}$ appear so similar. Indeed, we wish to view them as two different manifestations of the same underlying "idea", the *mere concept* of the natural numbers. In this view, there is some underlying concept $\mathfrak{N}$, and $\mathfrak{N}$-Alg and $\mathbb{N}$ are the manifestations of $\mathfrak{N}$ as a class of structures and as a formal syntax, respectively. The well-known link between these manifestations—that $(\mathbb{N}, \mathrm{zero}, \mathrm{succ})$ is the *initial nat-algebra*—is established by a given function

$$\mathrm{rec}_{\mathfrak{N}}(N, z, s) : \mathbb{N} \to N$$

for each $(N, z, s) : \mathfrak{N}$-Alg. This is yet another manifestation of the $\mathfrak{N}$-concept: it is not *just* a function, but a *homomorphism of* $\mathfrak{N}$*-algebras* from $(\mathbb{N}, \mathrm{zero}, \mathrm{succ})$ to (N, z, s). That is, it sends zero to z and must satisfy

$$\mathrm{rec}_{\mathfrak{N}}(N, z, s)(\mathrm{succ}\ n) = s(\mathrm{rec}_{\mathfrak{N}}(N, z, s)\ n)$$

for all $n : \mathbb{N}$.

We can go further. The critical property of the natural numbers is the *principle of mathematical induction*: that for any predicate on $\mathbb{N}$ (which we'll understand in the proof-relevant manner, i.e. as a family of sets $P : \mathbb{N} \to \mathrm{Set}$), in order to to prove (i.e. *inhabit*) $P(n)$ for all $n : \mathbb{N}$, it suffices to supply a "base case" proving $P(\mathrm{zero})$ and an

[1] For instance, it must be the case that zero = zero and $\mathrm{succ}(m) = \mathrm{succ}(m)$ for all m, by the logical properties of equality (which are prior to this definition); but there is no basis to conclude $\mathrm{zero} = \mathrm{succ}(n)$ for some n or that $\mathrm{succ}(m) = \mathrm{succ}(n)$ for some $m \neq n$, and so the "freeness" of this definition demands that such equations *not* hold.

"inductive step" which, for any n, turns proofs of $P(n)$ into proofs of $P(\text{succ } n)$. In summary, the "input data" to the inductive process can be summarized as the following.

pseudoAgda

```
record indData : Set where
  P : ℕ → Set
  BC : P zero
  IS : (n : ℕ) → P n → P (succ n)
```

If we look carefully at this definition, the "$\mathfrak{N}$-ness" is again apparent: this is the notion of a predicate over $\mathbb{N}$ which respects the nat-algebra structure in the appropriate way. And likewise with the *output* of induction as well: given such a $(P, \mathsf{BC}, \mathsf{IS})$, the principle of induction gives us some dependent function

$$\mathsf{ind}_{\mathfrak{N}}(P, \mathsf{BC}, \mathsf{IS}) : (n : \mathbb{N}) \to P(n)$$

which *also* respects the $\mathfrak{N}$-structure:

- $\mathsf{ind}_{\mathfrak{N}}(P, \mathsf{BC}, \mathsf{IS})(\text{zero}) = \mathsf{BC}$, and
- $\mathsf{ind}_{\mathfrak{N}}(P, \mathsf{BC}, \mathsf{IS})(\text{succ } n) = \mathsf{IS}(\mathsf{ind}_{\mathfrak{N}}(P, \mathsf{BC}, \mathsf{IS})\ n)$ for all n.

In what is to come, we'll refer to $(P, \mathsf{BC}, \mathsf{IS})$ as a *displayed* $\mathfrak{N}$-algebra (over the $\mathfrak{N}$-algebra $(\mathbb{N}, \text{zero}, \text{succ})$) and $\mathsf{ind}_{\mathfrak{N}}(P, p_{\text{zero}}, p_{\text{succ}})$ as a *section* of $(P, \mathsf{BC}, \mathsf{IS})$. Thus, the principle of induction merely says that every displayed $\mathfrak{N}$-algebra over $(\mathbb{N}, \text{zero}, \text{succ})$ admits a section.

The foregoing description (and particularly speak of abstract "concepts" "manifesting" as diverse mathematical constructions) may seem like we're trying to make ordinary, well-understood ideas like induction seem more mystical than they are. We can remedy this, hopefully, by being precise about what kind of thing the "concept of the natural numbers", $\mathfrak{N}$, is. The answer is that it is a *generalized algebraic theory*: $\mathfrak{N}$ is a GAT. In the present chapter, we lay out the methodology we'll call **generalized algebra**—the study of structures given as GATs. We'll set out a language for writing down GATs and demonstrate that we can perform all the above constructions for an arbitrary GAT $\mathfrak{G}$: from the mere concept of structure expressed in the GAT $\mathfrak{G}$, all these notions—algebra, morphism, syntax/initial algebra, displayed algebra, section, and so on—emanate automatically.

Generalized algebra will prove a powerful toolset for studying the semantics of type theory. By expressing the notion of 'model of type theory' as a GAT, we automatically get a theory of models and model homomorphisms (the algebras and morphisms) and, moreover, we automatically get a *syntax model* which, like $\mathbb{N}$ does for $\mathfrak{N}$, renders our type theory as a purely-syntactic, symbolic calculus. This is all highly modular, of course: whatever kind of type theory we're interested in (e.g. *directed type theory*), we just need to write its semantics down as a GAT to take advantage of all this machinery. Moreover, generalized algebra will, like any good system of mathematical logic, start to *talk about itself*: the syntax for writing down GATs is a type theory, so it can be viewed as the initial algebra of a special GAT. More usefully for our purposes, we'll find that the algebras, morphisms, displayed algebras, and sections for a given GAT arrange into a model of type theory. In this way, certain GATs $\mathfrak{G}$ can provide semantics for a

synthetic theory of $\mathfrak{G}$*-algebras*; one such GAT, the GAT of *categories*, will be the focal point of our investigation.

1.1 Specifying Structures as GATs

> "I called what alone mattered to me the conceptual content [*begriflichen Inhalt*]. Hence this definition must always be kept in mind if one wishes to gain a proper understanding of what my formula language is. That, too, is what led me to the name *Begriffsschrift*. Since I confined myself for the time being to expressing relations that are independent of the particular characteristics of objects, I was also able to use the expression 'formula language for pure thought'."
>
> Gottlob Frege, *Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought*

We'll have two syntaxes for specifying GATs. In section 1.3, we'll introduce ONEGAT, a dependent type theory whose *contexts* are all the different GATs. This language is a fragment of the signature language of Kaposi, Kovács, and Altenkirch (henceforth, 'KKA') [KKA19], and is ideally suited for defining concepts for *all GATs* (such as the notion of a $\mathfrak{G}$-algebra for arbitrary $\mathfrak{G}$) because it is given as a quotient inductive-inductive type. This means we can define, e.g. "$\mathfrak{G}$-algebra" by induction on $\mathfrak{G}$. However, for anything but the simplest GATs, their presentation in ONEGAT is so verbose as to be unreadable (see e.g. the examples in Appendix A)—ONEGAT functions as "GAT machine code": necessary to compute with, but not made for human eyes. So, we adopt a human-readable representation, a *syntactic sugar*, which compiles down to ONEGAT. We call this language NOUGAT. In the present section, we introduce the NOUGAT syntax by example and informally indicate how any GAT $\mathfrak{G}$ specified in this way gives rise to the notions of $\mathfrak{G}$-algebra, $\mathfrak{G}$-homomorphism, initial $\mathfrak{G}$-algebra, etc. But our *actual* procedure will be to compile $\mathfrak{G}$'s NOUGAT representation down to ONEGAT (this is done by the accompanying LEAN code), and then obtain these notions using the precisely-defined inductive definitions (taken from KKA).

In the examples that follow, we give the NOUGAT definition of each GAT, written as a snippet of LEAN code. The NOUGAT syntax for a GAT is a series of *declarations* enclosed between {⟦ ⟧} brackets; we give progressively more complex examples for what kinds of declarations are allowed. Alongside each $\mathfrak{G}$, we specify the corresponding category of $\mathfrak{G}$-algebras (set-based interpretations of $\mathfrak{G}$) and $\mathfrak{G}$-algebra homomorphisms (structure-preserving maps).

The most basic components we can specify in a GAT are *sorts* and *elements* of those sorts. In defining the corresponding notion of "algebra", this amounts to stipulating sets and elements of those sets, respectively.[2]

Example 1.1.1 (Sets—NOUGAT). [ONEGAT] The GAT $\mathfrak{Set}$ of **sets** is given by

[2] In other words, (_) - Alg is the interpretation of GATs into Set.

Lean—nouGAT

```
def 𝔖et : GAT := {[ X : U ]}
```

.

The category Set has sets as objects and functions as morphisms.

Example 1.1.2 (Pointed Sets—nouGAT). [oneGAT] The GAT $\mathfrak{P}$ of **pointed sets** is given by

Lean—nouGAT

```
def 𝔓 : GAT := {[ X : U, x : X ]}
```

$\mathrm{Set}_\bullet$ is the category whose

- objects are pairs (X, x_0) where X is a set and $x_0 : X$;
- morphisms $(X, x_0) \to (Y, y_0)$ are functions $f : X \to Y$ such that $f(x_0) = y_0$.

Example 1.1.3 (Bipointed Sets—nouGAT). [oneGAT] The GAT $\mathfrak{B}$ of **bipointed sets** is given by

Lean—nouGAT

```
def 𝔅 : GAT := {[ X : U, x : X, x' : X ]}
```

A $\mathfrak{B}$-algebra, a bipointed set, is a triple (X, x_0, x_1) where X is a set and $x_0, x_1 : X$. A mprphism of bipointed sets is a function preserving both points.

Note that every GAT[3] must start out with the declaration of at least one sort—we need some starting point, some "base", upon which to "attach" the rest of the structure. All of these categories of algebras are therefore "concrete categories" in the usual category-theoretic parlance, i.e. categories whose objects are structured sets and whose morphisms are structure-preserving functions on those sets.

Given an already-declared sort T, a GAT may include further components that depend on element(s) of T. We write and think of these as *functions*.

Example 1.1.4 (Natural Number Algebras—nouGAT). [oneGAT] The GAT $\mathfrak{N}$ of **natural number algebras** is given by

[3] Besides the *empty GAT*, which has a single, trivial model.

LEAN—NOUGAT

```
def 𝔑 : GAT := {[
    Nat  : U,
    zero : Nat,
    succ : Nat ⇒ Nat
]}
```

NatAlg is the category whose
- objects are triples (N, z, s) where N is a set, $z\colon N$, and s is a function $N \to N$;
- morphisms $(M, y, p) \to (N, z, s)$ are functions $\varphi\colon M \to N$ such that $\varphi(y) = z$ and $\varphi \circ p = s \circ \varphi$.

Example 1.1.5 (Even-Odd Algebras—NOUGAT). [ONEGAT] The GAT $\mathfrak{EO}$ of **even-odd algebras** is given by

LEAN—NOUGAT

```
def 𝔈𝔒 : GAT := {[
    Even  : U,
    Odd   : U,
    zero  : Even,
    succ  : Even ⇒ Odd,
    succ' : Odd ⇒ Even
]}
```

A $\mathfrak{EO}$-algebra consists of two sets, E and O, with a chosen element $z\colon E$ and functions

$$s\colon E \to O \quad \text{and} \quad s'\colon O \to E.$$

Example 1.1.6 (Quivers—NOUGAT). [ONEGAT] The GAT $\mathfrak{Quiv}$ of **quivers** is given by

LEAN—NOUGAT

```
def 𝔔uiv : GAT := {[
    V : U,
    E : V ⇒ V ⇒ U
]}
```

Quiv is the category whose
- objects are *quivers*/*directed multigraphs*: a set V of "vertices" and a function E assigning to every $v_0, v_1\colon V$ a set $E(v_0, v_1)$ of "edges from v_0 to v_1";

- morphisms $F\colon (V, E) \to (V', E')$ consist of two components

$$F_{\#1}\colon V \to V'$$
$$F_{\#2}\colon \{v_0\ v_1 : V\} \to E(v_0, v_1) \to E'(F_{\#1}\ v_0, F_{\#1}\ v_1)$$

The GATs $\mathfrak{N}$ and $\mathfrak{EO}$ feature *elements depending on elements*, e.g. for each element `t` of the abstract type `Nat`, there is another element `succ t`, also of `Nat`. The GAT $\mathfrak{Quiv}$ features a *sort depending on elements*: for any elements `v` and `v'` of `V`, we can apply the constructor `E` to get a new sort symbol `E v v'`. As we see, function constructors whose codomain is a prior sort symbol (like `succ`) are interpreted as functions on the underlying sets, whereas *sort-valued* functions, i.e. function constructors whose codomain is `U` are interpreted as families of sets.

Cartmell's original definition of GATs [Car86, pp. 223–227] used a framework of *well-formed inference rules*. We can view the nouGAT presentation as convenient syntax for writing such rules, where we use ⇒ to indicate dependency instead of the antecedent-consequent structure of an inference rule. For instance, the `succ'` constructor of $\mathfrak{EO}$ would be rendered as

$$\frac{\texttt{o}\colon \texttt{Odd}}{\texttt{succ'}\ \ \texttt{o}\colon \texttt{Even}}$$

and the `E` constructor of $\mathfrak{Quiv}$ would become

$$\frac{\texttt{v}\colon \texttt{V} \quad \texttt{v'}\colon \texttt{V}}{\texttt{E}\ \ \texttt{v}\ \ \texttt{v'}\colon \texttt{U}.}$$

We will use the rule syntax in a few places to more fully elaborate the meaning of the nouGAT syntax, but in general we prefer the programming-language-style presentation of nouGAT to the rule syntax. When the constructors of a GAT are listed sequentially, as they are in nouGAT, it's immediate to see what sort- and element-symbols are already in-scope: just those that occur above the current line. This is more convenient from a user's perspective, but also is superior technically: Cartmell's definition involves two stages, the definition of (possibly ill-formed) *pre-rules* and the subsequent restriction to those rules which are *well-formed*. The well-formedness condition essentially amounted to requiring that the relevant symbols were in scope when used, and were used appropriately to their type. By contrast, oneGAT(and, by extension, nouGAT) is *intrinsically well-formed*: any GAT we can express will be already well-formed by construction, and there is no need to perform a two-stage construction.

In the above examples, all the function constructors were simple, i.e. *non-dependent* functions. But we also allow function constructors whose codomain depends on their argument, such as the following.

Lean—nouGAT

```
def 𝔯Quiv : GAT := {[
    V : U,
    E : V ⇒ V ⇒ U,
    r : (v : V) ⇒ E v v
]}
```

The r constructor gives us an element of the sort E v v for every element v of V. In the rule syntax,

$$\frac{\mathtt{v} : \mathtt{V}}{\mathtt{r}\ \mathtt{v} : \mathtt{E}\ \mathtt{v}\ \mathtt{v}.}$$

In the algebra, this is interpreted as equipping the $\mathfrak{Quiv}$-algebra (V, E) with a *dependent function* r, producing for each $v : V$ some $r(v) : E(v, v)$. Of course, a homomorphism of $\mathfrak{rQuiv}$-algebras, $(F_{\#1}, F_{\#2}, F_{\#3}) : (V, E, r) \to (V', E', r')$, must commute with these dependent functions in the appropriate way: the third component, $F_{\#3}$ consists of the requirement that for all $v : V$, $r'(F_{\#1}\ v) = F_{\#2}(r\ v)$.

Notice that $\mathfrak{rQuiv}$ is an *extension* of $\mathfrak{Quiv}$ by one additional constructor; consequently, $\mathfrak{rQuiv}$-algebras consist of $\mathfrak{Quiv}$-algebras equipped with an additional piece of data and $\mathfrak{rQuiv}$-algebra homomorphisms are just $\mathfrak{Quiv}$-algebra homomorphisms respecting that additional data. This is a very typical situation in generalized algebra. As our GATs get longer, it will be convenient to have syntax for denoting this. Therefore, our "official" declaration of $\mathfrak{rQuiv}$ will make use of the following syntactic sugar.

Example 1.1.7 (Reflexive Quivers—NOUGAT)**.** [ONEGAT] The GAT of *reflexive quivers* is given by

LEAN—NOUGAT

```
def rQuiv : GAT := {{
  include Quiv;
    r : (v : V) ⇒ E v v
}}
```

The `include` syntax means just what users of a modular programming language are accustomed to it meaning: add all the declarations from the imported structure (in this case, the GAT $\mathfrak{Quiv}$) in order, so they're in scope for subsequent declarations.

It's important to point out what we're *not* allowed to write in a GAT. The most significant restriction is that the domain of any $\Rightarrow$ must be a previously-declared sort. We cannot have 'sort combinators' (e.g. constructors of shape $\mathtt{U} \Rightarrow \mathtt{U}$) or 'choice functions' (e.g. constructors of shape $(\mathtt{N} : \mathtt{U}) \Rightarrow \mathtt{N}$, as U is not itself a sort; we cannot have higher-order functions, e.g. $(\mathtt{A} \Rightarrow \mathtt{B}) \Rightarrow \mathtt{C}$, because $\mathtt{A} \Rightarrow \mathtt{B}$ is not a *sort*, even if A and B are. In other words, $\Rightarrow$ functions must have "small" domain (a sort), but can have either small or "large" codomain. This restriction against higher-order functions is relaxed somewhat in the notion of *second-order GAT (SOGAT)* [Uem21], allowing for second-order functions (but not arbitrary higher-order), at the expense of a more complex model theory. For the key properties of GATs—especially the existence of initial algebras—to work as desired, we are obliged to obey these restrictions.

The final aspect of GATs to introduce are *equations*. GATs are, after all, intended to be *algebraic theories*, and a necessary aspect of defining algebraic structures is stipulating the laws which those structures must obey. Accordingly, GATs allow for constructors which equate elements of previously-defined sorts. Note that we do not permit equations between the sorts themselves (nor will we need to for the structures we wish to study); our notion of 'GAT' is therefore slightly stricter than that of Cartmell,

who does allow for sort equations.

Example 1.1.8. [oneGAT] The GAT $\mathfrak{Mon}$ of **monoids** is given by

Lean—nouGAT

```
def 𝔐on : GAT := {[
    M     : U,
    u     : M,
    m     : M ⇒ M ⇒ M,
    lunit : (x : M) ⇒ m u x ≡ x,
    runit : (x : M) ⇒ m x u ≡ x,
    assoc : (x y z : M) ⇒ m x (m y z) ≡ m (m x y) z
]}
```

Mon is the category whose

- objects are monoids: a set M equipped with an element $u\colon M$ and function $\mu\colon M \to M \to M$ satisfying the following equations for all $x, y, z\colon M$:
  - $\mu(u, x) = x$
  - $\mu(x, u) = x$
  - $\mu(x, \mu(y, z)) = \mu(\mu(x, y), z)$.
- morphisms (*monoid homomorphisms*) $(M, u, \mu) \to (N, v, \nu)$ are functions $\varphi\colon M \to N$ such that $\varphi(u) = v$ and such that $\varphi(\mu(x, y)) = \nu(\varphi\ x, \varphi\ y)$ for all $x, y\colon M$.

Example 1.1.9 (Groups—nouGAT). [oneGAT] The GAT $\mathfrak{Grp}$ of **groups** is given by

Lean—nouGAT

```
def 𝔊rp : GAT := {[
  include 𝔐on;
    inv   : M ⇒ M,
    linv  : (x : M) ⇒ m (inv x) x ≡ u,
    rinv  : (x : M) ⇒ m x (inv x) ≡ u
]}
```

Grp is the category whose

- objects are groups: a monoid (M, u, μ) equipped with an operation $i\colon M \to M$ satisfying the following equations for all $x\colon M$:
  - $\mu(i\ x, x) = u$
  - $\mu(x, i\ x) = u$
- morphisms (*group homomorphisms*) $(M, u, \mu, i) \to (N, v, \nu, j)$ are monoid homomorphisms $\varphi\colon (M, u, \mu) \to (N, v, \nu)$ such that

$$\varphi(i\ x) = j(\varphi\ x)$$

for all $x : M$.

Remark 1.1.10. Some of the requirements of 'group homomorphism' stated above are superfluous: as any text on basic group theory will remind us, the requirement that $\varphi(\mu(x,y)) = \nu(\varphi\ x, \varphi\ y)$ will, when combined with the other laws of groups, prove that φ preserves the unit element and commutes with the inverse operations. One consequence of these facts is that Grp is a **full subcategory** of Mon—every *monoid* homomorphism between two groups is already a *group* homomorphism. When defining structures in the informal mathematical style, we will frequently follow the standard practice and omit such superfluous conditions. Note we have already omitted from the definition of 'homomorphism' the requirement that φ preserve the *proofs* of the left- and right-unit laws, the proof of the associative law, etc.[a] This is because we are assuming the uniqueness of identity proofs (and function extensionality) in our metatheory, hence the proofs are unique anyways and preservation is automatic. A treatment of this topic that takes, say, homotopy type theory as its metatheory would need to carefully attend to this matter, but our assumption of UIP permits us to ignore it.

[a] For instance: the action of φ on the equation $\mathrm{lunit}_M(m)$ between $\mu(u,m)$ and m gives us an equation in N between $\varphi(\mu(u,m))$ and $\varphi(m)$. But we can also obtain an equation of this form from the equation $\mathrm{lunit}_N(\varphi(m)) : \nu(v,\varphi(m)) = \varphi(m)$, composed with the proof of $\varphi(\mu(u,m)) = \nu(v,\varphi(m))$ obtained from the proofs that $\varphi(u) = u$ and $\varphi(\mu(x,y)) = \nu(\varphi\ x, \varphi\ y)$. These ought to be the same.

We can also now introduce the GAT which will be our central focus in subsequent chapters: the GAT of *categories*. In this GAT (and henceforth), we enclose arguments in curly-braces to mark them as *implicit*: note that applications of `comp`, for example, only the two arguments are given (the `Mor J K` and `Mor I J` arguments), with the first three arguments (the `I`, `J`, and `K`) left to be inferred. Later, this will prevent the GAT signatures from getting intolerably unwieldy.

Example 1.1.11. The GAT $\mathfrak{Cat}$ of **categories** is given by

LEAN—NOUGAT

```
def Cat : GAT := {{
  include rQuiv as (Obj,Hom,id);
    comp  : {I J K : Obj} ⇒
            Hom J K ⇒ Hom I J ⇒ Hom I K,
    lunit : {I J : Obj} ⇒ (j : Hom I J) ⇒
            comp (id J) j ≡ j,
    runit : {I J : Obj} ⇒ (j : Hom I J) ⇒
            comp j (id I) ≡ j,
    assoc : {I J K L : Obj} ⇒ (j : Hom I J) ⇒
            (k : Hom J K) ⇒ ( ℓ : Hom K L) ⇒
            comp ℓ (comp k j) ≡ comp (comp ℓ k) j
}}
```

The type $\mathfrak{Cat}$-Alg is given in Figure 1.2: a category $\mathcal{C}$ consists of
- a set $|\mathcal{C}|$ of "objects";
- for any objects $I, J : |\mathcal{C}|$, a set $\mathcal{C}\ [I, J]$ of "morphisms from I to J";
- a given *identity morphism* $\mathsf{id}_I : \mathcal{C}\ [I, I]$ for each object I; and
- a *composition operator*

$$\_ \circ \_ : \mathcal{C}\ [J, K] \to \mathcal{C}\ [I, J] \to \mathcal{C}\ [I, K]$$

for all objects I, J, K such that
  - $f \circ \mathsf{id}_I = f$ and $\mathsf{id}_J \circ f = f$ for all $f : \mathcal{C}\ [I, J]$,
  - $h \circ (g \circ f) = (h \circ g) \circ f$ for all appropriate f, g, h.

The `as` notation allows us to rename components, in this case renaming the sort `V` to `Obj`, `E` to `Hom`, and `r` to `id`.

pseudoAgda

```
record 𝔠𝔞𝔱-Alg : Set where
  Obj : Set
  Hom : Obj → Obj → Set
  id : (I : Obj) → Hom I I
  comp : {I J K : Obj} → Hom J K → Hom I J → Hom I K
  idr : {I J : Obj} → (j : Hom I J) → comp (id J) j = j
  idl : {I J : Obj} → (j : Hom I J) → comp j (id I) = j
  ass : {I J K L : Obj} → (j : Hom I J) → (k : Hom J K) → (ℓ : Hom K L)
    →
    comp ℓ (comp k j) = comp (comp ℓ k) j
```

Figure 1.2: pseudoAgda definition of the type $\mathfrak{Cat}$-Alg.

Example 1.1.12. The morphisms of the category Cat from $\mathcal{C}$ to $\mathcal{D}$ are **functors**, which consist of two components

$$F_{\#1} : |\mathcal{C}| \to |\mathcal{D}|$$
$$F_{\#2} : \{I\ J : |\mathcal{C}|\} \to \mathcal{C}\ [I, J] \to \mathcal{D}\ [F_{\#1}\ I, F_{\#1}\ J]$$

such that
- $F_{\#2}(\mathsf{id}_I) = \mathsf{id}_{F_{\#1}\ I}$ for all I; and
- $F_{\#2}(k \circ j) = (F_{\#2}\ k) \circ (F_{\#2}\ j)$ for all k, j.

Analogously to the case of monoids and groups, we have a notion of 'category equipped with inverse operations'—the GAT of *groupoids*—whose category of algebras is a full subcategory of Cat.

Example 1.1.13. The GAT $\mathfrak{Grpd}$ of **groupoids** is given by

LEAN–NOUGAT

```
def Grpd : GAT := {[
  include Cat;
    inv  :  {I J : Obj} ⇒ Hom I J ⇒ Hom J I,
    linv :  {I J : Obj} ⇒ (j : Hom I J) ⇒
            comp (inv j) j ≡ id I,
    rinv :  {I J : Obj} ⇒ (j : Hom I J) ⇒
            comp j (inv j) ≡ id J
]}
```

A $\mathfrak{Grpd}$-algebra, i.e. a groupoid, is a category $\mathcal{C}$ equipped with an operation $(\_)^{-1} : \mathcal{C}\ [I, J] \to \mathcal{C}\ [J, I]$ satisfying the following equations for all $j : \mathcal{C}\ [I, J]$:

- $j^{-1} \circ j = \mathrm{id}_I$
- $j \circ j^{-1} = \mathrm{id}_J$.

Our other major use of equations in a GAT will be to *truncate* a sort, i.e. stipulate that all elements of that sort are equal. This allows us to use the sort as a *proposition*, as in the following examples.

Example 1.1.14. The GAT $\mathfrak{PreOrd}$ of **preorders** is given by

LEAN–NOUGAT

```
def PreOrd : GAT := {[
  include rQuiv as (X,leq,_);
    leqη : {x x' : X} ⇒ {p q : leq x x'} ⇒ p ≡ q,
    trns : {x y z : X} ⇒
      leq x y ⇒ leq y z ⇒ leq x z
]}
```

PreOrd is the category whose

- objects are preorders: a set X equipped with an binary relation $\_ \leq \_ : X \to X \to \mathrm{Prop}$ such that, for all $x, y, z : X$,
  - $x \leq x$
  - if $x \leq y$ and $y \leq z$, then $x \leq z$;
- morphisms (*monotone maps*) $(X, \leq_X) \to (Y, \leq_Y)$ are functions $f : X \to Y$ such that $x \leq_X x'$ implies $f(x) \leq_X f(x')$ for all x, x'.

Example 1.1.15. The GAT $\mathfrak{Setoid}$ of **setoids** is given by

Lean–nouGAT

```
def Setoid : GAT := {[
  include PreOrd as (X,eq);
    sym : {x y : X} ⇒ eq x y ⇒ eq y x
]}
```

A Setoid-algebra, i.e. a setoid, is a preorder $(X, \sim)$ such that, for all x, y,

$$x \sim y \quad \text{implies} \quad y \sim x.$$

Let us also briefly mention that the equalities in these GAT signatures come equipped with a *transport* operation, allowing us to express equalities *over* previous equalities. The examples in this section are too simple for this to be relevant, but in the next section we will need to do this for our GAT signatures to be well-formed. We'll denote this as in the following toy example.

```
A : U
a₀ : A
a₁ : A
eq : a₀ ≡ a₁
B : A ⇒ U
b : (a : A) ⇒ B a
beq : (b a₀ #⟨ eq ⟩) ≡ b a₁
```

In this example, we want to assert an equality between b a_0 and b a_1. However, this is ill-typed because they are elements of different sorts, B a_0 and B a_1 respectively. Thus, we have to transport b a_0 along the equality eq in order for it to be an element of the sort B a_1. As we'll discuss later (and is addressed in [KKA19, p. 12]), this transport operation is imported from the metatheory, since the equality types of oneGAT are extensional. Therefore, when interpreting these signatures into Set, i.e. defining the notion of 'algebra', the ≡ equalities become metatheoretic equalities. Thus there is no need to make the transports explicit in the definition of algebra: the type of algebras for the signature above would be

$$\begin{aligned}(A\colon \mathrm{Set}) &\times (a_0\ a_1 : A)\\ &\times (a_0 = a_1)\\ &\times (B\colon A \to \mathrm{Set})\\ &\times (b\colon (a\colon A) \to B(a))\\ &\times b(a_0) = b(a_1)\end{aligned}$$

so, since $a_0 = a_1$, the last equation is well-formed already.

A basic fact about GATs is that they are *finitary*: each of the above nouGAT declarations must necessarily consist of only finitely-many components. The syntax above doesn't allow us to declare *component schemes*, i.e. families of components indexed by, say, (metatheoretic) natural numbers. It turns out that there's no essential obstacle to doing so: the difference between our GAT signature language oneGATand

the KKA signature language for QIITs (of which it is a fragment), is that the latter *does* permit *(dependent) functions with metatheoretic domain*, i.e. families of components indexed by sets in the metatheory (and identities among elements of those sets). The other constructions possible on GATs, in particular the construction of the initial algebra, can be made just as well for signatures of this form as for GATs. For instance, assuming the natural numbers $\mathbb{N}$ with their usual addition operator _ + _, KKA give the usual definition of the set $\mathbb{Z}$ of integers as a quotient of $\mathbb{N} \times \mathbb{N}$ [KKA19, pp. 4–5], which might be rendered in NOUGAT-like syntax as follows.

LEAN–NOUGAT

```
def ℨ : iGAT [N] := {⟦
    Int    : U,
    subtract  : N ⇒̂ N ⇒̂ Int,
    eq     : (a b c d : N) ⇒̂ (a + d = b + c) ⇒̂
        Eq (subtract a b) (subtract c d)
⟧}
```

The initial algebra for this signature is, of course, the set $\mathbb{Z}$ of integers, with the `subtract` function being the subtraction operation $\mathbb{N} \to \mathbb{N} \to \mathbb{Z}$. The name `iGAT` here is short for "indexed GAT" (or perhaps "infinitary GAT"), but is the same thing as a *quotient inductive-inductive type* signature. The $\hat{\Rightarrow}$ arrows are not the same as the $\Rightarrow$ arrows from before: these have $\mathbb{N}$ and (metatheoretic) equalities of natural numbers as their domain, rather than a previously-declared sort. Quoting KKA (Ibid. p. 4), "An $\Rightarrow$ function has small domain and large codomain, while $\hat{\Rightarrow}$ has metatheoretic domain and large codomain." For the most part, finitary GATs will suffice for our purposes, and so for the rest of this chapter we'll just concern ourselves with them. But we will *technically* need the full power of iGATs/QIIT signatures in section 2.3; we therefore refer the reader to KKA for the appropriate definitions.

We conclude this section with a brief consideration of initial algebras. The natural numbers, i.e. the initial $\mathfrak{N}$-algebra, are once again an instructive example. While an arbitrary $\mathfrak{N}$-algebra might contain all kinds of junk, we want *the* natural numbers to *only* consist of $zero$, $succ(zero)$, $succ(succ(zero))$, and so on. To put it differently: we want the set $\mathbb{N}$ of natural numbers to consist of *only* those elements *whose existence can be deduced purely from the concept* $\mathfrak{N}$. We'll make this more precise in the following way. Think of the signature $\mathfrak{N}$ as a *context* of a type-theoretic language; what terms-in-context t of the form

$$\{\!|\texttt{Nat : U, zero : Nat, succ : Nat} \Rightarrow \texttt{Nat}|\!\} \vdash \texttt{t} : \texttt{Nat}$$

are we able to write? This turns out to be exactly the natural numbers: `Nat` is a completely *abstract*[4] type; we don't "know" anything about it *except* that it comes equipped with `zero` and `succ`. Therefore the only such terms-in-context are `zero`, `succ(zero)`, etc. This is exactly the intended meaning behind the **data** keyword in Figure 1.1: the type is *freely generated* by the constructors, containing neither junk

[4] In precisely the sense of [Rey83].

(elements besides those constructible from zero and succ) nor noise (relations among the elements besides those which can be deduced from the definition). From here, the initiality is more-or-less straightforward: if everything in $\mathbb{N}$ follows from the $\mathfrak{N}$ concept, then naturally they must have an interpretation in any implementation of the $\mathfrak{N}$ concept, that is, in any $\mathfrak{N}$-algebra.

Though we'll need to wait until we define the oneGAT signature language in section 1.3 to make precise what "terms-in-context" are, this provides the template for how we'll construct initial algebras for an arbitrary GAT $\mathfrak{G}$: the set interpreting a sort $\mathtt{T}$ (such as $\mathtt{Nat}$) will be the terms-in-context

$$\mathfrak{G} \vdash \mathtt{t} : \mathtt{T}$$

and the elements (like $\mathtt{zero}$) will be interpreted as themselves ($\mathfrak{N} \vdash \mathtt{zero} : \mathtt{Nat}$). As an illustrative example, let's consider monoids. There is certainly at least one element of the initial monoid, $\mathbb{1}$, since the signature stipulates the identity element $\mathtt{u}$:

$$\mathfrak{Mon} \vdash \mathtt{u} : \mathtt{M}.$$

But, upon further reflection, this is all: though the operation $\mathtt{m} : \mathtt{M} \Rightarrow \mathtt{M} \Rightarrow \mathtt{M}$ might seem like it could produce new elements of $\mathbb{1}$ like $\mathtt{succ}$ does for $\mathbb{N}$, the only place we can start is by considering $\mathtt{m}\ \ \mathtt{u}\ \ \mathtt{u}$, which the equations stipulate gives us back $\mathtt{u}$ again. Once again, since the carrier type $\mathtt{M}$ is abstract, we don't have any other means of producing terms of type $\mathtt{M}$ besides what's in the signature, and $\mathtt{u}$ is apparently the only such term. At this stage, we're just considering things informally, but once we precisely define oneGAT, the construction of initial algebras, and state the appropriate principle of induction/unary parametricity, then we will be able to prove rigorously that $\mathbb{1}$ is indeed a singleton set.

1.2 The Semantics of Type Theory as a GAT

As discussed, we are especially interested in the construction of initial algebras for an arbitrary GAT because it means that any notion of 'model of type theory' which we can express as a GAT will then automatically admit an initial model, the *syntax model*. We therefore introduce our GAT notion of model of type theory: *categories with families*. As is well-known in the literature [ALN25], the same data that constitutes a CwF can be packaged in myriad forms, allowing one to study the semantics of type theory within a preferred mathematical framework. For instance, a pure category theorist might find the notion of *split comprehension category* an attractive presentation, as it defines 'model of type theory' in terms of Grothendieck fibrations and cartesian morphisms; an algebraic geometer (specifically someone well-versed in the theory of *stacks*) might instead prefer to work with *natural models* [Awo18]; and so on. Part of what sets CwFs[5] apart, we claim, is that they provide the most *elementary* presentation of this common data: though we make use of some small amount of category theory terminology in Example 1.2.2, CwFs are, for us, ultimately the algebras of a GAT $\mathfrak{CwF}$. As we saw in the previous section, algebras for GATs are defined in the most rudimentary

[5] At least as presented here.

terms: sets, elements of sets, and functions on those sets. The presentation of $\mathfrak{CwF}$-Alg given in Figure 1.3 provides us with the same notion of 'model of type theory' as split comprehension categories, natural models, and so on, but fully "unboxed", all the "packaging" removed, nothing but the actual content. Undoubtedly, the models of directed type theory pursued in subsequent chapters (and expressed as extensions of $\mathfrak{CwF}$) could be packaged up more conveniently to fit this or that mathematical framework. We leave such considerations to future work: here, we aspire to present the bare heart of the matter in full explicit detail, that is, as a GAT.

1.2.1 Categories with Families

The notion of CwF mathematically captures the fundamental structural dynamics of dependent type theory. In type theory, we are concerned with forming **terms**, each of which must have a specified **type**. We do so with the help of typed variable declarations available from the **context**, and we can transport the terms and types between contexts by way of **substitutions**. These are the four kinds of "stuff" in type theory, and will be the four sort components of the GAT of CwFs—the rest of the components are there to abstractly characterize how these interact.

In the syntax of type theory, our *contexts* are finite lists of typed variable declarations, i.e. lists of the form

$$x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$$

where the type A_2 could possibly contain the variable x_1, the type A_3 could possibly contain the variables x_1 and x_2, and so on. A *substitution* from $\Delta = (y_1 : B_1, \ldots, y_m : B_m)$ to $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ is an "implementation" of the variables of Γ using the variables of Δ, that is, a finite list of terms $(t_1, \ldots, t_n)$, each of which may contain the variables $y_1, \ldots, y_m$ and where

- $t_1 : A_1$,
- $t_2 : A_2[t_1/x_1]$ (i.e. the type A_2 with any occurrences of x_1 replaced by t_1)
- $t_3 : A_3[t_1/x_1, t_2/x_2]$

and so on. Now, to make this precise would bring in a lot of cumbersome details, such as the concern that the variable names x_i are all distinct. We want our model notion to abstract away from the syntactic minutiae of variable names, capture, scope, α-equivalence and so on. This is what is accomplished in the notion of CwF: abstractly characterizing the fact that contexts and substitutions are built up inductively as structured lists, and that the variables contained in a context can be used to construct terms and types in that context.

Example 1.2.1. The GAT of *categories with families* is given by

Lean–nouGAT

```
def 𝔈𝔴𝔉 : GAT := {[
  include 𝔈𝔞𝔱 as (Con,Sub,comp,id,_,_,_);
    empty    : Con,
    ϵ        : (Γ : Con) ⇒ Sub Γ empty,
    ϵ_η      : (Γ : Con) ⇒ (f : Sub Γ empty) ⇒
               f ≡ (ϵ Γ),
    Ty       : Con ⇒ U,
    substTy  : {Δ Γ : Con} ⇒ Sub Δ Γ ⇒ Ty Γ ⇒ Ty Δ,
    idTy     : {Γ : Con} ⇒ (A : Ty Γ) ⇒
               substTy (id Γ) A ≡ A,
    compTy   : {Θ Δ Γ : Con} ⇒ (A : Ty Γ)
               (δ : Sub Θ Δ) ⇒ (γ : Sub Δ Γ) ⇒
               substTy γ (substTy δ A)
               ≡ substTy (comp γ δ) A,
    Tm       : (Γ : Con) ⇒ Ty Γ ⇒ U,
    substTm  : {Δ Γ : Con} ⇒ {A : Ty Γ} ⇒
               (γ : Sub Δ Γ) ⇒
               Tm Γ A ⇒ Tm Δ (substTy γ A),
    idTm     : {Γ : Con} ⇒ {A : Ty Γ} ⇒ (t : Tm Γ A)
               substTm (id Γ) t       #⟨idTy A⟩
               ≡ t,
    compTm   : {Θ Δ Γ : Con} ⇒
               {A : Ty Γ} ⇒ (t : Tm Γ A) ⇒
               (δ : Sub Θ Δ) ⇒ (γ : Sub Δ Γ) ⇒
               substTm γ (substTm δ t)
                   #⟨compTy A γ δ⟩
               ≡ substTm (comp γ δ) t,
```

LEAN—NOUGAT

```
    ext      : (Γ : Con) ⇒ Ty Γ ⇒ Con,
    pair     : {Δ Γ : Con} ⇒ {A : Ty Γ} ⇒
               (γ : Sub Δ Γ) ⇒
               Tm Δ (substTy γ A) ⇒
               Sub Δ (ext Γ A),
    pair_nat: {Θ Δ Γ : Con} ⇒ {A : Ty Γ} ⇒
               (γ : Sub Δ Γ) ⇒
               (t : Tm Δ (substTy γ A)) ⇒
               (δ : Sub Θ Δ) ⇒
               comp (pair γ t) δ
               ≡ pair (comp γ δ)
                   (substTm δ t  #⟨compTy A γ δ⟩),
    p        : {Γ : Con} ⇒ (A : Ty Γ) ⇒
               Sub (ext Γ A) Γ
    v        : {Γ : Con} ⇒ (A : Ty Γ) ⇒
               Tm (ext Γ A) (substTy (p A) A),
    ext_β₁   : (Δ Γ : Con) ⇒ (A : Ty Γ) ⇒
               (γ : Sub Δ Γ) ⇒
               (t : Tm Δ (substTy γ A)) ⇒
               comp (p A) (pair γ t) ≡ γ,
    ext_β₂   : (Δ Γ : Con) ⇒ (A : Ty Γ) ⇒
               (γ : Sub Δ Γ) ⇒
               (t : Tm Δ (substTy γ A)) ⇒
               substTm (pair γ t) (v A)
                   #⟨compTy A (p A) (pair γ t)⟩
                   #⟨ext_β₁ γ t⟩
               ≡ t,
    ext_η    : (Γ : Con) ⇒ (A : Ty Γ) ⇒
               pair (p A) (v A)
               ≡ id (ext Γ A)
⟧
```

Example 1.2.2. The type of **categories with families (CwFs)**—the algebras for $\mathfrak{CwF}$—consists of the following data (spelled out more explicitly in Figure 1.3).

- A category whose objects are called **contexts** $\Delta, \Gamma : \mathsf{Con}$ and whose morphisms are called **substitutions** $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$;
- A terminal object $\bullet : \mathsf{Con}$, the **empty context**;
- A set $\mathsf{Ty}\ \Gamma$ of **types** for each context Γ, equipped with an operation

$$(A : \mathsf{Ty}\ \Gamma),\ (\sigma : \mathsf{Sub}\ \Delta\ \Gamma) \quad \mapsto \quad A[\sigma] : \mathsf{Ty}\ \Delta$$

  such that $A[\mathrm{id}_\Gamma] = A$ and $A[\gamma][\delta] = A[\gamma \circ \delta]$;
- A set $\mathsf{Tm}(\Gamma, A)$ of **terms** for each $\Gamma : \mathsf{Con}$ and $A : \mathsf{Ty}\ \Gamma$, equipped with an

operation

$$(t : \mathsf{Tm}(\Gamma, A)),\ (\sigma : \mathsf{Sub}\ \Delta\ \Gamma) \quad \mapsto \quad t[\sigma] : \mathsf{Tm}(\Delta, A[\sigma])$$

such that $t[\mathrm{id}_\Gamma] = t$ and $t[\gamma][\delta] = t[\gamma \circ \delta]$;

- An operation of **context extension**

$$(\Gamma : \mathsf{Con}),\ (A : \mathsf{Ty}\ \Gamma) \quad \mapsto \quad \Gamma \rhd A : \mathsf{Con}$$

equipped with the **weakening substitution** and **variable term**

$$\mathsf{p}_A : \mathsf{Sub}\ (\Gamma \rhd A)\ \Gamma \qquad \text{and} \qquad \mathsf{v}_A : \mathsf{Tm}(\Gamma \rhd A, A[\mathsf{p}_A])$$

and **pairing operation**

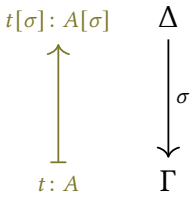
$$(\sigma : \mathsf{Sub}\ \Delta\ \Gamma),\ (t : \mathsf{Tm}(\Delta, A[\sigma])) \quad \mapsto \quad (\sigma, t) : \mathsf{Sub}\ \Delta\ \Gamma \rhd A$$

satisfying the laws

$$\begin{aligned} \mathsf{p} \circ (\sigma, t) &= \sigma \\ \mathsf{v}[\sigma, t] &= t \\ (\sigma, t) \circ \delta &= (\sigma \circ \delta, t[\delta]) \\ (\mathsf{p}, \mathsf{v}) &= \mathsf{id}. \end{aligned}$$

As here, we'll just write p and v when the type can be inferred.

We'll sometimes depict the substitution operation like this:



CwFs, in our view, sit in the "Goldilocks zone" of abstraction: they are far enough abstracted from the syntax to free us from its cumbersome bureaucracy, but not so abstract that the connection to syntax is obscured. With other, more mathematically-oriented notions of "model", the terms and types are cleverly encoded as part of the mathematical structure; often considerable work is needed to recognize the contours of type theory within that structure. We delight in *not* being clever: the types in context Γ are the elements of the set $\mathsf{Ty}\ \Gamma$, and the terms in Γ of type A are elements of the set $\mathsf{Tm}(\Gamma, A)$.

Let us make some comments on the components of a CwF, starting with the substitution p. Thinking again of contexts as lists of variable declarations, we understand this substitution as simply 'forgetting' the last variable in the context $\Gamma \rhd A$, but leaving the remainder of the variables in Γ untouched. This is exemplified by the following tactic, of which we will make frequent use.

pseudoAgda

```
record 𝔈𝔴𝔉-Alg where
  Con : Set
  Sub : Con → Con → Set
  id : (X : Con) → Sub X X
  comp : {X Y Z : Con} → Sub Y Z → Sub X Y → Sub X Z
  lunit : {X Y : Con} → (f : Sub X Y) → comp (id Y) f = f
  runit : {X Y : Con} → (f : Sub X Y) → comp f (id X) = f
  assoc : {W X Y Z : Con} →
    (e : Sub W X) → (f : Sub X Y) → (g : Sub Y Z) →
    comp g (comp f e) = comp (comp g f) e
  empty : Con
  ϵ : (Γ : Con) → Sub Γ empty
  ηϵ : {Γ : Con} → (f : Sub Γ empty) → f = ϵ Γ
  Ty : Con → Set
  substTy : {Δ Γ : Con} → Sub Δ Γ → Ty Γ → Ty Δ
  idTy : {Γ : Con} → (A : Ty Γ) → substTy (id Γ) A = A
  compTy : {Θ Δ Γ : Con} → (A : Ty Γ) →
    (δ : Sub Θ Δ) → (γ : Sub Δ Γ) →
    substTy γ (substTy δ A) = substTy (comp γ δ) A
  Tm : (Γ : Con) → Ty Γ → Set
  substTm : {Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → Tm Γ A → Tm Δ (substTy γ A)
  idTm : {Γ : Con}{A : Ty Γ} → (t : Tm Γ A) → substTm (id Γ) t = t
  compTm : {Θ Δ Γ : Con}{A : Ty Γ} → (t : Tm Γ A) →
    (δ : Sub Θ Δ) → (γ : Sub Δ Γ) →
    substTm γ (substTm δ t) = substTm (comp γ δ) t
  ext : (Γ : Con) → Ty Γ → Con
  pair : {Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → Tm Δ (substTy γ A) → Sub Δ (ext Γ A)
  pair_nat : {Θ Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → (t : Tm Δ (substTy γ A)) → (δ : Sub Θ Δ) →
    comp (pair γ t) δ = pair (comp γ δ) (substTm δ t)
  p : {Γ : Con} → (A : Ty Γ) → Sub (ext Γ A) Γ
  v : {Γ : Con} → (A : Ty Γ) → Tm (ext Γ A) (substTy (p A) A)
  ext_β₁ : {Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → (t : Tm Δ (substTy γ A)) →
    comp (p A) (pair γ t) = γ
  ext_β₂ : {Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → (t : Tm Δ (substTy γ A)) →
    substTm (pair γ t) (v A) = t
  ext_η : {Γ : Con}{A : Ty Γ} → pair (p A) (v A) = id (ext Γ A)
```

Figure 1.3: pseudoAgda presentation of $\mathfrak{Cw}\mathfrak{F}$-Alg.

Lemma 1.2.3. *There is a bijection between terms in* Γ *of type* A *and sections of* p_A*:*

$$\mathsf{Tm}(\Gamma, A) \underset{\mathsf{v}[\_]}{\overset{(\mathsf{id}_\Gamma, \_)}{\rightleftarrows}} (\tau \colon \mathsf{Sub}\ \Gamma\ (\Gamma \rhd A)) \times \mathsf{p} \circ \tau = \mathsf{id}_\Gamma.$$

In the abstract presentation, there is no ontological distinction between *variables* and *terms*: variables are just particular terms, the ones obtained directly from the context rather than constructed by way of term-formers. The variable terms do not have explicit names which can be shadowed, captured, fresh, etc., but are rather abstractly represented as *de Bruijn indices*.

Definition 1.2.4. For a given CwF, define the **de Bruijn indices** as terms v_n for $n \colon \mathbb{N}$ by:

$$\begin{aligned}
&\mathsf{v}_0\ \{\Gamma\}\{A_0\} && : \mathsf{Tm}(\Gamma \rhd A_0, A_0[\mathsf{p}]) \\
&\mathsf{v}_0\ \{\Gamma\}\{A_0\} && := \mathsf{v}_{A_0} \\
&\mathsf{v}_{n+1}\{\Gamma\}\{A_0\ \ldots\ A_{n+1}\} && : \mathsf{Tm}(\Gamma \rhd A_0 \rhd \cdots \rhd A_{n+1}, A_0[\mathsf{p} \circ \cdots \circ \mathsf{p}]) \\
&\mathsf{v}_{n+1}\{\Gamma\}\{A_0\ \ldots\ A_{n+1}\} && := (\mathsf{v}_n\ \{\Gamma\}\ \{A_0\ \ldots\ A_n\})[\mathsf{p}_{A_{n+1}}]
\end{aligned}$$

In order for $\Gamma \rhd A_0 \rhd \cdots \rhd A_{n+1}$ to be well-formed, the list of types $A_0\ \ldots\ A_{n+1}$ must be of a particular form:

$$\begin{aligned}
A_0 &\colon \mathsf{Ty}\ \Gamma \\
A_1 &\colon \mathsf{Ty}\ (\Gamma \rhd A_0) \\
&\vdots \\
A_{n+1} &\colon \mathsf{Ty}\ (\Gamma \rhd A_0 \rhd \cdots \rhd A_n).
\end{aligned}$$

A list of this form is what we'll often refer to as a **telescope over** Γ.

This notion of 'telescope' allows us to more conveniently state the following notion, which will be useful at various points of our development. It captures the idea of a substitution between $\Delta \rhd \cdots$ and $\Gamma \rhd \cdots$ which only substitutes terms in Δ for the variables in Γ, and leaves further variables otherwise unchanged.

Definition 1.2.5. Suppose $\sigma \colon \mathsf{Sub}\ \Delta\ \Gamma$ is some substitution and $A_0\ \ldots\ A_n$ is some telescope over Γ. Then we define the substitution

$$\begin{aligned}
\mathsf{q}(\sigma; A_0, \ldots, A_n) \quad : \quad & \mathsf{Sub}\ (\Delta \rhd A_0[\sigma] \rhd \cdots \rhd A_n[\mathsf{q}(\sigma; A_0, \ldots, A_{n-1})]) \\
& (\Gamma \rhd A_0 \rhd \cdots \rhd A_n).
\end{aligned}$$

Informally, we define $\mathsf{q}(\sigma; A_0, \ldots, A_n)$ by

$$((\sigma \circ \underbrace{\mathsf{p} \circ \cdots \circ \mathsf{p}}_{n+1}), \mathsf{v}_n, \ldots, \mathsf{v}_0).$$

But, more precisely, by induction on n: for $n = 0$, put

$$\mathsf{q}(\sigma; A_0) := (\sigma \circ \mathsf{p},\ \mathsf{v}_{A_0[\sigma]}) \quad : \mathsf{Sub}\ (\Delta \rhd A_0[\sigma])\ (\Gamma \rhd A_0).$$

Then, given $\mathsf{q}(\sigma; A_0, \dots, A_n)$, define

$$\mathsf{q}(\sigma; A_0, \dots, A_{n+1}) := (\mathsf{q}(\sigma; A_0, \dots, A_n) \circ \mathsf{p},\ \mathsf{v}_{A_{n+1}[\mathsf{q}(\sigma; A_0, \dots, A_n)]}).$$

When the telescope $A_0\ \dots\ A_n$ is clear from context, we may just write $\mathsf{q}(\sigma)$.

We'll encounter our main examples of CwFs—concrete CwFs, fibrant sub-CwFs of concrete CwFs, and syntax models—later in this chapter. But, for the sake of having a fruitful example at the present moment, we'll introduce one of the most important varieties of CwF: *presheaf models of type theory* [Hof97, Sect. 4]. To do so, we first recall the following definition.

Definition 1.2.6. Given a category $\mathcal{C}$ and a presheaf $F : \mathcal{C}^{\mathrm{op}} \Rightarrow \mathsf{Set}$, define the **category of elements** of F—denoted $\int F$—as follows.

$$|\textstyle\int F| := (I : |\mathcal{C}|) \times F\ I$$
$$(\textstyle\int F)[(I, x), (J, y)] := (j : \mathcal{C}[I, J]) \times (F\ j\ y = x)$$

Example 1.2.7. For any category $\mathcal{C}$, we have the **presheaf model** (on $\mathcal{C}$), which is the CwF whose

- contexts are presheaves $\Gamma : \mathcal{C}^{\mathrm{op}} \Rightarrow \mathsf{Set}$, with the empty context being the constant $\{\star\}$ presheaf;
- substitutions $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$ are natural transformations $\Delta \to \Gamma$;
- types $A : \mathsf{Ty}\ \ \Gamma$ are presheaves on the category of elements of Γ, i.e. $A : (\int \Gamma)^{\mathrm{op}} \Rightarrow \mathsf{Set}$, with $A[\sigma]$ defined by $A[\sigma]\ (I, \delta) := A(I, \sigma_I\ \delta)$;
- terms $t : \mathsf{Tm}(\Gamma, A)$ are operations assigning to each $I : |\mathcal{C}|$ and $\gamma : \Gamma(I)$ an element of the set $A(I, \gamma)$, in such a way that

  $$A\ i\ (t(I, \gamma)) = t(J, \Gamma\ i\ \gamma)$$

  for every $i : \mathcal{C}\ [J, I]$;
- extended context $\Gamma \rhd A$ is the presheaf sending $I : |\mathcal{C}|$ to the set $(\gamma : \Gamma\ I) \times A(I, \gamma)$.

As we'll see throughout this section, presheaf CwFs automatically come equipped with several important type-theoretic constructs, even when the "base category" $\mathcal{C}$ is completely arbitrary. But many of the most fruitful uses of presheaf models in the literature assume further structure on $\mathcal{C}$, and import that structure into the type theory the presheaf model interprets. One prominent example is the presheaf semantics of *cubical type theory* given in [BCH14][CCHM18, Sect. 8], which adds a special interval pre-type to type theory, justified by presheaf models on "cube categories".

1.2.2 Type Formers

The definition of 'CwF' characterizes the pure structure of type theory—contexts, types, terms, substitution, etc.—but none of the content. Indeed, the syntax model of this type theory, i.e. the initial $\mathfrak{CwF}$-algebra, is trivial: it consists of only the empty context $\bullet$, and there are no types, $\mathsf{Ty}\ \bullet = \emptyset$. This is because there's nothing in the GAT $\mathfrak{CwF}$ that actually allows us to produce types and terms; in order to have types and terms in our theory (and thereby have nontrivial syntax models), we must extend $\mathfrak{CwF}$ with type and term-formers. We don't undertake a detailed study of different type-theoretic constructs, but we rehearse a few canonical examples here for the sake of making our theory clear (and for useful contrast later on, when we develop similar types in directed type theory).

The simplest variety of type-formers are *finite types*, where we stipulate a type along with all of its terms. For instance, the two-element type, i.e. the type of *booleans*, is a feature in virtually every programming language, and has been put to use in countless ways. We develop the singleton *unit* type and the type of booleans in the present framework, for the sake of demonstration.

Definition 1.2.8. A CwF supports the **unit type** if it is equipped with the following data.

pseudoAgda

```
unit : {Γ : Con} ⟶ Ty Γ
unit[] : {Δ}{Γ}(σ : Sub Δ Γ) ⟶ unit [ σ ] = unit
star : {Γ : Con} ⟶ Tm(Γ, unit)
star[] : {Δ}{Γ}(σ : Sub Δ Γ) ⟶ star [ σ ] = star
unit-elim : {Γ}{M : Ty (Γ ▹ unit)} ⟶
  Tm(Γ, M [ id , star ]) ⟶ Tm(Γ ▹ unit, M)
unitβ : {Γ}{M}(m : Tm(Γ, M [ id , star ])) ⟶
  (unit-elim m)[ id , star ] = m
unitη : {Γ}{M}(m : Tm(Γ, M [ id , star ])) ⟶
  (z : Tm(Γ ▹ unit, M)) ⟶ (z [ id , star ] = m) ⟶
  z = unit-elim m
unit-elim[] : {Δ}{Γ}{A}{B}{M} ⟶
  (m : Tm(Γ, M[ id , star ])) ⟶ (σ : Sub Δ Γ) ⟶
  (unit-elim m)[ q(σ;unit) ] = unit-elim {Γ=Δ} (m [ σ ])
```

Definition 1.2.9. A CwF supports the **bool type** if it is equipped with the following data.

PSEUDOAGDA

```
bool : {Γ : Con} → Ty Γ
bool[] : {Δ}{Γ}(σ : Sub Δ Γ) → bool [ σ ] = bool
tt : {Γ : Con} → Tm(Γ, bool)
tt[] : {Δ}{Γ}(σ : Sub Δ Γ) → tt [ σ ] = tt
ff : {Γ : Con} → Tm(Γ, bool)
ff[] : {Δ}{Γ}(σ : Sub Δ Γ) → ff [ σ ] = ff
bool-elim : {Γ}{M : Ty (Γ ▷ bool)} →
  Tm(Γ, M [ id , tt ]) → Tm(Γ, M [ id , ff ]) →
  Tm(Γ ▷ bool, M)
boolβtt : {Γ}{M} →
  (mtt : Tm(Γ, M [ id , tt ])) → (mff : Tm(Γ, M [ id , ff ])) →
  (bool-elim mtt mff)[ id , tt ] = mtt
boolβff : {Γ}{M} →
  (mtt : Tm(Γ, M [ id , tt ])) → (mff : Tm(Γ, M [ id , ff ])) →
  (bool-elim mtt mff)[ id , ff ] = mff
boolη : {Γ}{M} →
  (mtt : Tm(Γ, M [ id , tt ])) → (mff : Tm(Γ, M [ id , ff ])) →
  (z : Tm(Γ ▷ bool, M)) →
  (z [ id , tt ] = mtt) → (z [ id , ff ] = mff) →
  z = bool-elim mtt mff
bool-elim[] : {Δ}{Γ}{A}{B}{M} →
  (mtt : Tm(Γ, M [ id , tt ])) → (mff : Tm(Γ, M [ id , ff ])) →
  (σ : Sub Δ Γ) →
  (bool-elim mtt mff)[ q(σ;bool) ]
  = bool-elim {Γ=Δ} (mtt [ σ ]) (mff [ σ ])
```

Remark 1.2.10. In the GAT signatures for $\mathfrak{CwF}_1$ and $\mathfrak{CwF}_2$ (Figure 1.4,Figure 1.5), we have omitted the unit−elim[] and bool−elim[] components, respectively. This is for reasons of simplicity: in order to get these equations to type-check, we need to perform a complex series of transports (using the prior equations `unit_stab` and `star_stab`, plus various properties of equality). As mentioned above, the ≡ equalities in the GAT get interpreted as metatheoretic equality of elements, so we can make these transports implicit in Definition 1.2.8 and Definition 1.2.9.

Example 1.2.11. The presheaf model (on any category $\mathcal{C}$) supports both unit and bool types: for any context $\Gamma : \mathcal{C}^{\mathrm{op}} \Rightarrow \mathrm{Set}$, define unit : $(\int\ \Gamma)^{\mathrm{op}} \Rightarrow \mathrm{Set}$ to be the constant-$\{\star\}$ presheaf. This is clearly stable under substitution by some natural transformation $\sigma : \mathrm{Sub}\ \Delta\ \Gamma$, and comes equipped with a term star defined by

$$\mathrm{star}\ (I : |\mathcal{C}|)\ (\gamma : \Gamma\ I) := \star : \mathrm{star}(I, \gamma).$$

This is, of course, the only such term; the natural transformation $(\mathrm{id}, \mathrm{star}) : \mathrm{Sub}\ \Gamma\ \Gamma \triangleright$

Lean–nouGAT

```
def 𝔈𝔴𝔉₁ : GAT := {[
  include 𝔈𝔴𝔉;
    unit : {Γ : Con} ⇒ Ty Γ,
    unit_stab : {Δ Γ : Con} ⇒ (σ : Sub Δ Γ) ⇒
        substTy σ unit ≡ unit,
    star : {Γ : Con} ⇒ Tm Γ unit,
    star_stab : {Δ Γ : Con} ⇒ (σ : Sub Δ Γ) ⇒
        substTm σ star   #⟨unit_stab σ⟩
        ≡ star,
    unit_elim : {Γ : Con} ⇒
        {M : Ty (ext Γ unit)} ⇒
        (m : Tm Γ (substTy (pair (id Γ) star) M)) ⇒
        Tm (ext Γ unit) M,
    unit_β : {Γ : Con} ⇒
        {M : Ty (ext Γ unit)} ⇒
        (m : Tm Γ (substTy (pair (id Γ) star) M)) ⇒
        substTm (pair (id Γ) star) (unit_elim m)
        ≡ m

]}
```

Figure 1.4: $\mathfrak{CwF}_1$, the GAT of CwFs supporting the unit type

unit is in fact a natural *isomorphism*, making it easy to define unit–elim.

The definition of bool is similar, using some two-element set $\{\top, \bot\}$ and defining tt and ff to be the constant-$\top$ and constant-$\bot$ global sections, respectively. The definition of bool–elim must be

$$\mathsf{bool\text{–}elim}\ I\ (\gamma, b) := \begin{cases} \mathsf{mtt}\ I\ \gamma & \text{if } b = \top \\ \mathsf{mff}\ I\ \gamma & \text{if } b = \bot. \end{cases}$$

These examples are relatively uninteresting from the standpoint of dependent type theory: both consist of a single type, uniformly asserted across all contexts. The unit[], star[], etc. laws essentially tell us that these types and terms don't meaningfully depend on the context. Indeed, these are the kinds of types which can exist in *simple type theory*, i.e. the simply typed lambda calculus. The following examples are *not* "simple" in this sense—they are properly *dependent* types.

In informal presentations of dependent type theory, Π-types are thought of as functions whose codomain (possibly) *depends* on the supplied argument: unlike an an ordinary function $f : A \to B$ which returns an element $f(t) : B$ for each $t : A$, a dependent function $f : \prod_{x:A} B(x)$ returns an element $f(x) : B(x)$, where now B is a family of types indexed over A. To write such a function by λ-abstraction, i.e. form $(\lambda x \to \Phi) : \prod_{x:A} B(x)$, we need to come up with an "expression" $\Phi : B(x)$ which

LEAN–NOUGAT

```
def 𝔈𝔴𝔉₂ : GAT := {[
  include 𝔈𝔴𝔉;
    bool : {Γ : Con} ⇒ Ty Γ,
    bool_stab : {Δ Γ : Con} ⇒ (σ : Sub Δ Γ) ⇒
        substTy σ bool ≡ bool,
    tt : {Γ : Con} ⇒ Tm Γ bool,
    tt_stab : {Δ Γ : Con} ⇒ (σ : Sub Δ Γ) ⇒
        substTm σ tt   #⟨bool_stab σ⟩
        ≡ tt,
    ff : {Γ : Con} ⇒ Tm Γ bool,
    ff_stab : {Δ Γ : Con} ⇒ (σ : Sub Δ Γ) ⇒
        substTm σ ff   #⟨bool_stab σ⟩
        ≡ ff,
    bool_elim : {Γ : Con} ⇒
        {M : Ty (ext Γ bool)} ⇒
        Tm Γ (substTy (pair (id Γ) tt) M) ⇒
        Tm Γ (substTy (pair (id Γ) ff) M) ⇒
        Tm (ext Γ bool) M,
    bool_β_tt : {Γ : Con} ⇒
        {M : Ty (ext Γ bool)} ⇒
        (mtt : Tm Γ (substTy (pair (id Γ) tt) M)) ⇒
        (mff : Tm Γ (substTy (pair (id Γ) ff) M)) ⇒
        substTm (pair (id Γ) tt) (bool_elim mtt mff)
        ≡ mtt,
    bool_β_ff : {Γ : Con} ⇒
        {M : Ty (ext Γ bool)} ⇒
        (mtt : Tm Γ (substTy (pair (id Γ) tt) M)) ⇒
        (mff : Tm Γ (substTy (pair (id Γ) ff) M)) ⇒
        substTm (pair (id Γ) ff) (bool_elim mtt mff)
        ≡ mff
]}
```

Figure 1.5: $\mathfrak{Cw}\mathfrak{F}_2$, the GAT of CwFs supporting the type of booleans

is allowed to "refer to a free variable" x of A. Of course, syntactic notions like "free variables" have been abstracted in our formalism of CwFs, replaced by context extension, de Bruijn indices, and the like.

Definition 1.2.12. A CwF supports Π-**types** if it is equipped with the following data.

pseudoAgda

```
Π   : {Γ : Con} ⟶ (A : Ty Γ) ⟶ Ty (Γ ▹ A) ⟶ Ty Γ
Π[] : {Δ Γ}{A}{B} ⟶ (σ : Sub Δ Γ) ⟶
  Π(A,B)[ σ ] = Π(A[ σ ], B [ q(σ;A) ])
lam  : {Γ}{A}{B} ⟶ Tm (Γ ▹ A, B) ⟶ Tm (Γ, Π(A,B))
app  : {Γ}{A}{B} ⟶ Tm (Γ, Π(A,B)) ⟶ Tm (Γ ▹ A, B)
lam[] : {Δ Γ}{A}{B}{t : Tm (Γ ▹ A, B)} ⟶ (σ : Sub Δ Γ) ⟶
  (lam t)[ σ ] = lam (t [ q(σ;A) ] )
Πβ  : {Γ}{A}{B} ⟶ (t : Tm (Γ ▹ A, B)) ⟶ app(lam t) = t
Πη  : {Γ}{A}{B} ⟶ (f : Tm (Γ, Π(A,B))) ⟶ lam(app f) = f
```

Similarly, the type $\sum_{x:\,A} B(x)$ can be defined for a family B of types indexed over A. This is the type of *dependent pairs* (a, b) where $b : B(a)$—generalizing the product type $A \times B$.

Definition 1.2.13. A CwF supports Σ-**types** if it is equipped with the following data.

pseudoAgda

```
Σ : {Γ : Con} ⟶ (A : Ty Γ) ⟶ Ty (Γ ▹ A) ⟶ Ty Γ
Σ[] : {Δ Γ : Con}(A : Ty Γ)(B : Ty (Γ ▹ A))(σ : Sub Δ Γ) ⟶
  Σ(A,B)[ σ ] = Σ {Γ=Δ} (A [ σ ]) (B [ q(σ;A) ])
pair : {Γ}{A}{B} ⟶ Sub (Γ ▹ A ▹ B) (Γ ▹ Σ(A,B))
pair[] : {Δ Γ : Con}(A : Ty Γ)(B : Ty (Γ ▹ A))(σ : Sub Δ Γ) ⟶
  q(σ;Σ(A,B)) ∘ pair = pair ∘ q(σ;A,B)
p∘pair : {Γ}{A}{B} ⟶ (p (Σ(A,B))) ∘ pair = (p A) ∘ (p B)
Σ-elim : {Γ}{A}{B}{M : Ty (Γ ▹ Σ(A,B))} ⟶
  Tm(Γ ▹ A ▹ B, M[pair]) ⟶ Tm(Γ ▹ Σ(A,B), M)
Σβ : {Γ}{A}{B}{M}(m : Tm(Γ ▹ A ▹ B, M[pair])) ⟶
  (Σ-elim m)[ pair ] = m
Ση : {Γ}{A}{B}{M}(m : Tm(Γ ▹ A ▹ B, M[pair])) ⟶
  (z : Tm(Γ ▹ A ▹ B, M[pair])) ⟶ (z[ pair ] = m) ⟶ z = Σ-elim m
Σ-elim[] : {Δ}{Γ}{A}{B}{M} ⟶
  (m : Tm(Γ ▹ A ▹ B, M[pair])) ⟶ (σ : Sub Δ Γ) ⟶
  (Σ-elim m)[ q(σ;Σ(A,B)) ] = Σ-elim {Γ=Δ} (m [ q(σ;A,B) ])
```

There's a bit of daylight between our intuitive idea of how to construct a term of

LEAN–NOUGAT

```
def 𝔈𝔴𝔉Π : GAT := {[
  include 𝔈𝔴𝔉;
    Pi : {Γ : Con} ⇒
        (A : Ty Γ) ⇒ Ty (ext Γ A) ⇒ Ty Γ,
    Pi_stab : {Δ Γ : Con} ⇒ (σ : Sub Δ Γ) ⇒
        (A : Ty Γ) ⇒ (B : Ty (ext Γ A)) ⇒
        substTy Δ Γ σ (Pi A B)
        ≡ Pi (substTy σ A)
             (substTy (pair
                    (comp σ (p (substTy σ A)))
                    (v (substTy σ A)))
             B),
    lam : {Γ : Con} ⇒
        {A : Ty Γ} ⇒ {B : Ty (ext Γ A)} ⇒
        Tm (ext Γ A) B ⇒ Tm Γ (Pi A B),
    app : {Γ : Con} ⇒
        {A : Ty Γ} ⇒ {B : Ty (ext Γ A)} ⇒
        Tm Γ (Pi A B) ⇒ Tm (ext Γ A) B,
    lam_stab : {Δ Γ : Con} ⇒ (σ : Sub Δ Γ) ⇒
        {A : Ty Γ} ⇒ {B : Ty (ext Γ A)} ⇒
        (t : Tm (ext Γ A) B) ⇒
        substTm σ (Pi A B) (lam t)
            #⟨Pi_stab σ A B⟩
        ≡  lam (substTm (pair
                    (comp σ (p (substTy σ A)))
                    (v (substTy σ A)))
                t),
    Pi_β : {Γ : Con} ⇒
        {A : Ty Γ} ⇒ {B : Ty (ext Γ A)} ⇒
        (t : Tm (ext Γ A) B) ⇒ app (lam t) ≡ t,
    Pi_η : {Γ : Con} ⇒
        {A : Ty Γ} ⇒ {B : Ty (ext Γ A)} ⇒
        (f : Tm Γ (Pi A B)) ⇒ lam (app f) ≡ f
]}
```

Figure 1.6: $\mathfrak{CwF}_{\Pi}$, the GAT of CwFs supporting Π-types.

Lean—nouGAT

```
def 𝔈𝔴𝔉_Σ : GAT := {[
  include 𝔈𝔴𝔉 renaming (pair to pair_Sub);
    Sigma : {Γ : Con} ⇒
        (A : Ty Γ) ⇒ Ty (ext Γ A) ⇒ Ty Γ,
    Sigma_stab : {Δ Γ : Con} ⇒ (σ : Sub Δ Γ) ⇒
        (A : Ty Γ) ⇒ (B : Ty (ext Γ A)) ⇒
        substTy Δ Γ σ (Sigma A B)
        ≡ Pi (substTy σ A)
             (substTy (pair_Sub
                    (comp σ (p (substTy σ A)))
                    (v (substTy σ A)))
              B),
    pair : {Γ : Con} ⇒
        {A : Ty Γ} ⇒ {B : Ty (ext Γ A)} ⇒
        Sub (ext (ext Γ A) B) (ext Γ (Sigma A B)),
    pair_comp : {Δ Γ : Con} ⇒
        (A : Ty Γ) ⇒ (B : Ty (ext Γ A)) ⇒
        comp
            (pair_Sub (comp σ (p (substTy σ A))) (v
   (substTy σ A)))
            pair
        ≡
        comp
            pair
            (pair_Sub
                (comp (pair_Sub (comp σ (p (substTy σ
   A))) (v (substTy σ A))) (p (substTy (pair_Sub (comp
   σ (p (substTy σ A))) (v (substTy σ A))) B)))
                (v (substTy (pair_Sub (comp σ (p
   (substTy σ A))) (v (substTy σ A))) B))
            )
]}
```

Figure 1.7: $\mathfrak{CwF}_\Sigma$, the GAT of CwFs supporting Σ-types.

type $\sum_{x:A} B(x)$—pair together a term $a : A$ and a term $b : B(a)$—with the specification of Σ-types given above. It'll be helpful to develop this point a bit more, as it will serve as useful contrast later on when we characterize Σ-types in *directed type theory*. We therefore have the following results.

Proposition 1.2.14. *In any CwF with Σ-types, there is a bijection*

$$\mathsf{Pair} \;:\; (s' : \mathsf{Tm}(\Gamma, A)) \times \mathsf{Tm}(\Gamma, B[\mathsf{id}, s']) \;\cong\; \mathsf{Tm}(\Gamma, \Sigma\ A\ B) : \;\mathsf{Unpair}.$$

Proof. We realize this by constructing a substitution $\mathsf{unpair} : \mathsf{Sub}\ (\Gamma \rhd \Sigma\ A\ B)\ (\Gamma \rhd A \rhd B)$ inverse to pair

$$\mathsf{unpair} := (\mathsf{p}_{\Sigma\ A\ B},\ \Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1,\ \Sigma\text{-}\mathsf{elim}\ \mathsf{v}_0)$$

This is an inverse to pair: for the one direction, we have

$$\begin{aligned}
\mathsf{unpair} \circ \mathsf{pair} &= (\mathsf{p}_{\Sigma\ A\ B} \circ \mathsf{pair},\ (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{pair}],\ (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_0)[\mathsf{pair}]) \\
&= (\mathsf{p}_A \circ \mathsf{p}_B,\ (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{pair}],\ (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_0)[\mathsf{pair}]) && (\mathsf{p}\circ\mathsf{pair}) \\
&= (\mathsf{p}_A \circ \mathsf{p}_B,\ \mathsf{v}_1,\ \mathsf{v}_0) && (\Sigma\beta) \\
&= \mathsf{id}_{\Gamma \rhd A \rhd B}.
\end{aligned}$$

For the other direction, we prove that $\Sigma\text{-}\mathsf{elim}\ m = m[\mathsf{unpair}]$ for arbitrary m. By $\Sigma\eta$, it suffices to show that $m[\mathsf{unpair}][\mathsf{pair}] = m$, which follows from the proof above $\mathsf{unpair} \circ \mathsf{pair} = \mathsf{id}$. So then

$$\begin{aligned}
\mathsf{id}_{\Sigma\ A\ B} &= (\mathsf{p}_{\Sigma\ A\ B},\ \mathsf{v}_0) \\
&= (\mathsf{p}_A \circ \mathsf{p}_B \circ \mathsf{unpair},\ \mathsf{v}_0) \\
&= (\mathsf{p}_A \circ \mathsf{p}_B \circ \mathsf{unpair},\ \Sigma\text{-}\mathsf{elim}(\mathsf{v}_0[\mathsf{pair}])) && (\Sigma\eta) \\
&= (\mathsf{p}_A \circ \mathsf{p}_B \circ \mathsf{unpair},\ \mathsf{v}_0[\mathsf{pair}][\mathsf{unpair}]) && (\text{above}) \\
&= (\mathsf{p}_A \circ \mathsf{p}_B,\ \mathsf{v}_0[\mathsf{pair}]) \circ \mathsf{unpair} \\
&= (\mathsf{p}_{\Sigma\ A\ B} \circ \mathsf{pair},\ \mathsf{v}_0[\mathsf{pair}]) \circ \mathsf{unpair} && (\mathsf{p}\circ\mathsf{pair}) \\
&= (\mathsf{p}_{\Sigma\ A\ B},\ \mathsf{v}_0) \circ \mathsf{pair} \circ \mathsf{unpair} && (\mathsf{p}\circ\mathsf{pair}) \\
&= \mathsf{id} \circ \mathsf{pair} \circ \mathsf{unpair}.
\end{aligned}$$

Now we're ready to define the "external" versions, Pair and Unpair.

$$\begin{aligned}
\mathsf{Pair}\ (s', t') &:= \mathsf{v}_0[\mathsf{pair} \circ (\mathsf{id}, s', t')] \\
\mathsf{Unpair}\ T' &:= ((\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{id}, T'], \Sigma\text{-}\mathsf{elim}\ \mathsf{v}_0)[\mathsf{id}, T'])
\end{aligned}$$

These are mutually inverse, as claimed:

$$\begin{aligned}
(\mathsf{Unpair}(\mathsf{Pair}(s',t')))_{\#1} &= (\mathsf{Unpair}(\mathsf{v}_0[\mathsf{Pair}\circ(\mathsf{id},s',t')]))_{\#1}\\
&= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{id},\mathsf{v}_0[\mathsf{Pair}\circ(\mathsf{id},s',t')]]\\
&= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{p}_A\circ\mathsf{p}_B\circ(\mathsf{id},s',t'),\mathsf{v}_0[\mathsf{Pair}\circ(\mathsf{id},s',t')]]\\
&= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{p}_{\Sigma\ A\ B}\circ\mathsf{pair}\circ(\mathsf{id},s',t'),\mathsf{v}_0[\mathsf{Pair}\circ(\mathsf{id},s',t')]]\\
&= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{p}_{\Sigma\ A\ B},\mathsf{v}_0][\mathsf{Pair}][\mathsf{id},s',t']\\
&= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{Pair}][\mathsf{id},s',t']\\
&= \mathsf{v}_1[\mathsf{id},s',t']\\
&= s'\\
(\mathsf{Unpair}(\mathsf{Pair}(s',t')))_{\#2} &= (\mathsf{Unpair}(\mathsf{v}_0[\mathsf{Pair}\circ(\mathsf{id},s',t')]))_{\#2}\\
&= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_0)[\mathsf{id},\mathsf{v}_0[\mathsf{Pair}\circ(\mathsf{id},s',t')]]\\
&= \mathsf{v}_0[\mathsf{id},s',t']\\
&= t'\\
\mathsf{Pair}(\mathsf{Unpair}\ T') &= \mathsf{v}_0[\mathsf{pair}][\mathsf{id},(\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{id},T'],(\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_0)[\mathsf{id},T']]\\
&= \mathsf{v}_0[\mathsf{pair}][\mathsf{p}_{\Sigma\ A\ B},\Sigma\text{-}\mathsf{elim}(\mathsf{v}_1),\Sigma\text{-}\mathsf{elim}(\mathsf{v}_0)][\mathsf{id},T']\\
&= \mathsf{v}_0[\mathsf{pair}][\mathsf{unpair}][\mathsf{id},T']\\
&= \mathsf{v}_0[\mathsf{id},T'] && \text{(above)}\\
&= T'.
\end{aligned}$$

□

Definition 1.2.15. Given $S\colon \mathsf{Tm}(\Gamma,\Sigma AB)$, write

$$\begin{aligned}
&\mathsf{pr}_1(S)\colon \mathsf{Tm}(\Gamma,A)\\
&\mathsf{pr}_1(S) := (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{id},S]\\
&\mathsf{pr}_2(S)\colon \mathsf{Tm}(\Gamma,B[\mathsf{id},\mathsf{pr}_1(S)])\\
&\mathsf{pr}_2(S) := (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_0)[\mathsf{id},S]
\end{aligned}$$

i.e. $\mathsf{pr}_1(S) := (\mathsf{Unpair}\ S)_{\#1}$ and $\mathsf{pr}_2(S) := (\mathsf{Unpair}\ S)_{\#2}$.

Proposition 1.2.16. *The projections are stable under substitution: for every* $S\colon \mathsf{Tm}(\Gamma,\Sigma AB)$ *and* $\sigma\colon \mathsf{Sub}\ \Delta\ \Gamma$

$$(\mathsf{pr}_1\ S)[\sigma] = \mathsf{pr}_1(S[\sigma]) \qquad (\mathsf{pr}_2\ S)[\sigma] = \mathsf{pr}_2(S[\sigma]) \tag{1.2.17}$$

Proof. Observe that the following square commutes.

$$\begin{array}{ccc} \Delta \rhd (\Sigma AB)[\sigma] & \xrightarrow{\mathsf{q}(\sigma;\Sigma AB)} & \Gamma \rhd \Sigma AB \\ {\scriptstyle \mathsf{id},S[\sigma]}\uparrow & & \uparrow{\scriptstyle \mathsf{id},S} \\ \Delta & \xrightarrow[\sigma]{} & \Gamma \end{array}$$

So then:

$$\begin{aligned} (\mathsf{pr}_1\ S)[\sigma] :&= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{id}, S][\sigma] \\ &= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{q}(\sigma; \Sigma AB)][\mathsf{id}, S[\sigma]] && \text{(above)} \\ &= (\Sigma\text{-}\mathsf{elim}(\mathsf{v}_1[\mathsf{q}(\sigma; A, B)]))[\mathsf{id}, S[\sigma]] && (\Sigma\text{-}\mathsf{elim}[]) \\ &= (\Sigma\text{-}\mathsf{elim}\ \mathsf{v}_1)[\mathsf{id}, S[\sigma]] \\ &=: \mathsf{pr}_1(S[\sigma]) \end{aligned}$$

where the penultimate identity follows from the fact that $\mathsf{q}(\sigma; A, B) := (\sigma \circ \mathsf{p} \circ \mathsf{p}, \mathsf{v}_1, \mathsf{v}_0)$. The proof for pr_2 is identical. □

Finally, there will be another kind of type-former playing a central role in our story: *identity types*. As indicated above, it is the modification of identity types into *hom-types* which chiefly distinguishes directed type theory. In *un*directed type theory, that is, standard Martin-Löf Type Theory, identity types come in two possible varieties: **intensional** and **extensional**. Both seek to make the metatheoretic *judgmental equality* expressible within the object language, but do so to different extents: extensional equality reflects judgmental equality exactly, so extensionally equal terms are indeed equal judgmentally. Intensional equality, on the other hand, provides a coarser notion of equality: there can be terms t, t' which are propositionally equal (i.e. the intensional identity type $\mathsf{Id}(t, t')$ is inhabited) but which are not judgmentally equal. In the present work, we'll mostly be interested in intensional identity types, but we introduce both.

Definition 1.2.18. A CwF supports **intensional identity types** if it is equipped with the following data.

pseudoAgda

```
Id : {Γ : Con}{A : Ty Γ} ⟶ (t t' : Tm(Γ, A)) ⟶ Ty Γ
Id[] : {Δ Γ : Con}{A : Ty Γ}{t}{t'} ⟶ (σ : Sub Δ Γ) ⟶
  Id(t,t')[ σ ] = Id (t[ σ ], t'[ σ ])
refl : {Γ}{A} ⟶ (t : Tm(Γ, A)) ⟶ Tm(Γ, Id(t,t))
refl[] : {Δ Γ : Con}{A : Ty Γ}{t} ⟶ (σ : Sub Δ Γ) ⟶
  refl_t[ σ ] = refl_t[σ]
J : {Γ}{A} ⟶ (t : Tm(Γ, A))
  ⟶ (M : Ty (Γ ▹ A ▹ Id(t[ p ], v₀)))
  ⟶ Tm(Γ, M[ id , t , refl_t ])
  ⟶ Tm(Γ ▹ A ▹ Id(t[ p ], v₀), M)
J[] : {Γ}{A}{t}{M}{m : Tm(Γ, M[ id , t , refl_t ])} ⟶
  (σ : Sub Δ Γ) ⟶ J(m[ σ ]) = (J m)[ q(σ; A, Id(t[p],v₀)) ]
Jβ : {Γ}{A}{t}{M} ⟶ (m : Tm(Γ, M[ id , t , refl_t ])) ⟶
  (J_{t,M} m)[ id , t , refl_t ] = m
```

Definition 1.2.19. A CwF supports **extensional identity types** if it is equipped with the following data.

pseudoAgda

```
Eq : {Γ : Con}{A : Ty Γ} ⟶ (t t' : Tm(Γ, A)) ⟶ Ty Γ
Eq[] : {Δ Γ : Con}{A : Ty Γ}{t}{t'} ⟶ (σ : Sub Δ Γ) ⟶
  Eq(t,t')[ σ ] = Eq(t[ σ ], t'[ σ ])
refl : {Γ}{A} ⟶ (t : Tm(Γ, A)) ⟶ Tm(Γ, Eq(t,t))
reflect : {Γ}{A}{t}{t'} ⟶ Tm(Γ, Eq(t,t')) ⟶ t = t'
canon : {Γ}{A}{t}{t'} ⟶ (h : Tm(Γ, Eq(t,t'))) ⟶ h = refl_t
```

1.3 A Signature Language for GATs

> "We cannot undertake to define *being* without falling into this absurdity: because we cannot define a word without starting with "*it is*" either expressed or implied. So to define *being*, we would have to say "it is", and thus use the word defined in its definition."
>
> Blaise Pascal, *Pensées et Opuscules*

For us, a "type theory" means "the initial algebra of (some extension of) the GAT of CwFs." As we saw in the previous section, we're able to articulate various systems of type- and term-formers as GAT extensions of $\mathfrak{CwF}$; moreover, since contexts, substitutions, and context extension are also explicitly axiomatized in $\mathfrak{CwF}$, we can also articulate operations on these data, such as occurs in modal and substructural type theories. Every GAT has an initial algebra, so we can rest assured that, whatever strange

system we devise in this manner, it makes sense to speak of its *syntax model*, i.e. think of it as a type theory.

But, so far, we've only loosely, informally indicated (at the end of section 1.1) what constitutes this initial algebra. Our goal in the present section is to be more precise. We'll do so by way of a **universal method**: we'll identify a single GAT $\mathfrak{U}$ with the remarkable property that, if $\mathfrak{U}$ has an initial algebra, then *every* GAT has an initial algebra. How can we do this? Well, we'll choose this $\mathfrak{U}$ to be an extension of $\mathfrak{CwF}$, so its initial algebra is a type theory. Specifically, the initial $\mathfrak{U}$-algebra will be a syntax *for writing down GATs*, a **signature language for GATs**. This type theory is what we'll call ONEGAT. We'll be able to write down every GAT in the ONEGAT syntax (recall that the NOUGAT syntax used above to introduce GATs is ultimately just syntactic sugar for ONEGAT). With this, we'll be able to make definitions (such as the notions of $\mathfrak{G}$-algebra and $\mathfrak{G}$-homomorphism) for arbitrary GATs $\mathfrak{G}$ by *induction*—the GAT $\mathfrak{G}$ will be a piece of ONEGAT syntax, built up recursively using the components of the ONEGAT language. Finally, we'll be able to make precise our intuitive description of initial algebras from above: recall, for instance, that we said the set of natural numbers would be defined as the "terms-in-context" of the form

$$\{\!|\texttt{Nat : U, zero : Nat, succ : Nat} \Rightarrow \texttt{Nat}|\!\} \vdash \texttt{t} : \texttt{Nat}.$$

A GAT will be a *context* in the ONEGAT language. In particular, the GAT of natural number algebras $\mathfrak{N}$ will be a context, which is what appears on the left of the turnstile above. In a type-theoretic language like ONEGAT, we will be able to speak of types (like `Nat`) obtained from the context, and terms of those types, giving formal meaning to the sequent above.

1.3.1 Introducing GATs

So let's introduce ONEGAT. We won't actually articulate ONEGAT as the initial algebra of a GAT—this would be a bit circular, as GATs are themselves written in ONEGAT! Rather, we'll assert it as a metatheoretic **quotient inductive-inductive type**; our assumption that the "universal GAT has an initial algebra" is the assumption that we actually do get syntax as declared in Figure 1.8, and can perform induction on this syntax as in Remark 1.3.12.

Definition 1.3.1. The language ONEGAT consists of the components

$$\begin{aligned}
\overline{\mathsf{Con}} &: \mathsf{Set} \\
\overline{\mathsf{Sub}} &: \overline{\mathsf{Con}} \to \overline{\mathsf{Con}} \to \mathsf{Set} \\
\overline{\mathsf{Ty}} &: \overline{\mathsf{Con}} \to \mathsf{Set} \\
\overline{\mathsf{Tm}} &: (\mathfrak{G} : \overline{\mathsf{Con}}) \to \overline{\mathsf{Ty}}\ \mathfrak{G} \to \mathsf{Set}
\end{aligned}$$

as given by the metatheoretic quotient inductive-inductive type in Figure 1.8.

Recall the different kinds of components we could add to a GAT: sorts, elements, functions (elements and sorts depending on element(s) of other sorts), and equations.

pseudoAgda

```
data Con : Set where
  ◇ : Con
  _▹_ : (𝔊 : Con) → Ty 𝔊 → Con

data Sub : Con → Con → Set where
  id : Sub 𝔊 𝔊
  _∘_ : Sub 𝔊 ℌ → Sub 𝔉 𝔊 → Sub 𝔉 ℌ
  ϵ : Sub 𝔊 ◇
  _,_ : (𝔤 : Sub ℌ 𝔊) → Tm ℌ (𝒜 [ 𝔤 ]) → Sub ℌ (𝔊 ▹ 𝒜)
  π₁ : Sub ℌ (𝔊 ▹ 𝒜) → Sub ℌ 𝔊
  idl : id ∘ 𝔤 = 𝔤
  idr : 𝔤 ∘ id = 𝔤
  assoc : (𝔥 ∘ 𝔤) ∘ 𝔣 = 𝔥 ∘ (𝔤 ∘ 𝔣)
  ◇η : (𝔰 : Sub 𝔊 ◇) → 𝔰 = ϵ
  ▹β₁ : π₁ ( 𝔤 , t ) = 𝔤
  ▹η : (t : Sub ℌ (𝔊 ▹ 𝒜)) → ( π₁ t , π₂ t ) = t
  ,∘ : ( 𝔤 , t ) ∘ 𝔣 = ( 𝔤 ∘ 𝔣 , t [ 𝔣 ] )

data Ty : Con → Set where
  _[_] : Ty 𝔊 → Sub ℌ 𝔊 → Ty ℌ
  U : Ty 𝔊
  El : Tm 𝔊 U → Ty 𝔊
  Π : (X : Tm 𝔊 U) → Ty (𝔊 ▹ El X) → Ty 𝔊
  Eq : (X : Tm 𝔊 U) → Tm 𝔊 (El X) → Tm 𝔊 (El X) → Ty 𝔊
  [id] : 𝒜 [ id ] = 𝒜
  [∘] : (𝒜 [ 𝔤 ]) [ 𝔥 ] = 𝒜 [ 𝔤 ∘ 𝔥 ]
  U[] : U [ 𝔤 ] = U
  El[] : (El X) [ 𝔤 ] = El (X [ 𝔤 ])
  Π[] : (Π X 𝒴)[ 𝔤 ] = Π (X [ 𝔤 ]) (𝒴 [ 𝔤 ∘ π₁(id) , π₂ id ])
  Eq[] : (Eq X t t')[ 𝔤 ] = Eq (X [ 𝔤 ]) (t [ 𝔤 ]) (t' [ 𝔤 ])

data Tm : (𝔊 : Con) → Ty 𝔊 → Set where
  _[_] : Tm 𝔊 𝒜 → (𝔤 : Sub ℌ 𝔊) → Tm ℌ (𝒜 [ 𝔤 ])
  π₂ : (t : Sub ℌ (𝔊 ▹ 𝒜)) → Tm ℌ (𝒜[ π₁ t ])
  app : Tm 𝔊 (Π X 𝒴) → Tm (𝔊 ▹ El X) 𝒴

  [id] : t [ id ] = t
  [∘] : (t [ 𝔤 ]) [ 𝔥 ] = t [ 𝔤 ∘ 𝔥 ]
  ▹β₂ : π₂ ( 𝔤 , t ) = t
  app[] : (app f)[ 𝔤 ∘ π₁(id) , π₂ id ] = app(f[ 𝔤 ])
  reflect : Tm 𝔊 (Eq X t t') → t = t'
```

Figure 1.8: QIIT definition of the onegat language. Taken (with slight modification) from [KKA19].

These are precisely what kinds of things we're able to write in ONEGAT. That is, ONEGAT consists of a type theoretic language with a universe U, a specially-restricted kind of Π-types, and extensional equality types. Let's examine these one-by-one, and see how the GATs introduced in section 1.1 can be expressed as the contexts of ONEGAT, i.e. as elements of the set $\overline{\mathsf{Con}}$.

The most basic GAT is the *empty GAT*, which has no components and whose algebras consist of no data (not even a carrier set). This is, of course, represented by the empty ONEGAT context, $\diamond$. As noted above, for any other GAT we must begin by asserting a sort. This is done by extending $\diamond$ by the universe U:

Example 1.3.2 (Sets—ONEGAT)**.** The GAT of **sets** (Example 1.1.1) is represented in ONEGAT as the following context.

ONEGAT

$\diamond$
$\triangleright\ \mathsf{U}$

If we want to *use* this sort (e.g. to declare elements of it, or have subsequent components depend on it), then we need a way to refer to it within the ONEGAT language. As before, we use *de Bruijn indices* to abstract away from the difficulties of named variables. We use slightly-different notations than we did in Definition 1.2.4, to (hopefully) be able to keep things straight in cases where the GAT we're articulating has its *own* internal notion of de Bruijn indices (i.e. the GAT $\mathfrak{CwF}$ and extensions of it).

Definition 1.3.3. Write wk for the substitution

$$\pi_1(\mathsf{id}_{\mathfrak{G}\triangleright\mathcal{A}}) \quad : \overline{\mathsf{Sub}}\ (\mathfrak{G}\triangleright\mathcal{A})\ \mathfrak{G}.$$

We'll make use of *de Bruijn indices* in the ONEGAT language:

$$\begin{aligned}
0 &:= \pi_2(\mathsf{id}_{\mathfrak{G}\triangleright\mathcal{A}_0}) && : \overline{\mathsf{Tm}}(\mathfrak{G}\triangleright\mathcal{A}_0, \mathcal{A}_0[\,\mathsf{wk}\,])\\
1 &:= 0[\,\mathsf{wk}\,] && : \overline{\mathsf{Tm}}(\mathfrak{G}\triangleright\mathcal{A}_1\triangleright\mathcal{A}_0, \mathcal{A}_1[\,\mathsf{wk}\circ\mathsf{wk}\,])\\
2 &:= 1[\,\mathsf{wk}\,] && : \overline{\mathsf{Tm}}(\mathfrak{G}\triangleright\mathcal{A}_2\triangleright\mathcal{A}_1\triangleright\mathcal{A}_0, \mathcal{A}_2[\,\mathsf{wk}\circ\mathsf{wk}\circ\mathsf{wk}\,])\\
&\ \vdots
\end{aligned}$$

Given a dependent function term $\mathrm{f}: \overline{\mathsf{Tm}}(\mathfrak{G}, \Pi\,\mathrm{X}\,\mathcal{Y})$ and an argument term $\mathrm{t}: \overline{\mathsf{Tm}}(\mathfrak{G}, \mathcal{A})$, it'll be convenient to use the abbreviation

$$\mathrm{f}\ @\ \mathrm{t} := (\mathsf{app}\ \mathrm{f})[\,\mathsf{id}, \mathrm{t}\,] \quad : \overline{\mathsf{Tm}}(\mathfrak{G}, \mathcal{Y}[\,\mathsf{id}, \mathrm{t}\,]).$$

The $\_\ @\ \_$ operator will be *left-associative*, matching the *right-associativity* of $\Rightarrow$.

Given such a term $\mathrm{X}: \overline{\mathsf{Tm}}\ \mathfrak{G}\ \mathsf{U}$, we can obtain $\mathsf{El}\ \mathrm{X}: \overline{\mathsf{Ty}}\ \mathfrak{G}$. This is how we'll add elements of a previously-declared sort to our GATs.

Example 1.3.4 (Pointed Sets—onegat)**.** The GAT $\mathfrak{P}$ of **pointed sets** (Example 1.1.2) is represented in onegat as the following context.

onegat

◇
▷ U
▷ El 0

Example 1.3.5 (Bipointed Sets—onegat)**.** The GAT $\mathfrak{B}$ of **bipointed sets** (Example 1.1.3) is represented in onegat as the following context.

onegat

◇
▷ U
▷ El 0
▷ El 1

Now, recall the restrictions on the use of $\Rightarrow$ discussed in section 1.1: its domain had to be *small*, i.e. a previously-declared sort. This prevented us from writing higher-order functions, choice functions, etc. This restriction is encoded in onegat by the shape of the Π type-former:

$$\Pi : (X : \overline{\mathsf{Tm}}\, \mathfrak{G}\, \mathsf{U}) \to \overline{\mathsf{Ty}}(\mathfrak{G} \triangleright \mathsf{El}\, X) \to \overline{\mathsf{Ty}}\, \mathfrak{G}.$$

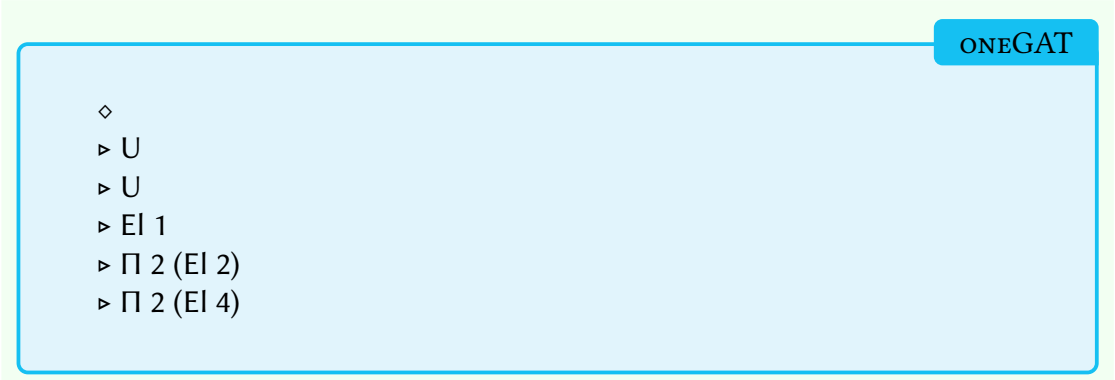
Contrast this with the Π-formation given in Definition 1.2.12: the domain is not a *type* (an element of $\overline{\mathsf{Ty}}\ \mathfrak{G}$), but rather a term of the universe U. The *co*domain is allowed to be "large", including U itself.

Example 1.3.6 (Natural Number Algebras—onegat)**.** The GAT $\mathfrak{N}$ of **natural number algebras** (Example 1.1.4) is represented in onegat as the following context.

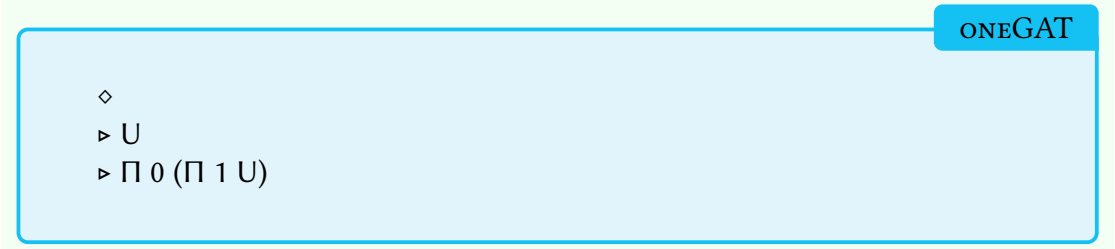
onegat

◇
▷ U
▷ El 0
▷ Π 1 (El 2)

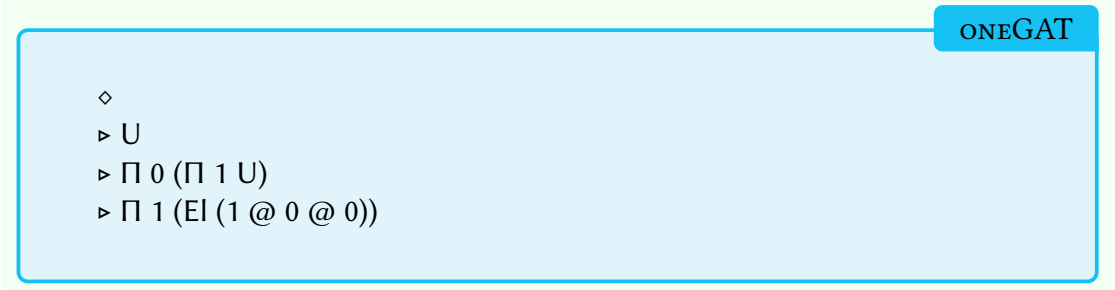
Example 1.3.7 (Even-Odd Algebras—onegat)**.** The GAT $\mathfrak{EO}$ of **even-odd algebras** (Example 1.1.5) is represented in onegat as the following context.



Example 1.3.8 (Quivers—oneGAT)**.** The GAT $\mathfrak{Quiv}$ of **quivers** (Example 1.1.6) is represented in oneGAT as the following context.



Example 1.3.9 (Reflexive Quivers—oneGAT)**.** The GAT $\mathfrak{rQuiv}$ of **reflexive quivers** (Example 1.1.7) is represented in oneGAT as the following context.



Recall that we defined $\mathfrak{rQuiv}$ in nouGAT by `include`-ing the GAT $\mathfrak{Quiv}$. Observe, then, that $\mathfrak{Quiv}$ is a prefix of $\mathfrak{rQuiv}$:

$$\mathfrak{rQuiv} \quad = \quad \mathfrak{Quiv} \triangleright \Pi\ 1\ (\mathsf{El}\ (1 @ 0 @ 0)).$$

This is precisely what the `include` keyword means.

Finally, oneGAT comes equipped with extensional identity types, so we can add equations to our GAT signatures.

Example 1.3.10 (Groups—oneGAT)**.** The GAT $\mathfrak{Grp}$ of **groups** (Example 1.1.9) is represented in oneGAT as the following context.

oneGAT

◇
▷ U
▷ El 0
▷ Π 1 (Π 2 (El 3))
▷ Π 2 (Eq (1 @ 2 @ 0) 0)
▷ Π 3 (Eq (2 @ 0 @ 3) 0)
▷ Π 4 (Π 5 (Π 6 (Eq (5 @ 2 @ (5 @ 1 @ 0)) (5 @ (5 @ 2 @ 1) @ 0))))
▷ Π 5 (El 6)
▷ Π 6 (Eq (5 @ (1 @ 0) @ 0) 6)
▷ Π 7 (Eq (6 @ 0 @ (2 @ 0)) 7)

All the equations in $\mathfrak{Grp}$ are between elements of the one sort (the carrier set of the group), i.e. are *homogeneous* equalities. But we also want to express *heterogeneous* equalities, that is, equations which are only well-formed by virtue of a previous equation. For instance, in the definition of 'CwF', we assert the equation

$$t[\mathsf{id}] = t$$

for all $t : \mathsf{Tm}(\Gamma, A)$. But the left-hand side of this is an element of the set $\mathsf{Tm}(\Gamma, A[\mathsf{id}])$ and the right-hand side is an element of $\mathsf{Tm}(\Gamma, A)$. It is only due to the equation $A[\mathsf{id}] = A$ that we can make this equation: we **transport** $t[\mathsf{id}]$ along the equation $A[\mathsf{id}] = A$ to get an element of $\mathsf{Tm}(\Gamma, A)$, and *this* we equate to t. Recall that we treat such transports in the metatheory as implicit, but in nouGAT we marked them explicitly. We will do likewise in oneGAT: the Eq type-former insists that the terms being equated are literally of the same type, so to define $\mathfrak{CwF}$ we must make use of the following transport operator.

Definition 1.3.11. [KKA19, p. 12] oneGAT has an operation

$$\frac{\begin{array}{cc} \mathrm{X} : \overline{\mathsf{Tm}}\ \mathfrak{G}\ \mathsf{U} & \mathcal{Y} : \overline{\mathsf{Ty}}\ (\mathfrak{G} \triangleright \mathsf{El}\ \mathrm{X}) \\ \mathrm{t}\ \mathrm{t}' : \overline{\mathsf{Tm}}\ \mathfrak{G}\ (\mathsf{El}\ \mathrm{X}) & \mathrm{eq} : \overline{\mathsf{Tm}}\ \mathfrak{G}\ \mathsf{Eq}\ \mathrm{t}\ \mathrm{t}' \quad \mathrm{y} : \overline{\mathsf{Tm}}\ \mathfrak{G}\ (\mathcal{Y}[\mathsf{id}, \mathrm{t}]) \end{array}}{\mathsf{transp}\ \mathrm{eq}\ \mathrm{y} : \overline{\mathsf{Tm}}\ \mathfrak{G}\ \mathcal{Y}[\mathsf{id}, \mathrm{t}'].}$$

To obtain transp eq y, use reflect to turn eq into a metatheoretic equality $\mathrm{t} = \mathrm{t}'$. Then transport y along this equality.

This transp operation is put to use in $\mathfrak{CwF}$: see Figure 1.9. There the equation between $t[\mathsf{id}]$ and t is the last line; the innermost context (where we're formulating the equation) is

$$\mathfrak{Cat} \triangleright \mathsf{El}\ 6 \triangleright \cdots \triangleright \mathsf{El}\ 15 \triangleright \mathsf{El}(6 \mathbin{@} 0) \triangleright \mathsf{El}(3 \mathbin{@} 1 \mathbin{@} 0)$$

So, in this context, the de Bruijn index 2 is some $\Gamma : \mathsf{Con}$ (the sort of contexts of the CwF—be careful to distinguish this from $\overline{\mathsf{Con}}$); the index 1 is some type $A : \mathsf{Ty}\ \Gamma$; and

onеGAT

```
𝔈at
▹ El 6
▹ Π 7 (El (7 @ 0 @ 1))
▹ Π 8 (Π (8 @ 0 @ 2) (Eq 0 (2 @ 1)))
▹ Π 9 U
▹ Π 10 (Π 11 (Π (11 @ 1 @ 0) (Π (3 @ 1) (El (4 @ 3)))))
▹ Π 11 (Π (2 @ 0) (Eq (2 @ 1 @ 1 @ (11 @ 1) @ 0) 0))
▹ Π 12 (Π 13 (Π 14 (Π (5 @ 0) (Π (15 @ 3 @ 2) (Π (16 @ 3 @ 2) (Eq (7 @
    4 @ 3 @ 0 @ (7 @ 5 @ 4 @ 1 @ 2)) (7 @ 5 @ 3 @ (15 @ 5 @ 4 @ 3
    @ 0 @ 1) @ 2)))))))
▹ Π 13 (Π (4 @ 0) U)
▹ Π 14 (Π 15 (Π (6 @ 0) (Π (16 @ 2 @ 1) (Π (4 @ 2 @ 1) (El (5 @ 4 @ (8
    @ 4 @ 3 @ 1 @ 2)))))))
▹ Π 15 (Π (6 @ 0) (Π (3 @ 1 @ 0) (Eq (transp (6 @ 2 @ 1) (3 @ 2 @ 2 @
    1 @ (16 @ 2) @ 0)) 0)))
```

Figure 1.9: The GAT of CwFs, up through the heterogeneous equation between $t[\mathrm{id}]$ and t. See Appendix A for the whole GAT.

index 0 is a term $t : \mathsf{Tm}(\Gamma, A)$. The de Bruijn index 6 is the identity law for Ty: that $A[\mathrm{id}]$ is A for any type A. So we instantiate this law for our context and type, $6 \mathbin{@} 2 \mathbin{@} 1$. Then $t[\mathrm{id}]$ is formally written as

$$3 \mathbin{@} 2 \mathbin{@} 2 \mathbin{@} 1 \mathbin{@} (16 \mathbin{@} 2) \mathbin{@} 0.$$

3 is the term-substitution operation; it requires us to explicitly supply the domain and codomain of the substitution (both of which are Γ, i.e. 2) and the type (A, i.e. 1). Then $16 \mathbin{@} 2$ is id_Γ, and 0 is t. So, we transport this along $6 \mathbin{@} 2 \mathbin{@} 1$, and then equate it to t, i.e. 0. This is the component appearing in Figure 1.9.

The onеGAT presentation of all the GATs in section 1.1 (and all the GATs in this thesis) are given in Appendix A. As demonstrated by the previous paragraph, working explicitly with onеGAT quickly gets very tedious and confusing, and it becomes very difficult to recognize what structure is being encoded. It is intentionally a very low-level representation, a "machine code" for GATs. It is for this reason that we introduced nouGAT: to serve as a readable interface for onеGAT. The complex onеGAT samples in this thesis (like Figure 1.9) were *not* produced by hand: they were produced by the nouGAT-to-onеGAT parser implemented in Lean. But the reason we have onеGAT is for its inductive structure: every GAT is of the form $\diamond \triangleright \mathcal{A}_0 \triangleright \mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n$, where the $\mathcal{A}_i$'s are formed by U, El, Π, Eq, etc. This means we can make definitions on onеGAT *by induction*; we now turn our attention to this process.

1.3.2 Eliminating GATs

A crucial property of a formal language is that every piece of syntax is built up from the constructors of the language in a unique way. In particular for oneGAT:

- every $\mathfrak{G} : \overline{\mathsf{Con}}$ is either $\diamond$ or $\mathfrak{G}' \triangleright \mathcal{A}$ for some $\mathfrak{G}'$ and $\mathcal{A}$;
- every $\mathcal{A} : \overline{\mathsf{Ty}}\ \mathfrak{G}$ is either U, $\mathsf{El}\ \mathrm{X}$, $\Pi\ \mathrm{X}\ \mathcal{Y}$, or $\mathsf{Eq}\ \mathrm{X}\ \mathrm{t}\ \mathrm{t}'$.

The 'either-or's here are exclusive: $\diamond$ is not $\mathfrak{G}' \triangleright \mathcal{A}$ for any $\mathfrak{G}'$ and $\mathcal{A}$, U is not $\mathsf{El}\ \mathrm{X}$ for any X, and so on. From these kinds of observations, we get an idea of how to perform **elimination** on oneGAT, and specifically how to make a uniform definition across all GATs: define just what to do on the oneGAT constructors.

Remark 1.3.12. Throughout, we'll want to make definitions inductively on the structure of the oneGAT language (e.g. Definition 1.3.13). For instance, we'll want to define F of the form "oneGAT $\to$ Set", i.e. four components

$$\begin{aligned}
F_{\mathsf{Con}} &: \overline{\mathsf{Con}} \to \mathsf{Set} \\
F_{\mathsf{Sub}} &: \overline{\mathsf{Sub}}\ \mathfrak{H}\ \mathfrak{G} \to F_{\mathsf{Con}}(\mathfrak{H}) \to F_{\mathsf{Con}}(\mathfrak{G}) \\
F_{\mathsf{Ty}} &: \overline{\mathsf{Ty}}\ \mathfrak{G} \to F_{\mathsf{Con}}(\mathfrak{G}) \to \mathsf{Set} \\
F_{\mathsf{Tm}} &: \overline{\mathsf{Tm}}(\mathfrak{G}, \mathcal{A}) \to (\Gamma : F_{\mathsf{Con}}(\mathfrak{G})) \to F_{\mathsf{Ty}}(\mathcal{A})\ \Gamma.
\end{aligned}$$

along with witnesses that these definitions respect the required equalities, e.g.

$$F(\mathsf{assoc}) : \forall z,\ F_{\mathsf{Sub}}((\mathfrak{h} \circ \mathfrak{g}) \circ \mathfrak{f})\ z = F_{\mathsf{Sub}}(\mathfrak{h} \circ (\mathfrak{g} \circ \mathfrak{f}))\ z.$$

In principle, we'd need to define F on all 32 components of Figure 1.8. However, in practice, many of the components end up being trivial; in particular, the equality checks (like $F(\mathsf{assoc})$) will often end up being refl or UIP. Therefore, we'll generally just present the main components of interest, like $F_{\mathsf{Con}}(\mathfrak{G} \triangleright \mathcal{A})$, $F_{\mathsf{Ty}}(\mathsf{U})$, $F_{\mathsf{Ty}}(\Pi\ \mathrm{X}\ \mathcal{Y})$, and $F_{\mathsf{Tm}}(\mathsf{El}\ \mathrm{X})$.

It's worth noting that the definitions of F_{Con}, F_{Sub}, F_{Ty} and F_{Tm} must necessarily be *simultaneous*—we cannot (in general) define F_{Con} first, then F_{Ty}, etc., or any other order. This is because the specification of oneGAT given in Figure 1.8 is **inductive-inductive**: the components of $\overline{\mathsf{Ty}}$ (e.g. Π) refer to the components of $\overline{\mathsf{Con}}$ (e.g. $\triangleright$), the components of $\overline{\mathsf{Tm}}$ (e.g. π_2) refer to the components of $\overline{\mathsf{Sub}}$, and so on. We can't untangle these from each other.

As our first example, we precisely define the notion of $\mathfrak{G}$-algebra

Definition 1.3.13. For any GAT $\mathfrak{G}$, define the type $\mathfrak{G}$-Alg of **$\mathfrak{G}$-algebras** as $\mathfrak{G}^{\mathsf{A}}$,

where

$$\begin{aligned}\left(\mathfrak{G} : \overline{\mathsf{Con}}\right)^{\mathsf{A}} &: \mathsf{Set}\\ \left(\mathfrak{h} : \overline{\mathsf{Sub}}\ \mathfrak{G}\ \mathfrak{G}\right)^{\mathsf{A}} &: \mathfrak{G}^{\mathsf{A}} \to \mathfrak{H}^{\mathsf{A}}\\ \left(\mathcal{A} : \overline{\mathsf{Ty}}\ \mathfrak{G}\right)^{\mathsf{A}} &: \mathfrak{G}^{\mathsf{A}} \to \mathsf{Set}\\ \left(\mathrm{t} : \overline{\mathsf{Tm}}\ \mathfrak{G}\ \mathcal{A}\right)^{\mathsf{A}} &: (\Gamma : \mathfrak{G}^{\mathsf{A}}) \to \mathcal{A}^{\mathsf{A}}\ \Gamma.\end{aligned}$$

is defined inductively on the structure of ONEGAT:

$$\begin{aligned}&\diamond^{\mathsf{A}} && := \{\star\}\\ &(\mathfrak{G} \triangleright \mathcal{A})^{\mathsf{A}} && := \left(\Gamma : \mathfrak{G}^{\mathsf{A}}\right) \times \mathcal{A}^{\mathsf{A}}\ \Gamma\\ &\mathsf{U}^{\mathsf{A}}\ \Gamma && := \mathsf{Set}\\ &(\mathsf{El}\ \mathrm{X})^{\mathsf{A}}\ \Gamma && := \mathrm{X}^{\mathsf{A}}\ \Gamma\\ &(\Pi\ \mathrm{X}\ \mathcal{Y})^{\mathsf{A}}\ \Gamma && := \left(\mathrm{x} : \mathrm{X}^{\mathsf{A}}\ \Gamma\right) \to \mathcal{Y}^{\mathsf{A}}\ \Gamma\ \mathrm{x}\\ &(\mathsf{app}\ \mathrm{t})^{\mathsf{A}}\ (\Gamma, \mathrm{x}) && := \mathrm{t}^{\mathsf{A}}\ \Gamma\ \mathrm{x})\\ &(\mathsf{Eq}\ \mathrm{s}\ \mathrm{t})^{\mathsf{A}}\ \Gamma && := (\mathrm{s}^{\mathsf{A}}\ \Gamma = \mathrm{t}^{\mathsf{A}}\ \Gamma).\end{aligned}$$

For the full definition, see [KKA19, Appendix A].

Though we don't make this claim precise, a survey of the ONEGAT examples will indicate that the only *terms* we deal with are de Bruijn indices and applications $\mathrm{f} \mathbin{@} \mathrm{t}$ (and transports). So, in order to use a definition like $(\_)^{\mathsf{A}}$, it's helpful to figure out what $(\mathrm{f} \mathbin{@} \mathrm{t})^{\mathsf{A}}$ is in terms of f^{A} and t^{A}; and what 0^{A}, 1^{A}, 2^{A}, etc. are.

Proposition 1.3.14. *For* $\mathrm{f} : \overline{\mathsf{Tm}}(\mathfrak{G}, \Pi\ \mathrm{X}\ \mathcal{Y})$ *and* $\mathrm{t} : \overline{\mathsf{Tm}}(\mathfrak{G}, \mathsf{El}\ \mathrm{X})$ *and* $\Gamma : \mathfrak{G}\text{-}\mathsf{Alg}$,

$$(\mathrm{f} \mathbin{@} \mathrm{t})^{\mathsf{A}}\ \Gamma = \mathrm{f}^{\mathsf{A}}\ \Gamma\ (\mathrm{t}^{\mathsf{A}}\ \Gamma).$$

Proof.

$$\begin{aligned}(\mathrm{f} \mathbin{@} \mathrm{t})^{\mathsf{A}}\ \Gamma &= ((\mathsf{app}\ \mathrm{f})[\ \mathsf{id}, \mathrm{t}\ ])^{\mathsf{A}}\ \Gamma\\ &= (\mathsf{app}\ \mathrm{f})^{\mathsf{A}}\ ((\mathsf{id}, \mathrm{t})^{\mathsf{A}}\ \Gamma)\\ &= (\mathsf{app}\ \mathrm{f})^{\mathsf{A}}\ (\mathsf{id}^{\mathsf{A}}\ \Gamma, \mathrm{t}^{\mathsf{A}}\ \Gamma)\\ &= (\mathsf{app}\ \mathrm{f})^{\mathsf{A}}\ (\Gamma, \mathrm{t}^{\mathsf{A}}\ \Gamma)\\ &= \mathrm{f}^{\mathsf{A}}\ \Gamma\ (\mathrm{t}^{\mathsf{A}}\ \Gamma)\end{aligned}$$

□

Proposition 1.3.15. *For $\mathfrak{G} : \overline{\mathsf{Con}}$ of length n[a] and $m < n$ is a de Bruijn index, then for $\Gamma = (x_1, x_2, \ldots, x_n) : \mathfrak{G}^{\mathsf{A}}$,*

$$\mathrm{m}^{\mathsf{A}}\ \Gamma = x_{n-m}.$$

[a] Where "length" is defined inductively: $\diamond$ has length 0, and $\mathfrak{G} \rhd \mathcal{A}$ has length $n+1$ if $\mathfrak{G}$ has length n.

With these, we can calculate out simple cases of algebras:

Example 1.3.16.

- $\mathfrak{Set}$-Alg:

$$\begin{aligned}(\diamond \rhd \mathsf{U})^{\mathsf{A}} &= (\Gamma : \diamond^{\mathsf{A}}) \times \mathsf{U}^{\mathsf{A}}\ \Gamma \\ &= \{\star\} \times \mathsf{Set}\end{aligned}$$

- $\mathfrak{P}$-Alg:

$$\begin{aligned}(\diamond \rhd \mathsf{U} \rhd \mathsf{El}\ 0)^{\mathsf{A}} &= (\Gamma : (\diamond \rhd \mathsf{U})^{\mathsf{A}}) \times (\mathsf{El}\ 0)^{\mathsf{A}}\ \Gamma \\ &= \{\star\} \times (X : \mathsf{Set}) \times (\mathsf{El}\ 0)^{\mathsf{A}}\ (\star, X) && \text{(above)} \\ &= \{\star\} \times (X : \mathsf{Set}) \times 0^{\mathsf{A}}\ (\star, X) \\ &= \{\star\} \times (X : \mathsf{Set}) \times X && \text{(Proposition 1.3.15)}\end{aligned}$$

- $\mathfrak{N}$-Alg:

$$\begin{aligned}&(\diamond \rhd \mathsf{U} \rhd \mathsf{El}\ 0 \rhd \Pi\ 1\ (\mathsf{El}\ 2))^{\mathsf{A}} \\ &= (\Gamma : (\diamond \rhd \mathsf{U} \rhd \mathsf{El}\ 0)^{\mathsf{A}}) \times (\Pi\ 1\ (\mathsf{El}\ 2))^{\mathsf{A}}\ \Gamma \\ &= \{\star\} \times (N : \mathsf{Set}) \times (z : N) \times (\Pi\ 1\ (\mathsf{El}\ 2))^{\mathsf{A}}\ (\star, N, z) && \text{(above)} \\ &= \{\star\} \times (N : \mathsf{Set}) \times (z : N) \times ((n : 1^{\mathsf{A}}(\star, N, z)) \to (\mathsf{El}\ 2)^{\mathsf{A}}(\star, N, z, n)) \\ &= \{\star\} \times (N : \mathsf{Set}) \times (z : N) \times ((n : 1^{\mathsf{A}}(\star, N, z)) \to 2^{\mathsf{A}}(\star, N, z, n)) \\ &= \{\star\} \times (N : \mathsf{Set}) \times (z : N) \times ((n : N) \to N) && \text{(Proposition 1.3.15)} \\ &= \{\star\} \times (N : \mathsf{Set}) \times N \times (N \to N)\end{aligned}$$

- $\mathfrak{Quiv}$-Alg

$$\begin{aligned}&(\diamond \rhd \mathsf{U} \rhd \Pi\ 0\ (\Pi\ 1\ \mathsf{U}))^{\mathsf{A}} \\ &= (\Gamma : (\diamond \rhd \mathsf{U})^{\mathsf{A}}) \times (\Pi\ 0\ (\Pi\ 1\ \mathsf{U}))^{\mathsf{A}}\ \Gamma \\ &= \{\star\} \times (V : \mathsf{Set}) \times (\Pi\ 0\ (\Pi\ 1\ \mathsf{U}))^{\mathsf{A}}\ (\star, V) && \text{(above)} \\ &= \{\star\} \times (V : \mathsf{Set}) \times ((v_0 : 0^{\mathsf{A}}(\star, V)) \to (\Pi\ 1\ \mathsf{U})^{\mathsf{A}}(\star, V, v_0)) \\ &= \{\star\} \times (V : \mathsf{Set}) \times ((v_0 : V) \to (v_1 : 1^{\mathsf{A}}(\star, V, v_0) \to \mathsf{U}^{\mathsf{A}}(\star, V, v_0, v_1)) \\ &= \{\star\} \times (V : \mathsf{Set}) \times (V \to V \to \mathsf{Set})\end{aligned}$$

- $\mathfrak{rQuiv}$-Alg

$$
\begin{aligned}
&(\diamond \triangleright \mathsf{U} \triangleright \Pi\ 0\ (\Pi\ 1\ \mathsf{U}) \triangleright \Pi\ 1\ (\mathsf{El}(1 @ 0 @ 0)))^{\mathsf{A}} \\
&= (\Gamma : (\diamond \triangleright \mathsf{U} \triangleright \Pi\ 0\ (\Pi\ 1\ \mathsf{U}))^{\mathsf{A}}) \times (\Pi\ 1\ (\mathsf{El}(1 @ 0 @ 0)))^{\mathsf{A}}\ \Gamma \\
&= \{\star\} \times (V : \mathsf{Set}) \times (E : V \to V \to \mathsf{Set}) \times (\Pi\ 1\ (\mathsf{El}(1 @ 0 @ 0)))^{\mathsf{A}}\ (\star, V, E) \\
&= \{\star\} \times (V : \mathsf{Set}) \times (E : V \to V \to \mathsf{Set}) \times ((v : V) \to ((1 @ 0) @ 0)^{\mathsf{A}}(\star, V, E, v)) \\
&= \{\star\} \times (V : \mathsf{Set}) \times (E : V \to V \to \mathsf{Set}) \times ((v : V) \to E\ v\ v) \qquad \text{(below)}
\end{aligned}
$$

$$
\begin{aligned}
((1 @ 0) @ 0)^{\mathsf{A}}(\star, V, E, v) &= ((1 @ 0)^{\mathsf{A}}(\star, V, E, v))\ 0^{\mathsf{A}}(\star, V, E, v) \\
&= (1^{\mathsf{A}}(\star, V, E, v))\ (0^{\mathsf{A}}(\star, V, E, v))\ (0^{\mathsf{A}}(\star, V, E, v)) \\
&= E\ v\ v
\end{aligned}
$$

1.3.3 Initial Algebras

The construction of the initial algebra for an arbitrary GAT $\mathfrak{G}$ is rather involved, so we'll just make a few observations about the construction (which hopefully make it easier to understand) and refer the reader to [KKA19, p. 14] and [KKA19, Appendix A] for all the technical details.

The key point, which we developed informally above, is that *the initial $\mathfrak{G}$-algebra consists of the data whose existence can be derived from $\mathfrak{G}$ alone.* In other words, the initial $\mathfrak{G}$-algebra contains all—and *only*—the "stuff" which necessarily follows from the mere concept of structure encoded in $\mathfrak{G}$. We make this precise using the ONEGAT language: for every sort symbol $\mathcal{S}$ in $\mathfrak{G}$, the carrier set interpreting $\mathcal{S}$ in the initial $\mathfrak{G}$-algebra will be the ONEGAT terms-in-context $\overline{\mathsf{Tm}}\ \mathfrak{G}\ (\mathsf{El}\ \mathcal{S})$.

Consider the simple case of pointed sets, the GAT $\mathfrak{P}$—written $\{\!|\mathrm{X} : \mathrm{U},\ \mathrm{x}_0 : \mathrm{X}|\!\}$ in NOUGAT, or $\diamond \triangleright \mathsf{U} \triangleright \mathsf{El}\ 0$ in ONEGAT. A $\mathfrak{P}$-algebra consists of a set plus an element of that set; for the initial $\mathfrak{P}$-algebra, we'll take the set to be

$$\mathbb{1} := \overline{\mathsf{Tm}}\ \mathfrak{P}\ (\mathsf{El}\ 1)$$

i.e. the collection of all those terms-in-context

$$\{\!|\ \mathrm{X} : \mathrm{U},\ \mathrm{x} : \mathrm{X}\ |\!\} \vdash \mathtt{t} : \mathrm{X}.$$

Since X is completely abstract here, all we "know" about it is that it comes equipped with a point x; we will therefore be able to conclude (see below) that the ONEGAT term

$$0 : \overline{\mathsf{Tm}}\ \mathfrak{P}\ (\mathsf{El}\ 1)$$

is indeed the *only* such point, i.e. the initial pointed set is a singleton (as the notation $\mathbb{1}$ suggests).

Likewise with natural number algebras: the carrier set of the initial $\mathfrak{N}$-algebra (i.e. the natural numbers) will be

$$\mathbb{N} := \overline{\mathsf{Tm}}\ \mathfrak{N}\ (\mathsf{El}\ 2).$$

This set contains a chosen element, "zero", which (humorously) ends up being the de Bruijn index 1:

$$\mathit{zero} := 1 : \overline{\mathsf{Tm}}\ \mathfrak{N}\ (\mathsf{El}\ 2)$$

and, for the endofunction $succ : \mathbb{N} \to \mathbb{N}$, we take the operation of applying the endofunction specified in the signature $\mathfrak{N}$, i.e. the de Bruijn index $0 : \overline{\mathsf{Tm}}\ \mathfrak{N}\ (\Pi\ 1\ (\mathsf{El}\ 2))$:

$$succ := \lambda \mathrm{t} \to 0\ @\ \mathrm{t} \quad : \mathbb{N} \to \mathbb{N}.$$

This is straightforward enough to do for these simple, concrete cases. But it's much more involved to define this for generic $\mathfrak{G}$. In particular, we note that it's not possible to define the initial $\mathfrak{G}$-algebra for all $\mathfrak{G}$ by induction on *just* $\mathfrak{G}$. This is because the definition is not straightforwardly compositional: by adding additional constructors to a GAT, the character of its initial algebra can change drastically. Consider: we can extend the GAT $\mathfrak{P}$ of pointed sets to the GAT $\mathfrak{N}$ of natural number algebras by the addition of just one constructor. But the carrier sets for their initial algebras,

$$\overline{\mathsf{Tm}}\ \mathfrak{P}\ (\mathsf{El}\ 1) \qquad \text{and} \qquad \overline{\mathsf{Tm}}\ \mathfrak{N}\ (\mathsf{El}\ 2),$$

respectively, aren't related in any obvious way—we don't obtain the latter as some modification of the former. More abstractly: say, $\mathfrak{F} = \mathfrak{G} \rhd \mathsf{U} \rhd \mathcal{T}$ for some telescope $\mathcal{T}$, and write $\mathcal{S}$ for the length of $\mathcal{T}$, i.e. the de Bruijn index picking out the sort symbol appearing between $\mathfrak{G}$ and $\mathcal{T}$. We want to say that the set interpreting the sort $\mathcal{S}$ in the initial $\mathfrak{F}$-algebra is

$$\overline{\mathsf{Tm}}\ \mathfrak{F}\ (\mathsf{El}\ \mathcal{S}).$$

But if we were defining initial algebras by straightforward induction, then the entirety of $\mathfrak{F}$ wouldn't be in scope throughout the induction: when defining the initial algebra interpretation of $\mathcal{S}$, we would only have access to $\mathfrak{G}$. But this is rather silly: it's only in the telescope $\mathcal{T}$ that the sort symbol $\mathcal{S}$ is even available, so any constructor for terms of this sort must be in $\mathcal{T}$. So, the moral of this story: we need to keep the *entire* GAT in scope throughout the induction, because every component is (potentially) relevant in producing terms.

Accordingly, KKA generalize the induction, fixing some GAT $\mathfrak{F}$ and then constructing

$$\begin{aligned}
(\mathfrak{G} : \overline{\mathsf{Con}})^{\mathsf{C}} &: (\mathfrak{g} : \overline{\mathsf{Sub}}\ \mathfrak{F}\ \mathfrak{G}) \to \mathfrak{G}^{\mathsf{A}} \\
(\mathcal{A} : \overline{\mathsf{Ty}}\ \mathfrak{G})^{\mathsf{C}} &: (\mathfrak{g} : \overline{\mathsf{Sub}}\ \mathfrak{F}\ \mathfrak{G}) \to \overline{\mathsf{Tm}}\ \mathfrak{F}\ (\mathcal{A}[\mathfrak{g}]) \to \mathcal{A}^{\mathsf{A}}(\mathfrak{G}^{\mathsf{C}}\ \mathfrak{g}) \\
(\mathfrak{h} : \overline{\mathsf{Sub}}\ \mathfrak{G}\ \mathfrak{H})^{\mathsf{C}} &: (\mathfrak{g} : \overline{\mathsf{Sub}}\ \mathfrak{F}\ \mathfrak{G}) \to \mathfrak{H}^{\mathsf{C}}(\mathfrak{h} \circ \mathfrak{g}) = \mathfrak{h}^{\mathsf{A}}(\mathfrak{G}^{\mathsf{C}}\mathfrak{g}) \\
(\mathrm{t} : \overline{\mathsf{Tm}}\ \mathfrak{G}\ \mathcal{A})^{\mathsf{C}} &: (\mathfrak{g} : \overline{\mathsf{Sub}}\ \mathfrak{F}\ \mathfrak{G}) \to \mathcal{A}^{\mathsf{C}}\ \mathfrak{g}\ (\mathrm{t}[\mathfrak{g}]) = \mathrm{t}^{\mathsf{A}}(\mathfrak{G}^{\mathsf{C}}\mathfrak{g})
\end{aligned}$$

by quotient induction-induction on the onegat structure. Then, at the end of the day, the initial $\mathfrak{F}$-algebra can be obtained as $\mathfrak{F}^{\mathsf{C}}(\mathrm{id}_{\mathfrak{F}})$. This allows us to keep the entire GAT (in this case, $\mathfrak{F}$) constantly in scope as we perform the induction ($\mathfrak{G}$ is the variable of induction here), as is necessary.

In particular, the key clause of this definition is

$$\mathsf{U}^{\mathsf{C}}\ (\mathfrak{g} : \overline{\mathsf{Sub}}\ \mathfrak{F}\ \mathfrak{G})\ (\mathrm{X} : \overline{\mathsf{Tm}}\ \mathfrak{F}\ (\mathsf{U}[\mathfrak{g}])) \quad := \quad \overline{\mathsf{Tm}}\ \mathfrak{F}\ (\mathsf{El}\ \mathrm{X}).$$

Notice that the right-hand side doesn't depend on $\mathfrak{G}$: as we said above, we want the whole GAT $\mathfrak{F}$ to be used when interpreting the sorts as sets of terms, not the variable of induction, $\mathfrak{G}$.

Let us note that, in defining $(\_)^{\mathsf{C}}$, KKA explicitly note their metatheoretic *coercions* (though these often end up being trivial in usual cases). For instance, one clause of the definition is

$$(\mathsf{El}\ \mathrm{X})^{\mathsf{C}}\ (\mathfrak{g} : \overline{\mathsf{Sub}}\ \mathfrak{F}\ \mathfrak{G})\ (\mathrm{t} : \overline{\mathsf{Tm}}\,\mathfrak{F}\,((\mathsf{El}\ \mathrm{X})[\mathfrak{g}])) \quad := \quad \mathsf{coe}\ (\mathrm{X}^{\mathsf{C}}\mathfrak{g})\ \mathrm{t}.$$

What's happening here is that we are given

$$\mathrm{t} \quad : \quad \overline{\mathsf{Tm}}\,\mathfrak{F}\,((\mathsf{El}\ \mathrm{X})[\mathfrak{g}])$$

and we need to produce

$$(\mathsf{El}\ \mathrm{X})^{\mathsf{C}}\ \mathfrak{g}\ \mathrm{t} \quad : \quad (\mathsf{El}\ \mathrm{X})^{\mathsf{A}}(\mathfrak{G}^{\mathsf{C}}\mathfrak{g}).$$

But, we can also make use of $\mathrm{X}^{\mathsf{C}}(\mathfrak{g})$, which, since $\mathrm{X} : \overline{\mathsf{Tm}}\,\mathfrak{G}\,\mathsf{U}$, is a (metatheoretic) equation

$$\mathsf{U}^{\mathsf{C}}\ \mathfrak{g}\ (\mathrm{X}[\mathfrak{g}]) = \mathrm{X}^{\mathsf{A}}(\mathfrak{G}^{\mathsf{C}}\mathfrak{g})$$

i.e.

$$\overline{\mathsf{Tm}}\,\mathfrak{F}\,((\mathsf{El}\ \mathrm{X})[\mathfrak{g}]) = \mathrm{X}^{\mathsf{A}}(\mathfrak{G}^{\mathsf{C}}\mathfrak{g}).$$

Notice the left-hand side is the type of t and the right-hand side is the desired type of $(\mathsf{El}\ \mathrm{X})^{\mathsf{C}}\ \mathfrak{g}\ \mathrm{t}$ (recall that $(\mathsf{El}\ \mathrm{X})^{\mathsf{A}}\ \Gamma = \mathrm{X}^{\mathsf{A}}\ \Gamma$). We've generally been treating coercions (and transports) in the metatheory as happening implicitly, so we would just write $(\mathsf{El}\ \mathrm{X})^{\mathsf{C}}\ \mathfrak{g}\ \mathrm{t} := \mathrm{t}$, but it's worth noting that this only makes sense by virtue of $\mathrm{X}^{\mathsf{C}}(\mathfrak{g})$.

For demonstration, we conclude with the calculations of the initial algebras for a few choice GATs. Here, all metatheoretic transports and coercions are made implicit.

Example 1.3.17.

$$\begin{aligned}
\mathfrak{Set}^{\mathsf{C}}\ \mathsf{id}_{\mathfrak{Set}} &= (\diamond \triangleright \mathsf{U})^{\mathsf{C}}\ \mathsf{id} \\
&= (\diamond^{\mathsf{C}}(\pi_1\ \mathsf{id}), \mathsf{U}^{\mathsf{C}}\ (\pi_1\ \mathsf{id})\ (\pi_2\ \mathsf{id})) \\
&= (\star, \overline{\mathsf{Tm}}\ \mathfrak{Set}\ (\mathsf{El}\ 0))
\end{aligned}$$

$$\begin{aligned}
\mathfrak{P}^{\mathsf{C}}\ \mathsf{id}_{\mathfrak{P}} &= (\diamond \triangleright \mathsf{U} \triangleright (\mathsf{El}\ 0))^{\mathsf{C}}\ \mathsf{id} \\
&= ((\diamond \triangleright \mathsf{U})^{\mathsf{C}}(\pi_1\ \mathsf{id}), (\mathsf{El}\ 0)^{\mathsf{C}}\ (\pi_1\ \mathsf{id})\ (\pi_2\ \mathsf{id})) \\
&= (\diamond^{\mathsf{C}}(\pi_1(\pi_1\ \mathsf{id})), \mathsf{U}^{\mathsf{C}}\ (\pi_1(\pi_1\ \mathsf{id}))\ (\pi_2(\pi_1\ \mathsf{id})), (\mathsf{El}\ 0)^{\mathsf{C}}\ (\pi_1\ \mathsf{id})\ 0) \\
&= (\star, \mathsf{U}^{\mathsf{C}}\ (\pi_1(\pi_1\ \mathsf{id}))\ 1, 0) \\
&= (\star, \overline{\mathsf{Tm}}\,\mathfrak{P}\ (\mathsf{El}\ 1), 0)
\end{aligned}$$

$$\begin{aligned}
\mathfrak{N}^{\mathsf{C}}\ \mathsf{id}_{\mathfrak{N}} &= (\diamond \triangleright \mathsf{U} \triangleright \mathsf{El}\ 0 \triangleright \Pi\ 1\ (\mathsf{El}\ 2))^{\mathsf{C}}\ \mathsf{id} \\
&= (\diamond \triangleright \mathsf{U} \triangleright (\mathsf{El}\ 0))^{\mathsf{C}}(\pi_1\ \mathsf{id}), (\Pi\ 1\ (\mathsf{El}\ 2))^{\mathsf{C}}\ (\pi_1\ \mathsf{id})\ 0) \\
&= (\star, \overline{\mathsf{Tm}}\ \mathfrak{N}\ (\mathsf{El}\ 2), 1, (\Pi\ 1\ (\mathsf{El}\ 2))^{\mathsf{C}}\ (\pi_1\ \mathsf{id})\ 0) \\
&= (\star, \overline{\mathsf{Tm}}\ \mathfrak{N}\ (\mathsf{El}\ 2), 1, \lambda \mathrm{t} \to (\mathsf{El}\ 2)^{\mathsf{C}}(\pi_1\ \mathsf{id}, \mathrm{t})\ (0 \mathbin{@} \mathrm{t})) \\
&= (\star, \overline{\mathsf{Tm}}\ \mathfrak{N}\ (\mathsf{El}\ 2), 1, \lambda \mathrm{t} \to 0 \mathbin{@} \mathrm{t})
\end{aligned}$$

1.3.4 Concrete CwFs

Let's return to our motivating example: the natural numbers. We have syntax for writing down the abstract concept of the natural numbers as the GAT $\mathfrak{N}$. This GAT provides a recipe for various kinds of structures: the type $\mathfrak{N}$-Alg of $\mathfrak{N}$-algebras, homomorphisms of $\mathfrak{N}$-algebras, the initial $\mathfrak{N}$-algebra $\mathbb{N}$, etc. All these structures have the same basic shape in common, matching the shape of the GAT $\mathfrak{N}$:

- Data corresponding to the sort `Nat : U`. For $\mathfrak{N}$-algebras, this was a set (for the initial algebra, a particular set); for nat-algebra homomorphisms, it was a function.
- Data corresponding to the element `zero : Nat`. For algebras, this was an element; for homomorphisms it was an equation saying that the function preserved the chosen element.
- Data corresponding to the function `succ : Nat ⇒ Nat`. For algebras, this was an operation on the carrier set; for homomorphisms, it was a proof that the underlying function commuted with the respective endofunctions.

We can extend this story to cover the process of *induction*. Recall that the "input" to induction also has the same "$\mathfrak{N}$-shape":

pseudoAgda

```
record indData : Set where
  P : ℕ → Set
  BC : P zero
  IS : (n : ℕ) → P n → P (succ n)
```

Here, the data corresponding to `Nat` is instead a *predicate* on $\mathbb{N}$; the data corresponding to `zero` is a witness of this predicate holds of the element $zero : \mathbb{N}$; and the data corresponding to `succ` is an operation transporting witnesses of the predicate across the function $succ : \mathbb{N} \to \mathbb{N}$. Now, this **logical predicate** structure is stated with respect to the initial $\mathfrak{N}$-algebra, but we can formulate it for an arbitrary $\mathfrak{N}$-algebra as follows.

Definition 1.3.18. A **displayed $\mathfrak{N}$-algebra** on a $\mathfrak{N}$-algebra (N, z, s) consists of

- a family of sets $N^{\mathsf{D}} : N \to \mathrm{Set}$;
- an element $z^{\mathsf{D}} : N^{\mathsf{D}}\ z$; and
- a dependent function $s^{\mathsf{D}} : (n : N) \to N^{\mathsf{D}}\ n \to N^{\mathsf{D}}(s\ n)$.

Write $(N^{\mathsf{D}}, z^{\mathsf{D}}, s^{\mathsf{D}}) : \mathfrak{N}\text{-DAlg}(N, z, s)$ to mean that $(N^{\mathsf{D}}, z^{\mathsf{D}}, s^{\mathsf{D}})$ is a displayed $\mathfrak{N}$-algebra on (N, z, s).

The word "displayed" is taken from Ahrens and Lumsdaine [AL19]. It reflects the intuition that a proof-relevant predicate $N^{\mathsf{D}} : N \to \mathrm{Set}$ can be pictured as a family of sets "over" the set N, and the additional structure $z^{\mathsf{D}}, s^{\mathsf{D}}$ making this into a displayed algebra is likewise *displayed over* the data in the "base" algebra (N, z, s). Now, Ahrens and Lumsdaine do not introduce displayed nat-algebras; rather, they study the displayed algebras for a different GAT: *categories*.

Definition 1.3.19. A **displayed category** on a category Γ is given by the following data.

PSEUDOAGDA

```
record 𝔠𝔞𝔱-DAlg (Γ : 𝔠𝔞𝔱-Alg) : Set where
  Obj^D : |Γ| → Set
  Hom^D : {γ₀ γ₁ : |Γ|} → Obj^D γ₀ → Obj^D γ₁ → Γ [γ₀,γ₁] → Set
  id^D : {γ : |Γ|} → (γ^D : Obj^D γ) → Hom^D γ^D γ^D id_γ
  comp^D : {γ₀ γ₁ γ₂ : |Γ|}{γ₀₁ : Γ [ γ₀ , γ₁ ]}{γ₁₂ : Γ [ γ₁ , γ₂ ]} →
    {γ₀^D : Obj^D γ₀}{γ₁^D : Obj^D γ₁}{γ₂^D : Obj^D γ₂} →
    Hom^D γ₁^D γ₂^D γ₁₂ → Hom^D γ₀^D γ₁^D γ₀₁ →
    Hom^D γ₀^D γ₂^D (γ₁₂ ∘ γ₀₁)
  idr^D : {γ₀}{γ₁}{γ₀^D}{γ₁^D}{γ₀₁} → (γ₀₁^D : Hom^D γ₀^D γ₁^D γ₀₁) →
    comp^D (id^D γ₁^D) γ₀₁^D = γ₀₁^D
  idl^D : {γ₀}{γ₁}{γ₀^D}{γ₁^D}{γ₀₁} → (γ₀₁^D : Hom^D γ₀^D γ₁^D γ₀₁) →
    comp^D γ₀₁^D (id^D γ₀^D) = γ₀₁^D
  ass^D : {γ₀}{γ₁}{γ₂}{γ₃}{γ₀^D}{γ₁^D}{γ₂^D}{γ₃^D}{γ₀₁}{γ₁₂}{γ₂₃} →
    (γ₀₁^D : Hom^D γ₀^D γ₁^D γ₀₁) → (γ₁₂^D : Hom^D γ₁^D γ₂^D γ₁₂) →
    (γ₂₃^D : Hom^D γ₂^D γ₃^D γ₂₃) →
    comp^D γ₂₃^D (comp^D γ₁₂^D γ₀₁^D) = comp^D (comp^D γ₂₃^D γ₁₂^D) γ₀₁^D
```

Given a displayed category A over Γ, we'll sometimes write $A[\gamma_{01} \mid a_0, a_1]$ for $A.\mathsf{Hom}^D\ a_0\ a_1\ \gamma_{01}$ and write $\_ \circ^D \_$ for the displayed composition operator $A.\mathsf{comp}^D$.

We've done the same thing here as in Definition 1.3.18, but replacing $\mathfrak{N}$ with $\mathfrak{Cat}$: for every sort, a predicate; for every element, a witness that the predicate holds of that element; for every function, an appropriate transformation of witnesses. This kind of definition can be made generically for any GAT.

Definition 1.3.20. For any GAT $\mathfrak{G}$ and $\Gamma : \mathfrak{G}\text{-Alg}$, define the type $\mathfrak{G}\text{-DAlg}(\Gamma)$ of **displayed $\mathfrak{G}$-algebras** on Γ as $\mathfrak{G}^D(\Gamma)$, where

$$\left(\mathfrak{G} : \overline{\mathsf{Con}}\right)^D : \mathfrak{G}^A \to \mathsf{Set}$$

$$\left(\mathfrak{h} : \overline{\mathsf{Sub}}\ \mathfrak{G}\ \mathfrak{H}\right)^D : \{\Gamma : \mathfrak{G}\text{-Alg}\} \to \mathfrak{G}^D\ \Gamma \to \mathfrak{H}^D(\mathfrak{h}^A\ \Gamma)$$

$$\left(\mathcal{A} : \overline{\mathsf{Ty}}\ \mathfrak{G}\right)^D : \{\Gamma : \mathfrak{G}\text{-Alg}\} \to \mathfrak{G}^D\ \Gamma \to \mathcal{A}^A\ \Gamma \to \mathsf{Set}$$

$$\left(\mathrm{t} : \overline{\mathsf{Tm}}\ \mathfrak{G}\ \mathcal{A}\right)^D : \{\Gamma : \mathfrak{G}\text{-Alg}\} \to (\Gamma^D : \mathfrak{G}^D\ \Gamma) \to \mathcal{A}^D\ \Gamma^D\ (\mathrm{t}^A\ \Gamma)$$

is defined inductively on the structure of oneGAT:

$$
\begin{aligned}
(\mathfrak{G} \triangleright \mathcal{A})^{\mathsf{D}}\ (\Gamma, \alpha) \quad &:= (\Gamma^{\mathsf{D}} : \mathfrak{G}^{\mathsf{D}}\ \Gamma) \times \mathcal{A}^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ \alpha \\
\mathsf{U}^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ T \quad &:= T \to \mathsf{Set} \\
(\mathsf{El}\ \mathrm{X})^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ \mathrm{x} \quad &:= \mathrm{X}^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ \mathrm{x} \\
(\Pi\, \mathrm{X}\, \mathcal{Y})^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ f \quad &:= \{\mathrm{x} : \mathrm{X}^{\mathsf{A}}\ \Gamma\} \to (\mathrm{x}^{\mathsf{D}} : \mathrm{X}^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ \mathrm{x}) \to \mathcal{Y}^{\mathsf{D}}\ (\Gamma^{\mathsf{D}}, \mathrm{x}^{\mathsf{D}})\ (f\ \mathrm{x}) \\
(\mathsf{Eq}\ \mathrm{s}\ \mathrm{t}')^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ \mathrm{eq} \quad &\mathsf{tr}\ \mathrm{eq}\ (\mathrm{s}^{\mathsf{D}}\ \Gamma^{\mathsf{D}}) = (\mathrm{t}^{\mathsf{D}}\ \Gamma^{\mathsf{D}})
\end{aligned}
$$

For the full definition, see [KKA19, Appendix A].

In the present work, we'll mainly be interested in displayed categories, and closely-related examples (displayed setoids, preorders, and groupoids). But the notion of displayed structure is an interesting and flexible one, about which much can be said. For instance, as a brief tangent, let us note that *proof-irrelevant* displayed algebras (i.e. displayed algebras where all the predicates take values in Prop rather than Set) correspond to **sub-algebras**. The $\mathfrak{G} = \mathfrak{Grp}$ case is paradigmatic: a proof-irrelevant displayed group on some group $(G, e, *, i)$ consists of a predicate $G^{\mathsf{D}} : G \to \mathsf{Prop}$ along with

- $e^{\mathsf{D}} : G^{\mathsf{D}}\ e$
- $*^{\mathsf{D}} : (g\ g' : G) \to G^{\mathsf{D}}\ g \to G^{\mathsf{D}}\ g' \to G^{\mathsf{D}}(g * g')$.
- $i^{\mathsf{D}} : (g : G) \to G^{\mathsf{D}}\ g \to G^{\mathsf{D}}(i\ g)$

The other components are trivial (they're equalities between elements of $G^{\mathsf{D}}\ g$, which we assume to be a Prop). Now, these are exactly the requirements of a *subgroup* H of G (with $G^{\mathsf{D}}g$ being the predicate that $g \in H$): it contains the identity element, and it's closed under the group operation and group inverses. So this makes precise the idea that a "sub-structure" is a structure that is "closed under all the relevant operations". We won't develop this idea further, but there will be a few places where we speak of sub-algebras (e.g. in section 1.4 and Proposition 2.2.1), where we could achieve greater generality by explicitly considering proof-irrelevant sub-algebras.

But instead we're going to consider the other half of the process of natural number induction: *the output*. If we supply a displayed $\mathfrak{N}$-algebra (P, BC, IS) over $(\mathbb{N}, zero, succ)$, then the principle of mathematical induction says we get a dependent function of shape

$$\mathsf{elim} : (n : \mathbb{N}) \to P(n)$$

with the particular property that $\mathsf{elim}\ zero = BC$ and $\mathsf{elim}(succ\ n) = IS(n, \mathsf{elim}\ n)$. This is also $\mathfrak{N}$-shaped: now the `Nat` sort corresponds to the elim-function, which is some kind of **section** of the predicate P, and the `zero` and `succ` components correspond to the β-laws of the induction principle just mentioned. Once again, this notion can be generalized to any GAT.

Definition 1.3.21. For any GAT $\mathfrak{G}$, any $\Gamma : \mathfrak{G}$-Alg, and any $\Gamma^{\mathsf{D}} : \mathfrak{G}^{\mathsf{D}}\ \Gamma$, define the

type $\mathfrak{G}$-Sect Γ Γ^{D} of **sections** of Γ^{D} as $\mathfrak{G}^{\mathsf{S}}$ Γ Γ^{D}, where

$$\left(\mathfrak{G} : \overline{\mathsf{Con}}\right)^{\mathsf{S}} : (\Gamma : \mathfrak{G}^{\mathsf{A}}) \to \mathfrak{G}^{\mathsf{D}}\ \Gamma \to \mathsf{Set}$$
$$\left(\mathfrak{h} : \overline{\mathsf{Sub}}\ \mathfrak{G}\ \mathfrak{H}\right)^{\mathsf{S}} : \{\Gamma : \mathfrak{G}\text{-}\mathsf{Alg}\}\{\Gamma^{\mathsf{D}} : \mathfrak{G}^{\mathsf{D}}\ \Gamma\} \to \mathfrak{G}^{\mathsf{S}}\ \Gamma\ \Gamma^{\mathsf{D}} \to \mathfrak{H}^{\mathsf{S}}\ (\mathfrak{h}^{\mathsf{A}}\ \Gamma)\ (\mathfrak{h}^{\mathsf{D}}\ \Gamma^{\mathsf{D}})$$
$$\left(\mathcal{A} : \overline{\mathsf{Ty}}\ \mathfrak{G}\right)^{\mathsf{S}} : \{\Gamma\}\{\Gamma^{\mathsf{D}}\} \to \mathfrak{G}^{\mathsf{S}}\ \Gamma\ \Gamma^{\mathsf{D}} \to (\alpha : \mathcal{A}^{\mathsf{A}}\ \Gamma) \to \mathcal{A}^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ \alpha \to \mathsf{Set}$$
$$\left(\mathrm{t} : \overline{\mathsf{Tm}}\ \mathfrak{G}\ \mathcal{A}\right)^{\mathsf{S}} : \{\Gamma\}\{\Gamma^{\mathsf{D}}\} \to (\Gamma^{\mathsf{S}} : \mathfrak{G}^{\mathsf{S}}\ \Gamma\ \Gamma^{\mathsf{D}}) \to \mathcal{A}^{\mathsf{S}}\ \Gamma^{\mathsf{S}}\ (\mathrm{t}^{\mathsf{A}}\ \Gamma)\ (\mathrm{t}^{\mathsf{D}}\ \Gamma^{\mathsf{D}})$$

is defined inductively on the structure of ONEGAT:

$$\begin{aligned}
(\mathfrak{G} \triangleright \mathcal{A})^{\mathsf{S}}\ (\Gamma, \alpha)\ (\Gamma^{\mathsf{D}}, \alpha^{\mathsf{D}}) &:= (\Gamma^{\mathsf{S}} : \mathfrak{G}^{\mathsf{S}}\ \Gamma\ \Gamma^{\mathsf{D}}) \times \mathcal{A}^{\mathsf{S}}\ \Gamma^{\mathsf{S}}\ \alpha\ \alpha^{\mathsf{D}} \\
\mathsf{U}^{\mathsf{S}}\ \Gamma^{\mathsf{S}}\ T\ T^{\mathsf{D}} &:= (x : T) \to T^{\mathsf{D}}\ x \\
(\mathsf{El}\ \mathrm{X})^{\mathsf{S}}\ \Gamma^{\mathsf{S}}\ \mathrm{x}\ \mathrm{x}^{\mathsf{D}} &:= \mathrm{X}^{\mathsf{S}}\ \Gamma^{\mathsf{S}}\ \mathrm{x} = \mathrm{x}^{\mathsf{D}} \\
(\Pi\ \mathrm{X}\ \mathcal{Y})^{\mathsf{S}}\ \Gamma^{\mathsf{S}}\ f\ f^{\mathsf{D}} &:= (\mathrm{x} : \mathrm{X}^{\mathsf{A}}\ \Gamma) \to \mathcal{Y}^{\mathsf{S}}\ (\Gamma^{\mathsf{S}}, \mathsf{refl})\ (f\ \mathrm{x})\ (f^{\mathsf{D}}(\mathrm{X}^{\mathsf{S}}\ \Gamma^{\mathsf{S}}\ \mathrm{x}))
\end{aligned}$$

For the full definition, see [KKA19, Appendix A].

The notion of section takes the output of natural number induction, an abstracts *what kind of thing it is* to make sense for an arbitrary GAT. But we can go further: every GAT admits a principle of induction.

Proposition 1.3.22. *[KKA19, p. 17] For every GAT $\mathfrak{G}$ and every displayed $\mathfrak{G}$-algebra Γ^{D} over the initial $\mathfrak{G}$-algebra $\mathbb{G}$, there is a section*

$$\mathsf{elim}_{\mathfrak{G}}\ \Gamma^{\mathsf{D}} \quad : \quad \mathfrak{G}\text{-}\mathsf{Sect}\ \mathbb{G}\ \Gamma^{\mathsf{D}}.$$

The proof of this claim, like the construction of initial algebras themselves, requires a more generalized quotient induction-induction. The $\overline{\mathsf{Con}}$ motive used to define elim for a fixed GAT $\mathfrak{F}$ and a fixed displayed algebra Φ^{D} on the initial $\mathfrak{F}$-algebra $\mathbb{F}$ is

$$\mathfrak{G}^{\mathsf{E}} : (\mathfrak{g} : \overline{\mathsf{Sub}}\ \mathfrak{F}\ \mathfrak{G}) \to \mathfrak{G}^{\mathsf{S}}\ (\mathfrak{g}^{\mathsf{A}}\ \mathbb{F})\ (\mathfrak{g}^{\mathsf{D}}\ \Phi^{\mathsf{D}}).$$

Then, $\mathfrak{F}^{\mathsf{E}}(\mathsf{id}_{\mathfrak{F}})$ gives us an element of $\mathfrak{F}^{\mathsf{S}}\ \mathbb{F}\ \Phi^{\mathsf{D}}$, as desired. We once again refer the reader to [KKA19, Appendix A] for the full details of $(\_)^{\mathsf{E}}$.

This result allows us to perform some **parametricity**-style reasoning about these initial algebras. For instance, we said above that the initial pointed set was a singleton set (and denoted it $\mathbb{1}$. Our informal reasoning was that this set consists of terms-in-context

$$\{\![\ \mathrm{X} : \mathrm{U},\ \mathrm{x} : \mathrm{X}\ ]\!\} \vdash \mathrm{t} : \mathrm{X}$$

and that the type X here is completely **abstract**—we don't "know" anything about it besides that it comes equipped with x. Therefore, there's no other way to construct terms of type X, so it must be just x. Similarly, the initial $\mathfrak{Set}$-algebra is the set of terms-in-context where we don't even have x:

$$\{\![\ \mathrm{X} : \mathrm{U}\ ]\!\} \vdash \mathrm{t} : \mathrm{X}.$$

Here, the abstractness of X means we have *no* way to construct such a `t`, so this set should be empty. And in the case of $\mathfrak{N}$, we think that the carrier set of the initial algebra, $\mathbb{N}$, the set of terms-in-context

$$\{\!|\texttt{Nat : U, zero : Nat, succ : Nat} \Rightarrow \texttt{Nat}|\!\} \vdash \texttt{t} : \texttt{Nat}$$

should just consist of the *standard natural numbers*: our only way to make terms `t` like this should be by repeatedly applying `succ` to `zero`.

Reynolds [Rey83] famously made this kind of "abstractness" precise by way of *logical relations*: from the fact that the constructs of the language appropriately preserve logical relations, we can rigorously reason about what kinds of terms *can* and *cannot* be constructed. Here, we're not dealing with logical *relations*, but rather logical *predicates*—the displayed algebras. Proposition 1.3.22 is thus not a statement of *relational parametricity*, but **unary parametricity**. But unary parametricity is still a powerful tool for these kinds of arguments.

Corollary 1.3.23. *The initial $\mathfrak{Set}$-algebra is the empty set.*

Proof. Write $\mathbb{Set}$ for the initial set. Define the displayed $\mathfrak{Set}$-algebra $P : \mathbb{Set} \to \mathrm{Set}$ by

$$P(x) := \emptyset.$$

This is a displayed $\mathfrak{Set}$-algebra on $\mathbb{Set}$—there are no other components to respect. So, by Proposition 1.3.22, obtain a section

$$\mathsf{elim}\ P \quad : \mathbb{Set} \to \emptyset.$$

Therefore, $\mathbb{Set}$ must be empty. □

Corollary 1.3.24. *Writing $(\mathbb{1}, x_0)$ for the initial $\mathfrak{P}$-algebra, we have that*

$$\mathbb{1} = \{x_0\}.$$

Proof. Consider the predicate $P : \mathbb{1} \to \mathrm{Set}$ given by

$$P(x) := (x = x_0)$$

(where we interpret the right-hand side as the set $\{\star \mid x = x_0\}$, i.e. the empty set if $x \neq x_0$ and the singleton set $\{\star\}$ if $x = x_0$). This is a displayed $\mathfrak{P}$-algebra, because we have

$$\star : P(x_0);$$

therefore, by Proposition 1.3.22 obtain

$$\mathsf{elim}\ P \quad : \quad (x : \mathbb{1}) \to x = x_0$$

and conclude $\mathbb{1} = \{x_0\}$. □

Proving that $\mathbb{N}$ consists of exactly the standard natural numbers is a bit more subtle, since we can't write infinite disjunctions like

$$(x = zero) \vee (x = succ(zero)) \vee (x = succ(succ(zero))) \vee \cdots$$

We can, however, do the following.

Corollary 1.3.25. *For every $n : \mathbb{N}$, either $n = zero$ or $n = succ(n')$ for some $n' : \mathbb{N}$.*

Proof. This predicate,

$$P(x) := (x = zero) \vee ((x' : \mathbb{N}) \times x = succ(x'))$$

is a displayed $\mathfrak{N}$-algebra over $(\mathbb{N}, zero, succ)$:

- $zero = zero$, so we have $P(zero)$;
- for any $n : \mathbb{N}$, we can prove $P(succ\ n)$ by the right disjunct, putting $x' = n$ (we don't even need the inductive hypothesis, $P(n)$).

So obtain the claim by Proposition 1.3.22. □

We could abstract the claim "all displayed algebras admit a section" to be a property of a given algebra; that is, we could say $\Gamma : \mathfrak{G}\text{-Alg}$ is "inductive" if every displayed algebra Γ^{D} over Γ has a section $elim\ \Gamma\ \Gamma^{\mathsf{D}}$. Then Proposition 1.3.22 is just the claim that the initial algebra in particular is always "inductive" in this sense. There's some use to this, particularly when $\mathfrak{G}$ encodes a type-theoretic language (i.e. is an extension of $\mathfrak{CwF}$): we might say that a CwF $\mathcal{C}$ is **contextual** when it's possible to perform *induction on contexts*, i.e. every displayed CwF over $\mathcal{C}$ has a section. This notion of "contextuality" is formulated differently than the standard notion (see e.g. [CCD21, Defn. 2]), which instead insists on an appropriate *length function* $\ell : \mathsf{Con} \to \mathbb{N}$. Such a length function can easily be defined when $\mathcal{C}$ is contextual in our sense: just construct the displayed CwF with $\mathsf{Con}^{\mathsf{D}} := \mathbb{N}$ and $\bullet^{\mathsf{D}} := 0$ and $(\Gamma \triangleright A)^{\mathsf{D}}\ \Gamma^{\mathsf{D}}\ A^{\mathsf{D}} := \Gamma^{\mathsf{D}} + 1$ (the other motives—$\mathsf{Sub}^{\mathsf{D}}$, Ty^{D}, and Tm^{D}—are trivial).

Now consider the following construction: say we have two categories, Δ and Γ, a functor $\sigma : \Delta \Rightarrow \Gamma$ and a displayed category A over Γ. Then, what we can do is *reindex* A along σ to get a displayed category over Δ; we'll suggestively call this $A[\sigma] : \mathfrak{Cat}\text{-DAlg}\ \Delta$. $A[\sigma]$ is defined by

$$\begin{aligned} A[\sigma].\mathsf{Obj}^{\mathsf{D}}\ \delta &:= A.\mathsf{Obj}^{\mathsf{D}}\ (\sigma\ \delta) \\ (A[\sigma])[\delta_{01} \mid a_0, a_1] &:= A[\sigma(\delta_{01}) \mid a_0, a_1] \qquad (a_i : A.\mathsf{Obj}^{\mathsf{D}}\ (\sigma\ \delta_i) \text{ for } i = 0, 1) \end{aligned}$$

and so on. What this tells us is that $\mathfrak{Cat}$-DAlg is not just an operation sending categories to sets, but is indeed a *presheaf* on Cat. Moreover, we can give a contravariant "morphism part" to $\mathfrak{Cat}$-Sect as well: given a section $t : \mathfrak{Cat}\text{-Sect}\ \Gamma\ A$ and a functor $\sigma : \Delta \Rightarrow \Gamma$, we can similarly precompose t by σ to obtain some $t[\sigma] : \mathfrak{Cat}\text{-Sect}\ \Delta\ A[\sigma]$.

As our notation in the previous paragraph is beginning to suggest, we have the beginnings of a CwF here: one whose contexts are categories, substitutions are functors, types are *displayed categories*, and terms are *sections*. Once again, this is not unique to the GAT $\mathfrak{Cat}$, but indeed can be made generically across GATs.

Definition 1.3.26. For any GAT $\mathfrak{G}$, write $\mathscr{C}(\mathfrak{G})$ for the CwF whose
- Contexts Γ are $\mathfrak{G}$-algebras;
- Substitutitons $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$ are $\mathfrak{G}$-algebra homomorphisms from Δ to Γ;
- Types $A : \mathsf{Ty}\ \Gamma$ are displayed $\mathfrak{G}$-algebras over Γ;
- Terms $t : \mathsf{Tm}(\Gamma, A)$ are sections $\mathfrak{G}$-Sect Γ A.

The morphism parts of the Ty and Tm presheaves are defined by pre-composition, analogously to the above. The remaining components—the terminal object, context extension, etc.—can be defined by quotient induction-induction on $\mathfrak{G}$.

Remark 1.3.27. The original impetus of Ahrens and Lumsdaine [AL19] in defining displayed categories was to provide an alternative formulation for diagrams (of categories) of the form

$$\begin{array}{c} E \\ \downarrow \\ B. \end{array}$$

The data of a category E plus such a functor $E \to B$ is equivalent to a displayed category over B.

We go the other way around: given a displayed category A over Γ, we want to define the category (i.e. context in $\mathscr{C}(\mathfrak{Cat})$), $\Gamma \rhd A$, equipped with (among other things) the substitution (=functor)

$$\begin{array}{c} \Gamma \rhd A \\ \downarrow{\scriptstyle \mathsf{p}} \\ \Gamma. \end{array}$$

$\Gamma \rhd A$ is defined as the 'total category' of the displayed category A: its objects are pairs (γ, a) with $\gamma : |\Gamma|$ and $a : \mathsf{Obj}^{\mathsf{D}}\ \gamma$ and morphisms (γ_0, a_0) to (γ_1, a_1) given by a Γ-morphism γ_{01} from γ_0 to γ_1, and a displayed morphism over γ_{01} from a_0 to a_1:

$$(\Gamma \rhd A)\ [(\gamma_0, a_0), (\gamma_1, a_1)] \quad := \quad (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \times A[\gamma_{01} \mid a_0, a_1].$$

The p functor is then the first projection.

The name "concrete CwF" is a nod to the term "concrete category", which is used in category theory to refer to categories whose objects are some set-based structure (e.g. groups) and whose morphisms are the functions on those sets that appropriately preserve the structure. A concrete CwF $\mathscr{C}(\mathfrak{G})$ takes this one step further: not only is the underlying category a concrete category—the objects/contexts are algebras and the morphisms/substitutions are structure-preserving functions—the rest of the CwF structure is similarly "concrete" (types are the appropriate notion of "displayed structure", terms are sections of this displayed structure, etc.).

We began by articulating the "central dogma of category theory": that every notion of "structure" comes equipped with a notion of "structure-preserving function", i.e. constitutes a *category*. The mathematical practice we expound here, *generalized algebra*

can be seen as a refinement, a sharpening of category theory. Accordingly, we have our own **central dogma of generalized algebra**: any notion of "structure" should not only come with a notion of "structure-preserving function", but also notions of "displayed structure" and "section" too. In other words: any notion of structure ought to arrange into a CwF. It's not obvious what kinds of "structure" might satisfy this dictum *besides* $\mathfrak{G}$-algebras,[6] and we don't undertake a characterization of *which* CwFs arise as the concrete CwF of some GAT. But this dogma serves to highlight (one key aspect of) what makes GATs so nice as to deserve their own study.

To conclude, let us note that there are some aspects of these CwFs which KKA introduce, but which won't concern us. Namely, they showed that the CwFs that arise in this way are automatically equipped with extensional identity types, and, moreover have *constant families* (equivalently: are *democratic CwFs*). The latter property means that each *context* Δ can be viewed as a *type* $\mathsf{K}(\Delta) : \mathsf{Ty}\ \Gamma$ (for all Γ) such that substitutions $\mathsf{Sub}\ \Gamma\ \Delta$ are naturally isomorphic to terms $\mathsf{Tm}(\Gamma, \mathsf{K}\ \Delta)$. We won't explore this further—we're more interested in the type- and term-formers which can be modeled in (modifications of) specific concrete CwFs, rather than the constructs available in *every* concrete CwF—but these aspects definitely deserve further consideration in light of the topics covered in subsequent chapters (in particular, the interaction between the constant families/democracy and the polarization structure of the concrete CwFs of categories and preorders studied in chapter 2).

1.4 Fibrancy and the Looking-Glass Question

Once again, we're not content to just fulfill the basic structural components of a CwF: we want to study *interesting* type theories, i.e. ones equipped with various type- and term-forming operations. We noted a moment ago that every concrete CwF interprets extensional identity types, but, as mentioned earlier, extensional equality comes with a significant downside: its typechecking is not decidable. Accordingly, much of the focus of type-theoretic research in recent decades has instead focused on *intensional* type theory. What if we want to model intensional equality in concrete CwFs?

Let's focus on a particular case: the concrete CwF of *setoids*. Setoids are a natural candidate for giving a semantics of intensional equality: a setoid is, after all, a set equipped with a chosen equivalence relation, which is (in general) a coarser notion of 'equality' than metatheoretic (i.e. judgmental, extensional) equality. And indeed it seems like we can get off to a good start, by interpreting the Id type former and its constructor, refl.

Example 1.4.1. The type of **displayed setoids** on a setoid Γ is given by

[6] Perhaps topological spaces? Topologies seem to be impossible to write down exactly in the GAT setting (as their definition inherently involves operations on subsets), but we *can* write down *interior algebras* as a GAT. So maybe the definition of displayed interior algebra, section, etc. can be translated to topologies.

PSEUDOAGDA

```
record 𝔖etoid-DAlg (Γ : 𝔖etoid-Alg) : Set where
  X^D     : |Γ| → Set
  _~^D_   : {x y : |Γ|} → X^D x → {base : x ~_Γ y} → X^D y → Prop
  rfl^D   : {x : |Γ|} → (x^D : X^D x) → x^D ~^D{base=Γ.rfl x} x^D
  sym^D   : {x y : |Γ|}{x^D : X^D x}{y^D : X^D y}{fwd : x ~_Γ y} →
            x^D ~^D{base=fwd} y^D  →  y^D ~^D{base=Γ.sym fwd} x^D
  trns^D  : {x y z : |Γ|}{x^D : X^D x}{y^D : X^D y}{z^D : X^D z}
            {xy : x ~_Γ y}{yz : y ~_Γ z} →
            x^D ~^D{base=xy} y^D  →  y^D ~^D{base=yz} z^D  →
            x^D ~^D{base=Γ.trns xy yz} z^D
```

The type of **sections** of a displayed setoid $(X^{\mathrm{D}}, {\sim}^{\mathrm{D}})$ on a setoid $\Gamma = (|\Gamma|, {\sim}_\Gamma)$ is given by

PSEUDOAGDA

```
record 𝔖etoid-Sect (|Γ|,~_Γ) (X^D,~^D) : Set where
  #1 : (x : |Γ|) → X^D x
  #2 : {x y : |Γ|} → (base : x ~_Γ y) → (#1 x) ~^D (#1 y)
```

Definition 1.4.2. The concrete CwF $\mathscr{C}(\mathfrak{Setoid})$ interprets the Id type-former and refl constructor of Definition 1.2.18:

PSEUDOAGDA

```
Id : {Γ : Con}{A : Ty Γ} → (t t' : Tm(Γ, A)) → Ty Γ
  Id(t,t') : |Γ| → Set
  Id(t,t') x :=  A._~^D_ {base=Γ.rfl x} (t x) (t' x)

  Id(t,t')._~^D_ : Id(t,t') x → {base : x ~_Γ y} → Id(t,t') y → Prop
  x^D ~^D y^D := ⊤

refl : {Γ : Con}{A : Ty Γ} → (t : Tm(Γ, A)) → Tm(Γ, Id(t,t))
  refl_t : (x : |Γ|) → Id(t,t) x
  refl_t x := A.rfl^D t(x)
```

But we run into an issue when trying to interpret the elimination principle for intensional identity types, the *J-rule*. To see the issue, it's simpler to specialize J to the principle of *transport*. Transport allows us to turn elements of $B[\mathrm{id}, t]$ into elements of $B[\mathrm{id}, t']$ by "transporting them along an equality" between t and t'. Intuitively: if t and t' are equal, then any "property" of t (or any "structure" depending on t) can be

transformed into the analogous property/structure over t'. We saw above that we can transport over *extensional* equalities: the ONEGAT language had extensional equality, and we made use of the transport principle there to be able to express heterogeneous equations in GAT signatures. But transport is also a property of intensional equality: using the J-rule, we can prove transport.

Definition 1.4.3. In the initial CwF with intentional identity types, we have an operator

$$\frac{f : \mathsf{Tm}(\Gamma, \mathsf{Id}(t, t')) \quad s : \mathsf{Tm}(\Gamma, B[\mathsf{id}, t])}{\mathrm{tr}_B\ f\ s := (\mathsf{J}\ s)[\mathsf{id}, t', f] : \mathsf{Tm}(\Gamma, B[\mathsf{id}, t'])} \tag{1.4.4}$$

The above proof is done in the initial $\mathfrak{CwF}_{I=}$-algebra, the syntax of type theory with intensional identity types. So, now some metatheoretic reasoning: *if* $\mathscr{C}(\mathfrak{Setoid})$ had an appropriate J-rule (making it a $\mathfrak{CwF}_{I=}$-algebra), then, since we can prove the transport law in the syntax $\mathfrak{CwF}_{I=}$-algebra, $\mathscr{C}(\mathfrak{Setoid})$ would have to validate it too. If $\mathscr{C}(\mathfrak{Setoid})$ doesn't support transport (as we'll show), then this means it cannot be a $\mathfrak{CwF}_{I=}$-algebra.

Let's unpack what it would mean for $\mathscr{C}(\mathfrak{Setoid})$ to validate the transport law. Γ is a setoid, A is a displayed setoid over Γ, and B is a displayed setoid over $\Gamma \triangleright A$. So B consists of a family of sets

$$B : (x : |\Gamma|) \to A(x) \to \mathsf{Set}$$

together with a heterogeneous equivalence relation

$$\begin{aligned}(\_ \sim^{\mathsf{D}} \_) : B(x_0, a_0) \to\ & \\ & \{x_{01} : x_0 \sim_\Gamma x_1\} \to \{a_{01} : a_0 \sim^{\mathsf{D}}_{\{\mathsf{base}=x_{01}\}} a_1\} \to \\ & B(x_1, a_1) \to \mathsf{Prop}.\end{aligned}$$

The terms t, t' send elements x of Γ to elements of $A(x)$, such that $x_{01} : x_0 \sim_\Gamma x_1$ implies $t(x_0) \sim^{\mathsf{D}}_{\{\mathsf{base}=x_{01}\}} t(x_1)$ and $t'(x_0) \sim^{\mathsf{D}}_{\{\mathsf{base}=x_{01}\}} t'(x_1)$; the term s sends x to $s(x) : B(x, t(x))$ and x_{01} to

$$s(x_{01}) : s(x_0) \sim^{\mathsf{D}}_{\{x_{01}=x_{01}\}\{a_{01}=t(x_{01})\}} s(x_1);$$

and the f sends x to $f(x) : t(x) \sim^{\mathsf{D}}_{\{\mathsf{base}=\Gamma.\mathsf{rfl}\ x\}} t'(x)$. Does this give us enough to define a term of type $B[\mathsf{id}, t']$?

No.

Proposition 1.4.5. *The concrete CwF of setoids $\mathscr{C}(\mathfrak{Setoid})$ does not validate the transport law Equation 1.4.4.*

Proof. Consider the counterexample:

- $\Gamma := \{\star\}$, the terminal setoid;
- $A(\star) := \{0, 1\}$, equipped with $a_{01} : 0 \sim^{\mathsf{D}} 1$;
- $B(\star, 0) := \{\dagger\}$ and $B(\star, 1) := \emptyset$;

- $t(\star) := 0$;
- $t'(\star) := 1$;
- $f(\star) := a_{01}$;
- $s(\star) := \dagger$.

If $\mathscr{C}(\mathfrak{Setoid})$ validated the transport law, then from f and s we could obtain

$$\mathsf{tr}\ f\ s : \mathsf{Tm}(\Gamma, B[\mathsf{id}, t']).$$

But this would send $\star$ to an element of $B(\star, t'(\star)) = B(\star, 1) = \emptyset$, a contradiction. □

Consequently, the identity types defined in Definition 1.4.2 are apparently not worth much: they cannot support the J-rule, so it's not clear how to eliminate (that is, *do anything with*) them.

A reader familiar with the semantics of dependent type theory might wonder what the relationship is between our *concrete CwF of setoids* and the *setoid model* of Hofmann and Altenkirch [Hof95a; Alt99], which also has setoids as contexts and setoid morphisms as substitutions. They are not the same, and we've hit upon the key difference: the setoid model *does* model intensional equality, including the J-rule and transport. So what does the setoid model do different semantically that allows it to do what $\mathscr{C}(\mathfrak{Setoid})$ apparently cannot?

The difference is **fibrancy**. If we examine the formulation of the setoid model given in [ABK+21, p. 7], we see that the types of the setoid model are still displayed setoids, but with two extra pieces of data.

Definition 1.4.6. A displayed setoid $(X^{\mathsf{D}}, \sim^{\mathsf{D}})$ over $(X, \sim)$ is called **fibrant** if it comes equipped with the following data.

PSEUDOAGDA

```
coe : {x y : X} → (base : x ~ y) → X^D x → X^D y
coh : {x y : X} → (base : x ~ y) → (x^D : X^D x) → x^D ~^D (coe base x^D)
```

These components encode a kind of *functoriality* of X^{D} over X: if $x \sim y$, then we can convert—or should we say *transport*—elements $x^{\mathsf{D}} : X^{\mathsf{D}}\ x$ over to elements of $X^{\mathsf{D}}\ y$ using coe, and have the result be related to x^{D} in the displayed equivalence relation $\sim^{\mathsf{D}}$. That 'fibrancy' gives rise to transport and corresponds to functoriality is made precise in the following claims.

Definition 1.4.7. If the displayed setoid $B : \mathsf{Ty}(\Gamma \triangleright A)$ is assumed to be *fibrant* in the sense of Definition 1.4.6, then the transport law of Definition 1.4.3 is validated.

PSEUDOAGDA

```
tr : {Γ}{A}{B}{t}{t'} ⟶ Tm(Γ,Id(t,t')) ⟶ Tm(Γ,B[id,t]) ⟶ Tm(Γ,B[id,t'])
 (tr f s) : (x : |Γ|) ⟶ B(x, t' x)
 (tr f s) := coe {x=(x,t(x))} {y=(x,t'(x))} (Γ.rfl x, f x) s(x)

 (tr f s) : (x₀₁ : x₀ ~Γ x₁) ⟶ (tr f s) x₀ ~D_{x01=x01}{a01=f(x01)} (tr f s) x₁
 (tr f s) x₀₁ :=
   trnsD
    (symD (coh (Γ.rfl x₀, f x₀) (x₀,t(x₀))))
    (trnsD (s x₀₁) (coh (Γ.rfl x₁, f x₁) (x₁,t(x₁))))
```

Proposition 1.4.8. *The type of fibrant displayed setoids on a setoid* Γ *is equivalent to the type of functors from* Γ *(regarded as a category with subsingleton hom-sets) into the category* Setoid.

Proof. Given $A\colon \Gamma \Rightarrow \mathrm{Setoid}$, the corresponding family of sets $X^{\mathrm{D}}\colon |\Gamma| \to \mathrm{Set}$ is given by $X^{\mathrm{D}}(x) := |A(x)|$. The heterogeneous equivalence relation (over base$\colon x \sim_\Gamma y$) is given by

$$x^{\mathrm{D}} \sim^{\mathrm{D}} y^{\mathrm{D}} := (A \text{ base } x^{\mathrm{D}}) \sim_{A(y)} y^{\mathrm{D}}.$$

Given a fibrant displayed setoid $(X^{\mathrm{D}}, \sim^{\mathrm{D}}, \mathrm{coe}, \mathrm{coh})$, the setoid $A(x)$ is defined as the set $X^{\mathrm{D}}(x)$ with relation

$$a \sim_{A(x)} a' \quad := a \sim^{\mathrm{D}}_{\{\text{base}=\Gamma.\text{rfl } x\}} a'.$$

Given base$\colon x \sim_\Gamma y$, the setoid morphism $A(x) \to A(y)$ is defined by sending $a\colon |A(x)|$ to coe base $a\colon |A(y)|$. This preserves the equivalence relation: given $p\colon a \sim_{A(x)} a'$, i.e. $p\colon a \sim^{\mathrm{D}}_{\{\text{base}=\Gamma.\text{rfl } x\}} a'$, we have

$$\begin{aligned} \mathrm{sym}^{\mathrm{D}}(\text{coh base } a) &: \text{coe base } a \sim^{\mathrm{D}}_{\{\text{base}=\Gamma.\text{sym base}\}} a \\ p &: a \sim^{\mathrm{D}}_{\{\text{base}=\Gamma.\text{rfl } x\}} a' \\ \text{coh base } a' &: a' \sim^{\mathrm{D}}_{\{\text{base}=\text{base}\}} \text{coe base } a' \end{aligned}$$

So, using $\mathrm{trns}^{\mathrm{D}}$ to concatenate these, we have a relation between coe base a and coe base a' in the setoid $A(y)$, as desired. □

Without fibrancy, we can still view a displayed setoid over Γ as a family of setoids $\{A(\gamma)\}_{\gamma\colon \Gamma}$ as spelled out in this proof; the fibrancy only comes in when we want to obtain a setoid morphism $A(x) \to A(y)$ from $x \sim y$.

For our purposes, the explicitly functorial formulation—that a fibrant displayed setoid over Γ is a functor $\Gamma \Rightarrow \mathrm{Setoid}$—will be more convenient; we just have to translate the notion of 'term' along this equivalence: see Figure 1.10. We therefore have the setoid model.

pseudoAgda

```
record Tm (Γ : Con) (A : Ty Γ) : Set where
  #1 : (x : |Γ|) ⟶ |A x|
  #2 : (x₀₁ : x₀ ~Γ x₁) ⟶ A x₀₁ (#1 x₀) ~A(x₁) (#1 x₁)
```

Figure 1.10: Explicit definition of the terms in the setoid model.

Definition 1.4.9. The **setoid model of type theory**—denoted $\mathscr{C}^{\mathsf{fib}}(\mathfrak{Setoid})$—is the concrete CwF of setoids, $\mathscr{C}(\mathfrak{Setoid})$, but with fibrant types

$$\mathsf{Ty}\ \Gamma := \Gamma \Rightarrow \mathrm{Setoid}.$$

There is evidently a 'forgetful' CwF morphism $\mathscr{C}^{\mathsf{fib}}(\mathfrak{Setoid}) \to \mathscr{C}(\mathfrak{Setoid})$ which forgets the fibrancy data (and which is the identity on contexts, substitutions, and terms). However, this morphism is neither injective (the setoid model is not a sub-CwF of $\mathscr{C}(\mathfrak{Setoid})$: the fibrancy data—coe in particular—is *data*, not a property) nor surjective (for instance, the counterexample displayed setoid B constructed for Proposition 1.4.5 cannot possibly be made fibrant). So $\mathscr{C}^{\mathsf{fib}}(\mathfrak{Setoid})$ is both a restriction and an enrichment of $\mathscr{C}(\mathfrak{Setoid})$.

As mentioned in the introduction, a key motivating example for us will be the groupoid model of Hofmann and Streicher [HS95]. We could repeat the above analysis to show that the concrete CwF of groupoids, $\mathscr{C}(\mathfrak{Grpd})$, doesn't support intensional identity types; after all, setoids can be viewed as groupoids whose hom-sets are propositions, so Proposition 1.4.5 basically needs no modification. As with the setoid model, the groupoid model arises as the fibrant version of the concrete CwF of groupoids, with only slightly more elaborate fibrancy data: we now need to add coherence equations to coe and coh, which were automatic in the setoid case because all elements of a proposition are equal. The addition of fibrancy makes it possible to interpret intensional identity types.

Definition 1.4.10. A displayed groupoid A over Γ is called **fibrant** if it comes equipped with the following data.

pseudoAgda

```
coe  : {γ₀ γ₁ : |Γ|} ⟶ (γ₀₁ : Γ[γ₀,γ₁]) ⟶ A γ₀ ⟶ A γ₁
coh  : {γ₀}{γ₁} ⟶ (γ₀₁ : Γ[γ₀,γ₁]) ⟶ (a₀ : A γ₀) ⟶ A[γ₀₁ | a₀ , coe γ₀₁ a₀]
coe_id : coe id_γ a = a
coe∘ : coe (γ₁₂ ∘ γ₀₁) a₀ = coe γ₁₂ (coe γ₀₁ a₀)
coh_id : coh id_γ a = id^D γ a
coh∘ : coh (γ₁₂ ∘ γ₀₁) a₀ = coh γ₁₂ (coe γ₀₁ a₀) ∘^D coh γ₀₁ a₀
```

pseudoAgda

```
record Tm (Γ : Con) (A : Ty Γ) : Set where
  #1 : (γ : |Γ|) ⟶ |A γ|
  #2 : (γ₀₁ : Γ[ γ₀ , γ₁ ]) ⟶ (A γ₁)[ A γ₀₁ (#1 γ₀), (#1 γ₁) ]
  #3 : #2 id_γ = id
  #4 : #2 (γ₁₂ ∘ γ₀₁) = (#2 γ₁₂) ∘ A γ₁₂ (#2 γ₀₁)
```

Figure 1.11: Explicit definition of the terms in the groupoid model.

Proposition 1.4.11. *The type of fibrant displayed groupoids on a groupoid* Γ *is equivalent to the type of functors from* Γ *into the category* Grpd.

Definition 1.4.12. The **groupoid model of type theory**—denoted $\mathscr{C}^{\mathrm{fib}}(\mathfrak{Grpd})$—is the concrete CwF of setoids, $\mathscr{C}(\mathfrak{Grpd})$, but with fibrant types

$$\mathsf{Ty}\ \Gamma := \Gamma \Rightarrow \mathrm{Grpd}.$$

Definition 1.4.13. The groupoid model semantics of identity types are given in Figure 1.12.

A critical difference between the setoid and groupoid models is their validation (or refutation) of the **uniqueness of identity proofs** principle (UIP). UIP says that witnesses of propositional equality are themselves unique (up to propositional equality): for any two terms $f\ f' : \mathsf{Tm}(\Gamma, \mathsf{Id}(t, t'))$, there must exist some term

$$\mathsf{UIP}(f, f') \quad : \mathsf{Tm}(\Gamma, \mathsf{Id}(f, f')).$$

In other words, UIP asserts that identity is a *proposition*—the identity types do not have their own elaborate internal structure. The setoid model *does* obey UIP: roughly, since identity types are interpreted using hom-sets (as in Figure 1.12) and the hom-sets of setoids are subsingletons, there's not "room" to construct distinct, parallel identity proofs. On the other hand, the groupoid model was devised specifically to refute UIP, to prove that UIP was independent of the laws of Martin-Löf Type Theory. So, in the type theory of the groupoid model, there can be identity types $\mathsf{Id}(t, t')$ between which have multiple terms that are not themselves identical. Note, however, that the groupoid model satisfies "UIP, one level up": the identity types *between terms of identity types* are propositions.

Now combine this with observation with two fundamental proofs conducted in the syntax of type theory with intensional identity types: symmetry and transitivity of identity.

Definition 1.4.14. In the initial CwF with intensional identity types, we have an operator

pseudoAgda

```
Id : {Γ : Con}{A : Ty Γ} → Tm(Γ, A) → Tm(Γ, A) → Ty Γ

  Id(t,t') : |Γ| → Grpd
  | Id(t,t') γ | := (A γ) [ t γ , t' γ ]
  (Id(t,t') γ) [ i , j ] := (i = j)

  Id(t,t') : (γ₀₁ : Γ [ γ₀ , γ₁ ]) → (Id(t,t') γ₀) ⇒ (Id(t,t') γ₁)
  Id(t,t') γ₀₁ f₀ := t'(γ₀₁) ∘ A γ₀₁ f₀ ∘ t(γ₀₁)⁻¹

refl : {Γ : Con}{A : Ty Γ} → (t : Tm(Γ, A)) → Tm(Γ, Id(t,t))
  refl_t : (γ : |Γ|) → |Id(t,t) γ|
  refl_t γ := id_t(γ)

J : (t : Tm(Γ, A))
  → (M : Ty (Γ ▷ A ▷ Id(t[ p ], v₀)))
  → Tm(Γ, M[ id , t , refl_t ])
  → Tm(Γ ▷ A ▷ Id(t[ p ], v₀), M)

  (J_t,M m) : (γ : |Γ|) → (a : |A γ|) → (x : (A γ) [t γ , a]) → | M(γ, a, x) |
  (J_t,M m) γ a x = M (id_γ, x) (m γ)  -- x = x ∘ A id_γ id_tγ ∘ t id_γ

  (J_t,M m) : (γ₀₁ : Γ [γ₀,γ₁])
      → (a₀₁ : Aγ₁ [A γ₀₁ a₀, a₁])
      → M(γ₁, a₁, a₀₁ ∘ A γ₀₁ x₀ ∘ t(γ₀₁)⁻¹ )[
            M(γ₀₁,a₀₁) ((J_t,M m) γ₀ a₀ x₀),
            ((J_t,M m) γ₁ a₁ (a₀₁ ∘ A γ₀₁ x₀ ∘ t(γ₀₁)⁻¹))
        ]
  (J_t,M m) γ₀₁ a₀₁ = M (id_γ1, a₀₁ ∘ A γ₀₁ x₀ ∘ t(γ₀₁)⁻¹) (m γ₀₁)
```

Figure 1.12: Explicit definition of identity types in the groupoid model.

$$\frac{f : \mathsf{Tm}(\Gamma, \mathsf{Id}(t, t'))}{f^{-1} := (\mathsf{J}_{t,\mathsf{Id}(\mathsf{v}_0,t)}\ \mathsf{refl}_t)[\mathsf{id}, t', f] : \mathsf{Tm}(\Gamma, \mathsf{Id}(t', t))} \tag{1.4.15}$$

Definition 1.4.16. In the initial CwF with intensional identity types, we have an operator

$$\frac{f : \mathsf{Tm}(\Gamma, \mathsf{Id}(t, t'))\quad g : \mathsf{Tm}(\Gamma, \mathsf{Id}(t', t''))}{f \cdot g := (\mathsf{J}_{t',\mathsf{Id}(t,\mathsf{v}_0)}\ f)[\mathsf{id}, t'', g] : \mathsf{Tm}(\Gamma, \mathsf{Id}(t, t''))} \tag{1.4.17}$$

Since both the groupoid and setoid models are CwFs with intensional identity types, both of these constructions can be done in either model too. Let's think about the setoid model first. In the setoid model, the type $\mathsf{Id}(t, t')$ is a *proposition*: by UIP, $\mathsf{Id}(t, t')$ can have at most one inhabitant (up to propositional equality). Thus we can think of Id as a propositional *relation* between the terms of a given type: for every pair of terms t, t', obtain a proposition $\mathsf{Id}(t, t')$. Moreover, by the refl_t term, Definition 1.4.14, and Definition 1.4.16, it is an *equivalence relation*, i.e. a setoid. We thus say that the types of the setoid model are **synthetic setoids**: every type A in the setoid model has the structure of a setoid: the elements of the setoid are the terms of A and the $\sim$ relation is the identity relation Id. "Synthetic" here means that we do not have to *prove* that a given type A is a setoid in this sense; if we can write down a type in the type theory of the setoid model, then it automatically comes equipped with Id, which is already proved reflexive, symmetric, and transitive.[7]

The same situation unfolds with the groupoid model. In the groupoid model, identity types $\mathsf{Id}(t, t')$ are not propositions but *sets*: there can be $f\ f' : \mathsf{Tm}(\Gamma, \mathsf{Id}(t, t'))$ where the identity type $\mathsf{Id}(f, f')$ is *empty*, i.e. false; but, if there is an inhabitant $\phi : \mathsf{Tm}(\Gamma, \mathsf{Id}(f, f'))$, then ϕ must be unique (for any other $\phi' : \mathsf{Tm}(\Gamma, \mathsf{Id}(f, f'))$, obtain an element of $\mathsf{Id}(\phi, \phi')$). So Id is not a binary relation on A, a family of propositions indexed over pairs of terms t, t' of type A, but a binary *family of sets*. In view of this, the refl term, Definition 1.4.14, and Definition 1.4.16 say that the types of the groupoid model are **synthetic groupoids**: the terms are the objects and $\mathsf{Id}(t, t')$ is the hom-set of the groupoid. The term refl_t of type $\mathsf{Id}(t, t)$ is the identity morphism, the term $f \cdot g$ defined in Definition 1.4.16 is the composition of f and g, and f^{-1} defined in Definition 1.4.14 is the inverse operation. The category laws (the unit laws and associativity) and the groupoid laws

$$\mathsf{Id}(f \cdot f^{-1}, \mathsf{refl}_t) \qquad \text{and} \qquad \mathsf{Id}(f^{-1} \cdot f, \mathsf{refl}_{t'})$$

can be proved using the J-rule (again we emphasize that we prove this once-and-for-all, so any type we can write down is automatically equipped with the groupoid structure).

So the types of the setoid model are synthetic setoids, and the types of the groupoid model are synthetic groupoids. How curious! In both these cases, the semantic structure (the equivalence relation and the groupoid structure, respectively) gets reflected into the syntax (in the form of the Id-types), such that the type theory provides a synthetic

[7] To define an "analytic" setoid, i.e. a setoid in the usual sense, we must explicitly define its relation and prove reflexivity, symmetry, and transitivity by hand. If we want to define another, we must do all that once again.

theory of the original semantic structure. What isn't clear is whether this is something highly-specific to groupoids and setoids, or if this can be done more generally. We state the following question.[8]

Question (Looking-Glass). How is the structure of a GAT $\mathfrak{G}$ reflected in the type theory of (a fibrant version of) the concrete CwF $\mathscr{C}(\mathfrak{G})$ of (displayed) $\mathfrak{G}$-algebras?

It's not obvious which GATs would support such a thing. Can we model a language of synthetic $\mathfrak{N}$-algebras in the category of $\mathfrak{N}$-algebras? A synthetic theory of monoids in Mon? We don't know. However, we can make some observations. First, it seems that fibrancy plays a significant role; after all, it is *not* in the concrete CwFs $\mathscr{C}(\mathfrak{Setoid})$ and $\mathscr{C}(\mathfrak{Grpd})$ that we were able to obtain these synthetic theories, but in their fibrant versions, the setoid and groupoid models.[9] Accordingly, the answer to the Looking-Glass Question, to the extent that it exists, is likely tied up with the answer to the following question.

Question (Fibrancy). Given a GAT $\mathfrak{G}$, what is the appropriate notion of 'fibrancy' for the types of $\mathscr{C}(\mathfrak{G})$?

For groupoids specifically, we have a very good benchmark for what 'fibrancy' ought to mean: recalling Proposition 1.4.11, we see that a fibrant displayed groupoid over Γ is just a functor from Γ to the groupoid of groupoids. We stated Proposition 1.4.11 referring to Grpd as a *category*, but all the functors $A(\gamma_{01}) : A(\gamma_0) \Rightarrow A(\gamma_1)$ picked out in this way are isomorphisms, so really we could understand this as going into the core groupoid of Grpd. We'll see the same for categories: a fibrant displayed category on a category Γ is just a functor $\Gamma \Rightarrow \mathrm{Cat}$. This is all to say: there's some kind of *cosmological* aspect to the Fibrancy Question: the only GATs $\mathfrak{G}$ for which we can satisfactorily answer the Fibrancy Question are those ones where the category of $\mathfrak{G}$-algebras can itself be understood as a $\mathfrak{G}$-algebra.[10] Perhaps this limits the scope of $\mathfrak{G}$ in the Fibrancy Question to just category-like (or quiver-like) structures; or perhaps something more clever can be done. Further consideration is certainly required.

Moreover, our use of identity types to achieve the synthetic structure seems to be an artifact of the fact that the synthetic structure we're trying to capture involves *binary relations/families of sets*. If, for instance, some version of the concrete CwF of monoids, $\mathscr{C}(\mathfrak{Mon})$, were to provide semantics for a synthetic monoid theory, the appropriate type-theoretic construct wouldn't be identity types. So an aspect of this question is determining what kind of type theory is even capable of writing down a synthetic theory of $\mathfrak{G}$-algebras, something which is obviously quite specific to $\mathfrak{G}$. We frame this

[8] Since there's talk of "reflecting" the semantics into the syntax, we take some literary-poetic license and call this the "looking glass question."

[9] Perhaps a synthetic theory of setoids (resp. groupoids) *can* be modeled in the raw concrete CwF of setoids (resp. groupoids), where we put in the witnesses of symmetry and transitivity (and associativity, unit, etc.) by hand instead of proven in the syntax. But this seems much less convenient than the fibrant versions, where we are able to reason with the powerful J-rule.

[10] Setoids, and later preorders, won't exactly fit this mold: in saying "a fibrant setoid over Γ is just a functor $\Gamma \Rightarrow \mathrm{Setoid}$," we're not treating the category Setoid as a setoid, but rather treating the setoid Γ as a category.

issue as the following question.

Question (Synthesis). Given a GAT $\mathfrak{G}$ with fibrant-concrete CwF $\mathscr{C}^{\mathrm{fib}}(\mathfrak{G})$, what extension $\mathfrak{G}\mathfrak{Cw}\mathfrak{F}$ of $\mathfrak{Cw}\mathfrak{F}$ is there such that (a) $\mathscr{C}^{\mathrm{fib}}(\mathfrak{G})\colon \mathfrak{G}\mathfrak{Cw}\mathfrak{F}$-Alg; and (b) the syntax model $\mathbb{G}\mathbb{C}\mathbb{w}\mathbb{F}\colon \mathfrak{G}\mathfrak{Cw}\mathfrak{F}$-Alg provides a synthetic theory of $\mathfrak{G}$-algebras?

The logical extreme of these considerations—which we don't pursue here—is a synthetic theory of CwFs modeled in the CwF of CwFs. The GAT $\mathfrak{Cw}\mathfrak{F}$ seems to clear the Fibrancy Question comfortably: like categories and groupoids, CwFs have the appropriate 'cosmological' character, namely that the category of CwFs has the structure of a CwF, given by the concrete CwF of CwFs, $\mathscr{C}(\mathfrak{Cw}\mathfrak{F})$. So, for a given CwF Γ, it makes sense to define "fibrant displayed CwFs" as those which organize into a CwF-morphism $\Gamma \to \mathscr{C}(\mathfrak{Cw}\mathfrak{F})$; thus we have arrived at $\mathscr{C}^{\mathrm{fib}}(\mathfrak{Cw}\mathfrak{F})$. What remains to figure out, then, is what type- and term-formers can we add to the syntax of type theory which pick out the sense in which the types of $\mathscr{C}^{\mathrm{fib}}(\mathfrak{Cw}\mathfrak{F})$ are synthetic CwFs. We leave this investigation for whichever brave soul has the patience to carry it out.

The purpose of the present work is to answer the Synthesis Question for $\mathfrak{Cat}$, the GAT of categories. The appropriate notion of 'fibrant displayed category' is well-known (Grothendieck opfibrations), and indeed the category model, $\mathscr{C}^{\mathrm{fib}}(\mathfrak{Cat})$, has already appeared in [Nor19].[11] The main part of our task, then, will be to decide which category-model constructions ought to be abstracted and added to the syntax of type theory, in order to arrive at a type theory whose types are *synthetic categories*.

[11] Though not referred to there as 'the category model'.

Chapter 2

The Polarity Calculus

> "Now, if you'll only attend, Kitty, and not talk so much, I'll tell you all my ideas about Looking-glass House. First, there's the room you can see through the glass—that's just the same as our drawing room, only the things go the other way … the books are something like our books, only the words go the wrong way … Oh, Kitty! how nice it would be if we could only get through into Looking-glass House! I'm sure it's got, oh! such beautiful things in it! Let's pretend there's a way of getting through into it, somehow, Kitty."
>
> Lewis Carroll, *Through the Looking-Glass*

> "What indeed can be more similar to, and in all parts more equal to, my hand or my ear than its image in the mirror? And yet I cannot put such a hand as is seen in the mirror in the place of its original; for if the one was a right hand, then the other in the mirror is a left, and the image of the right ear is a left one, which can never take the place of the former."
>
> Immanuel Kant, *Prolegomena to Any Future Metaphysics*

Category theory has been concerned with an abstract mathematical treatment of *duality*, quite literally from its very first sentence [EM45, p. 231]. No wonder, then, that our investigation into category theory's self-reflection in $\mathscr{C}(\mathfrak{Cat})$ would at some point require a treatment of *opposite categories*. What is interesting, however, is the flipped order of logical priority: in standard (analytic) treatments of category theory, we must (of course) first define "category" before speaking of the "opposite category" operation. In the present theory, however, it is otherwise: we must first explore the duality structure of our types *before* we can investigate them as synthetic categories. In other words, an understanding of the type theory of the category model as a **polarized type theory** is prerequisite to a study of it as a *directed* type theory.

There is a semantic reason for this, as well as a syntactic one. The semantic reason is that the interpretation of *identity* types in the groupoid model makes indispensable use of *symmetry*: given two terms t and t' of the same type A, the semantics of the identity type $\mathsf{Id}(t, t')$ makes essential use of the fact that A is a functor valued in *groupoids* (see

Definition 1.4.13). If we instead wish to work in the category model—whose types A are functors valued in *categories*—and interpret the directed equality type $\mathsf{Hom}(t, t')$, then, it turns out, we need the domain term t to be *negative*, i.e. a term of the *opposite type* A^-. Thus we are required to have a theory of type negation ready-to-hand before we can even *form* hom-types, let alone work with them.

Even if we were able to semantically validate an *unpolarized* version of Hom-formation (or if we didn't care about semantics), the modal typing discipline of *type polarities* plays a critical role in regulating directed type theory. This is because our chief tool for reasoning with directed equality types will be the *directed J-rule*, also known as *directed path induction*. In *undirected type theory* (such as the type theory of the groupoid model), we can use the J-rule to easily prove the symmetry of identity (Definition 1.4.14)—from a term of type $\mathsf{Id}(t, t')$, obtain a term of type $\mathsf{Id}(t', t)$—hence types are synthetic *groupoids*. We don't want our directed J-rule to be capable of this, lest our directed type theory "collapse" into undirected type theory. It's imperative, then, that some syntactic *guard* be put in place to prevent the J-rule from proving symmetry. The requirement that t be negative and t' positive in order for $\mathsf{Hom}(t, t')$ to be well-formed achieves this: given f of type $\mathsf{Hom}(t, t')$, we certainly cannot construct f^{-1} of type $\mathsf{Hom}(t', t)$—the latter type is not well-formed! We will have to weaken this system of polarities somewhat to make it useful (e.g. to type the reflexivity term), but the category model will allow us to ensure symmetry is independent of our directed type theory, and thus that it is indeed a *directed* type theory.

The preorder and category models of type theory will, like the setoid and groupoid models, be defined as strictly fibrant concrete CwFs. This means that phenomena of interest—including polarity—will manifest both "deeply" (at the level of contexts and substitutions) and "shallowly" (at the level of types and terms). So not only will our notion of "**polarized CwF**" provide for the negation of *types*, but of *contexts* and *substitutions* as well. The "deeply-polarized" portion will make it possible to define **polarized function types**, terms of which will serve as the *functors* in our eventual synthetic category theory. section 2.3 will detail the modifications to the polarity calculus—namely the introduction of **neutral** contexts and types—which is needed to make these function types viable.

2.1 Polarity Structure of the Preorder and Category Models

Let's begin with the preorder model. As with the setoid model, we will obtain the preorder model $\mathscr{C}^{\mathsf{fib}}(\mathfrak{PreOrd})$ as a restriction of the concrete CwF of preorders and displayed preorders. Once again, the restriction is just in the types: in order to obtain a model with a well-behaved notion of "transport", it is necessary to restrict the types to "strictly fibrant". This can be done by attaching additional components to the definition of 'displayed preorder'—which we spell out in Figure 2.1—but for the present chapter it will be more convenient to use the equivalent definition: *preorder-valued functors*.

pseudoAgda

```
coe  : {x y : X} → X^D x → {base : x ≤ y} → X^D y
coh  : {x y : X}{base : x ≤ y} → (x^D : X^D x) → x^D ≤^D{base=base} (coe x^D)
cart : {x y z : X}{base : x ≤ y}{extn : y ≤ z} → (x^D : X^D x) → (z^D : X^D z) →
       {target : x^D ≤^D{base=trns base extn} z^D} →
       (coe {base=base} x^D) ≤^D{base=extn} z^D
```

Figure 2.1: Additional 'fibrancy data' on a displayed preorder (compare with Definition 1.4.6)

Definition 2.1.1. The **preorder model of type theory**—denoted $\mathscr{C}^{\mathrm{fib}}(\mathfrak{PreOrd})$—is the concrete CwF of preorders $\mathscr{C}(\mathfrak{PreOrd})$, restricted to strictly fibrant types

$$\mathsf{Ty}\ \Gamma := \Gamma \Rightarrow \mathrm{PreOrd}.$$

Saying $A\colon \Gamma \Rightarrow \mathrm{PreOrd}$ for some preorder $\Gamma = (X, \leq)$ means that $A(x)$ is a preorder for each element $x\colon X$, and that $A(xy)$ is a monotone map $A(x)$ to $A(y)$ for every $xy\colon x \leq y$. As before, we can view every such functor as a displayed preorder by way of (a version of) the Grothendieck construction:

$$x^{\mathrm{D}} \leq^{\mathrm{D}}_{\{\mathrm{base}=xy\}} y^{\mathrm{D}} \quad := \quad A\ xy\ x^{\mathrm{D}} \leq y^{\mathrm{D}}$$

where the relation on the right-hand side is that of the preorder $A(y)$. Thus it makes sense to speak of "sections" of such a functor A, the terms of type A in the preorder model.

Definition 2.1.2. Given a preorder Γ and a functor $A\colon \Gamma \Rightarrow \mathrm{PreOrd}$ (i.e. $A\colon \mathsf{Ty}\ \Gamma$ in the preorder model), a **section** t of A consists of two components

$$t_{\#1}\colon (x\colon \Gamma) \to A_{\#1}(x)$$
$$t_{\#2}\colon (xy\colon x \leq y) \to A\ xy\ (t_{\#1}\ x) \leq t_{\#1}(y).$$

The intriguing thing about strictly fibrant concrete CwFs (such as the preorder model) is that *operations on contexts become operations on types*: since types in Γ are functors $\Gamma \Rightarrow \mathrm{PreOrd}$, any endofunctor $F : \mathrm{PreOrd} \Rightarrow \mathrm{PreOrd}$ induces an action on types, defined by post-composition by F. The endofunctor we're interested in here is the *opposite* operation.

Definition 2.1.3. The **opposite** of a preorder $\Gamma = (X, \leq)$ is the preorder $\Gamma^{\mathrm{op}} := (X, \geq)$, where

$$x \geq y \quad := \quad y \leq x.$$

Given a preorder morphism $f\colon \Delta \to \Gamma$, write f^{op} for the same underlying function f, but regarded as a preorder morphism $\Delta^{\mathrm{op}} \to \Gamma^{\mathrm{op}}$. This makes $(\_)^{\mathrm{op}}$ into an endofunctor on PreOrd.

Definition 2.1.4. The object- and morphism-parts of the endofunctor $(\_)^{\mathrm{op}}$ on PreOrd give semantics for the operations in the type theory of the preorder model

$$\frac{\Gamma : \mathsf{Con}}{\Gamma^- : \mathsf{Con}} \qquad \frac{\sigma : \mathsf{Sub}\ \Delta\ \Gamma}{\sigma : \mathsf{Sub}\ (\Delta^-)\ (\Gamma^-)}$$

respectively.

It also gives semantics for an operation on types:

$$\frac{A : \mathsf{Ty}\ \Gamma}{A^- : \mathsf{Ty}\ \Gamma}$$

defined by post-composition with $(\_)^{\mathrm{op}}$:

$$\Gamma \xrightarrow{A} \mathrm{PreOrd} \xrightarrow{(\_)^{\mathrm{op}}} \mathrm{PreOrd}. \qquad (2.1.5)$$

In summary, the $(\_)^{\mathrm{op}}$ endofunctor induces three 'negation' operations: one on contexts, one on substitutions, and one on types. Let us make some observations about these operations. First, notice that there is *not* a negation operation on *terms*. There's no obvious place to insert the $(\_)^{\mathrm{op}}$ endofunctor into Definition 2.1.2 to get it to act on terms. Indeed, we can easily come up with examples of types A which are *inhabited* (i.e. A admits a term) but whose opposite A^- is *uninhabited*—preorders can be quite different than their opposites! This disconnect between the terms of A and A^- will ultimately pose issues for us—especially if we want types to be synthetic categories and A^- to represent the opposite category of A—but we'll address that point later.

Secondly, let us emphasize that the type-negation operation happens *within* a given context: if A is a type in context Γ, then A^- is a type in that same context Γ, *not* its opposite, Γ^-. Fundamentally, this boils down to the fact that $(\_)^{\mathrm{op}}$ is covariant (for $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$, its negation σ^- is in $\mathsf{Sub}\ \Delta^-\ \Gamma^-$, not $\mathsf{Sub}\ \Gamma^-\ \Delta^-$), so the post-composition depicted in Equation 2.1.5 doesn't involve negating Γ at all. As a consequence, the $(\_)^{\mathrm{op}}$ endofunctor on contexts is *not* a CwF-endomorphism on the preorder model.[1]

Next, we note the following properties.

Proposition 2.1.6. *The type-negation operation is a* ***natural transformation*** *from* Ty *to itself: for all contexts* Δ, Γ*, every type* $A : \mathsf{Ty}\ \Gamma$*, and every* $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$*,*

$$A[\sigma]^- \quad = \quad (A^-)[\sigma]. \qquad (2.1.7)$$

Proposition 2.1.8. *The negation operations are all* ***involutions****.*

$$\frac{\Gamma : \mathsf{Con}}{(\Gamma^-)^- = \Gamma} \qquad \frac{\sigma : \mathsf{Sub}\ \Delta\ \Gamma}{(\sigma^-)^- = \sigma} \qquad \frac{A : \mathsf{Ty}\ \Gamma}{(A^-)^- = A} \qquad (2.1.9)$$

With this, we can begin to speak of the negation operations on the preorder

[1] In the next section, we *will* find a way to understand negation as a CwF (iso)morphism, but between the CwF structure and an auxiliary CwF we'll define.

pseudoAgda

```
coe  : {γ₀ γ₁ : |Γ|} → (γ₀₁ : Γ[γ₀,γ₁]) → A γ₀ → A γ₁
coh  : {γ₀}{γ₁} → (γ₀₁ : Γ[γ₀,γ₁]) → (a₀ : A γ₀) → A[ γ₀₁ | a₀ , coe(γ₀₁,a₀) ]
cart  : {γ₀}{γ₁}{γ₂}{a₀ : A γ₀}{a₂ : A γ₂} →
      (γ₀₁ : Γ[γ₀,γ₁]) → (γ₁₂ : Γ[γ₁,γ₂]) → (a₀₂ : A[ γ₁₂ ∘ γ₀₁ | a₀ , a₂ ]) →
      A[ γ₁₂ | coe(γ₀₁,a₀) , a₂ ]
coe_id : coe id_γ a = a
coe∘ : coe (γ₁₂ ∘ γ₀₁) a₀ = coe γ₁₂ (coe γ₀₁ a₀)
coh_id : coh id_γ a = id^D γ a
coh∘ : coh (γ₁₂ ∘ γ₀₁) a₀ = coh γ₁₂ (coe γ₀₁ a₀) ∘^D coh γ₀₁ a₀
cartβ  : {γ₀}{γ₁}{γ₂}{a₀}{a₂}{γ₀₁}{γ₁₂} → (a₀₂ : A[ γ₁₂ ∘ γ₀₁ | a₀ , a₂ ]) →
      cart(γ₀₁,γ₁₂,a₀₂) ∘^D coh(a₀,γ₀₁) = a₀₂
cartη  : {γ₀}{γ₁}{γ₂}{a₀}{a₂}{γ₀₁}{γ₁₂} → (a₁₂ :  A[ γ₁₂ | coe(a₀,γ₀₁) , a₂ ]) →
      cart(γ₀₁, γ₁₂, a₁₂ ∘^D coh(a₀,γ₀₁)) = a₁₂
```

Figure 2.2: Additional 'fibrancy data' on a displayed category (compare with Definition 1.4.10)

model as a **polarity calculus**. As we'll see more precisely in the next section, a CwF equipped with involutive negation operations arranges into a type-theoretic 'dipole': two equivalent-and-opposite type theories—one "positive" and the other "negative"—intimately bound together. This is the essence of polarized type theory as practiced here.

Though we will occasionally refer to the preorder model as an example,[2] ultimately our primary example is instead the *category model*, which is polarized in exactly the same way.

Definition 2.1.10 (Category Model)**.** The **category model of type theory**—denoted $\mathscr{C}^{\mathrm{fib}}(\mathfrak{Cat})$—is the concrete CwF of categories $\mathscr{C}(\mathfrak{Cat})$, restricted to strictly fibrant types

$$\mathrm{Ty}\ \Gamma := \Gamma \Rightarrow \mathrm{Cat}.$$

Definition 2.1.11. The **opposite** of a category Γ is the category Γ^{op}, whose objects are the same but with morphisms 'turned around'

$$\Gamma^{\mathrm{op}}\ [\gamma_0, \gamma_1] \quad := \quad \Gamma\ [\gamma_1, \gamma_0].$$

The identity morphisms and composition in Γ^{op} are inherited from Γ.

$(\_)^{\mathrm{op}}$ extends into an endofunctor on Cat: any functor $\sigma\colon \Delta \Rightarrow \Gamma$ can be viewed

[2] And it deserves more extensive treatment, analogous to contemporary study of the setoid model, e.g. [ABK+21].

as a functor $\sigma^{\mathrm{op}} : \Delta^{\mathrm{op}} \Rightarrow \Gamma^{\mathrm{op}}$:

$$\frac{\dfrac{\dfrac{\delta_{10} : \Delta^{\mathrm{op}}\ [\delta_0, \delta_1]}{\delta_{10} : \Delta\ [\delta_1, \delta_0]}}{\sigma(\delta_{10}) : \Gamma\ [\sigma(\delta_1), \sigma(\delta_0)]}}{\sigma(\delta_{10}) : \Gamma^{\mathrm{op}}\ [\sigma(\delta_0), \sigma(\delta_1)].}$$

Definition 2.1.12. The object- and morphism-parts of the endofunctor $(\_)^{\mathrm{op}}$ on Cat give semantics for the operations in the type theory of the category model

$$\frac{\Gamma : \mathsf{Con}}{\Gamma^- : \mathsf{Con}} \qquad \frac{\sigma : \mathsf{Sub}\ \Delta\ \Gamma}{\sigma : \mathsf{Sub}\ (\Delta^-)\ (\Gamma^-)}$$

respectively.

It also gives semantics for an operation on types:

$$\frac{A : \mathsf{Ty}\ \Gamma}{A^- : \mathsf{Ty}\ \Gamma}$$

defined by post-composition with $(\_)^{\mathrm{op}} : \mathrm{Cat} \Rightarrow \mathrm{Cat}$.

Functors $\Gamma \Rightarrow \mathrm{Cat}$ correspond to *split opfibrations*, displayed categories equipped with the appropriate 'fibrancy data' (Figure 2.2). The fibrancy data required is a combination of the fibrancy data needed for the groupoid model and for the preorder model—the category model is proof-relevant like the former, and asymmetric like the latter.

It will be helpful to spell out explicitly the terms of the category model, and investigate how they interact with the negation operations. As with the setoid, groupoid, and preorder models, the terms of type A in the category model are just the sections of A, viewed as a displayed category over its context.

Definition 2.1.13. Given a category Γ and a functor $A : \Gamma \Rightarrow \mathrm{Cat}$ (i.e. $A : \mathsf{Ty}\ \Gamma$ in the category model), a **section** t of A consists of two components

$$t_{\#1} : (\gamma : \Gamma) \to |A\ \gamma|$$
$$t_{\#2} : (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \to (A\ \gamma_1)\ [A\ \gamma_{01}\ (t_{\#1}\ \gamma_0),\ t_{\#1}(\gamma_1)]$$

such that

$$t_{\#2}\ \mathsf{id}_\gamma = \mathsf{id}_{t_{\#1}(\gamma)} \tag{2.1.14}$$
$$t_{\#2}(\gamma_{12} \circ \gamma_{01}) = (A\ \gamma_{01}\ (t_{\#2}\ \gamma_{12})) \circ t_{\#2}(\gamma_{01}). \tag{2.1.15}$$

So let's consider: what's the difference between

- a term of type A (where $A : \mathsf{Ty}\ \Gamma$),
- a term of type A^- (where $A : \mathsf{Ty}\ \Gamma$), and
- a term of type A (where $A : \mathsf{Ty}\ \Gamma^-$)?

As we'll repeatedly find, polarity in the category model always manifests in the *morphism part* of types and terms—it is only in the morphisms, after all, that a category

differs from its opposite. Accordingly, types in Γ and Γ^- have the same kind of object part—an operation sending objects γ to categories $A(\gamma)$—and likewise for terms: regardless of whether $t : \mathsf{Tm}(\Gamma, A)$, $t : \mathsf{Tm}(\Gamma, A^-)$, or $t : \mathsf{Tm}(\Gamma^-, A)$, its object part sends γ to an object of $A(\gamma)$. However, the morphism parts are quite different: for $\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]$, the three different morphism parts are as follows.

pseudoAgda

```
-- t : Tm(Γ, A) where A : Ty Γ
t γ₀₁ : (A γ₁) [ A γ₀₁ (t γ₀), t γ₁ ]
-- t : Tm(Γ, A⁻) where A : Ty Γ
t γ₀₁ : (A γ₁) [ t γ₁, A γ₀₁ (t γ₀) ]
-- t : Tm(Γ⁻, A) where A : Ty Γ⁻
t γ₀₁ : (A γ₀) [ A γ₀₁ (t γ₁), t γ₀ ]
```

So the difference between terms of A and A^- is relatively shallow: both morphism parts send γ_{01} to a morphism in $A(\gamma_1)$ between $A\ \gamma_{01}\ (t\ \gamma_0)$ and $t(\gamma_1)$, but in different directions. Negating the context, however, is a deeper difference: note that the last of these three yields a morphism in $A(\gamma_0)$—since A is now contravariant, the functor $A(\gamma_{01})$ sends objects of $A(\gamma_1)$ to objects of $A(\gamma_0)$ rather than the other way around, so we can only form a morphism between $t(\gamma_0)$ and $t(\gamma_1)$ by transporting the *latter* over to the category where the *former* resides, $A(\gamma_0)$. Of course, we could employ both kinds of negation, and consider terms $t : \mathsf{Tm}(\Gamma^-, A^-)$ for some $A : \mathsf{Ty}\ \Gamma^-$. Then $t(\gamma_{01})$ would be of shape $(A\ \gamma_0)\ [t(\gamma_0), A\ \gamma_{01}\ (t\ \gamma_1)]$. This form matches that of the Grothendieck construction for *contra*variant (pseudo)functors, the same way the morphism part for $\mathsf{Tm}(\Gamma, A)$ matches the Grothendieck construction for *co*variant (pseudo)functors; no wonder, then, that $\mathsf{Tm}((\_)^-, (\_)^-)$ will serve as the negative counterpart to Tm in the constructions of the next section.

Henceforth, we'll use the terms "shallow" and "deep" to name the two levels of *reversal* demonstrated in the previous paragraph: the category and preorder models are **deeply-polarized** because they're equipped with negation operators on contexts and substitutions, and **shallowly-polarized** because they admit a type-negation operation. It's worth noting that these notions are separable: we could have deep polarity without shallow, or *vice versa*. For instance, note that the raw concrete CwFs $\mathscr{C}(\mathfrak{PreOrd})$ and $\mathscr{C}(\mathfrak{Cat})$ still have deep polarity—we still have the endofunctor on algebras—but not shallow: the type-negation operation is only definable in the preorder and category models because of their fibrancy requirement, so we can't define type-negation on these models. On the other hand, we could follow North [Nor19] and only study the 'shallow' polarities, not giving ourselves syntax for negating a context or substitution. But for our purposes, it is most natural (and necessary) to include both kinds of polarity: as discussed above, the shallow polarity is needed to properly regulate directed path induction, whereas the deep polarity is needed to make dependent functions work in polarized models (as we'll see in the next section).

We now turn our attention to the task of abstracting the notion 'polarization structure' exhibited by the preorder and category models into a general-purpose mathematical concept: *polarized CwFs (PCwFs)*. We do so for two reasons. Of course, abstraction

is generally good mathematical hygiene—freeing one's constructions from the peculiarities of a particular example tends to increase their modularity, applicability, and overall elegance. However, this isn't too big a motivation for us: unlike, say, the concept of a 'group',[3] we don't anticipate there being a incredibly diverse array of instances of 'PCwFs'—for the purposes of the present work, "PCwF" will be little more than shorthand for "the preorder or category model". Our real motivation for introducing 'PCwF' as an abstract concept is specific to the practice of generalized algebra: *if it can be expressed as a GAT, then it admits an initial algebra.* That is, since our concept of a 'PCwF' will be given as a GAT (namely an extension of $\mathfrak{CwF}$), it will admit a syntax model. We actually won't be too interested in the syntax of polarized type theory itself, but this will be the first step towards our ultimate goal: the GAT of *directed CwFs* and its syntax model, *directed type theory*.

With all this in mind, we state our definition of *polarized CwF*.

Definition 2.1.16. A **polarized category with families (PCwF)** consists of a CwF $(\mathsf{Con}, \mathsf{Sub}, \bullet, \mathsf{Ty}, \mathsf{Tm}, \triangleright^+, \ldots)$[a] equipped with the following operations.

- An endofunctor $(\_)^- : \mathsf{Con} \Rightarrow \mathsf{Con}$ such that $(\Gamma^-)^- = \Gamma$ and $(\sigma^-)^- = \sigma$ for all Γ and σ, and such that $\bullet^- = \bullet$.
- A natural transformation $(\_)^- : \mathsf{Ty} \to \mathsf{Ty}$ such that $(A^-)^- = A$ for all A.

We'll refer to this extra structure as a **polarization** on the underlying CwF.

These are the algebras for the GAT $\mathfrak{PCwF}$.

[a] Note that we start to denote the context extension operator as $\triangleright^+$ instead of just $\triangleright$. This is in anticipation of the constructions of the next section.

On first glance, this definition may seem to be missing something crucial. In the preorder and category models, the type-negation operation arose by the fiberwise application of the context-negation endofunctor, so there was some fundamental connection between the deep and shallow polarities. But we cannot say "type negation is fiberwise context-negation" with regards to an arbitrary CwF: performing fiberwise application requires that our CwF is not only a concrete CwF, but one whose types are appropriately fibrant. So there's nothing in the definition of PCwF which requires these to be related.

However, we embrace this as a feature: we wish to ensure that the deep and shallow polarities are appropriately independent, e.g. that one can employ type-negation without involving context-negation. This is in contrast to the theory of Licata and Harper [LH11], which permits context- and substitution-negation operations, but only a *negative context extension* operation, not type-negation. So, in that theory, type-negation, to the extent it's possible, remains inextricably bound up in the deep polarity calculus. As we introduce negative context extension in the next section, we'll see that it incorporates type-negation, meaning that the Licata-Harper theory involves type-negation without making it accessible in the syntax. Though this doesn't pose a problem within the context of that theory, within our approach it would force all the *synthetic functors* in our synthetic category theory to be contravariant—quite an undesirable outcome. Thus we'll uphold the independence of shallow polarity, and not seek to require that it arises from the deep polarity endofunctor.

[3] Or even 'CwF'.

Lean–nouGAT

```
def 𝔓ℭ𝔴𝔉 : GAT := {|
  include ℭ𝔴𝔉;

    neg_Con    : Con ⇒ Con,
    neg_Sub    : {Δ Γ : Con} ⇒ Sub Δ Γ ⇒
        Sub (neg_Con Δ) (neg_Con Γ),
    neg_Ty     : {Γ : Con} ⇒ Ty Γ ⇒ Ty Γ,
    neg_empty  : neg_Con empty ≡ empty,
    neg_id     : {Γ : Con} ⇒
        neg_Sub (id Γ) ≡ id (neg_Con Γ),
    neg_comp   : {Θ Δ Γ : Con} ⇒
        (δ : Sub Θ Δ) ⇒ (γ : Sub Δ Γ) ⇒
        neg_Sub (comp γ δ)
        ≡ comp (neg_Sub γ) (neg_Sub δ),
    neg_nat    : {Δ Γ : Con} ⇒
        (γ : Sub Δ Γ) ⇒ (A : Ty Γ) ⇒
        neg_Ty (substTy γ A) ≡ substTy γ (neg_Ty A),
    invl_Con   : (Γ : Con) ⇒ neg_Con(neg_Con Γ) ≡ Γ,
    invl_Sub   : {Δ Γ : Con} ⇒ (γ : Sub Δ Γ) ⇒
        neg_Sub(neg_Sub γ)
            #⟨ invl_Con Δ ⟩
            #⟨ invl_Con Γ ⟩
        ≡ γ,
    invl_Ty    : {Γ : Con} ⇒ (A : Ty Γ) ⇒
        neg_Ty (neg_Ty A) ≡ A
|}
```

Figure 2.3: The GAT $\mathfrak{PCwF}$ of Polarized CwFs.

2.2 Theory of Polarized CwFs

In this section, our task will be to further develop the theory of PCwFs, and thereby get a better understanding of what they are and how to operate with them. Since PCwFs are a category-theoretic structure ($\mathfrak{PCwF}$ extends $\mathfrak{Cat}$, so every PCwF is based on a category), but the totality of PCwFs also organize into a category, we can study PCwFs at two levels:[4] we can explore the *category* of PCwFs—universal properties stated with PCwFs, the relationship between CwF and PCwF, etc.—but we can also consider the structure of a given PCwF—how its type theory behaves, different ways of arranging the data, etc. We consider both viewpoints here, starting with the former.

An appropriate way to think of a PCwF is as a CwF, topped with a 'polarization'—the endofunctor on contexts and endo-natural transformation on Ty making the underlying CwF into a PCwF. Taking this view, we might consider situations such as the following: given a sub-CwF $\mathcal{D} \hookrightarrow \mathcal{C}$ where $\mathcal{C}$ comes equipped with a polarization, does the polarization transfer (i.e. *restrict*) to the sub-CwF $\mathcal{D}$? We obtain the following fact.

Proposition 2.2.1. *Suppose $\mathcal{D}$ is a sub-**CwF** $\mathcal{D} \hookrightarrow \mathcal{C}$ of a **PCwF** $\mathcal{C}$, which is closed under the negation operation $(\_)^-$ of $\mathcal{C}$:*

- *for every $\Gamma : \mathsf{Con}_{\mathcal{D}}$, its negation $\Gamma^- : \mathsf{Con}_{\mathcal{C}}$ is also in $\mathsf{Con}_{\mathcal{D}}$,*
- *for every $\sigma : \mathsf{Sub}_{\mathcal{D}}\ \Delta\ \Gamma$, its negation $\sigma^- : \mathsf{Sub}_{\mathcal{C}}\ \Delta^-\ \Gamma^-$ is also in $\mathsf{Sub}_{\mathcal{D}}\ \Delta^-\ \Gamma^-$,*
- *for every $A : \mathsf{Ty}_{\mathcal{D}}\ \Gamma$, its negation $A^- : \mathsf{Ty}_{\mathcal{C}}\ \Gamma$ is also in $\mathsf{Ty}_{\mathcal{D}}\ \Gamma$;*

*then $\mathcal{D}$ is equipped with a polarization, the **restriction** of $(\_)^-$ to $\mathcal{D}$.*

Of course, the PCwF structure $\mathcal{D}$ produced here is then a sub-*PCwF* of $\mathcal{C}$. There's more that could be said on this matter—for instance, if any of these closure conditions did not hold, then one could consider the *least* sub-PCwF of $\mathcal{C}$ containing $\mathcal{D}$, and so on. But this statement of Proposition 2.2.1 is adequate for our purpose here, specifically our canonical examples.

Example 2.2.2. The groupoid model inherits the polarization of the category model by Proposition 2.2.1: the opposite of every groupoid is a groupoid and the fiberwise opposite of a family of groupoids is still a family of groupoids, so we can restrict the category model's polarization to the groupoid model. Likewise, the polarization of the preorder model restricts to the setoid model.

It's a bit funny to consider the opposite category of a groupoid, because groupoids are inherently *self-dual*: for any groupoid Γ, the functor $\Gamma \to \Gamma^{\mathrm{op}}$ which is the identity on objects and sends $\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]$ to its inverse $\gamma_{01}^{-1} : \Gamma^{\mathrm{op}}\ [\gamma_0, \gamma_1]$ is evidently self-inverse, and in particular an isomorphism.[5] So, unlike the context-negation operation on the category model—where Γ and Γ^- can be meaningfully different—the polarization becomes essentially trivial when restricted to the groupoid model. Simply for the sake of building up the surrounding theory of PCwFs,[6] we make the phrase "essentially

[4] "Macroscopic" and "microscopic", if you will.

[5] Remember this isomorphism for the next section!

[6] "We must also remember that a reserve of knowledge is always an advantage, and that the most practical of mathematicians may be seriously handicapped if his knowledge is the bare minimum which

trivial" precise with the following definition.

Definition 2.2.3. The functor $\nabla : \mathrm{CwF} \to \mathrm{PCwF}$ equips a CwF with the *trivial polarization*, where all the negation operations are the identity:

$$\Gamma^- := \Gamma \qquad \sigma^- := \sigma \qquad A^- := A.$$

Remark 2.2.4. The polarizations on the setoid and groupoid models obtained by restricting the preorder and category polarizations (Example 2.2.2) are equivalent to the trivial ones: for any groupoid Γ, there is an isomorphism between Γ itself (i.e. Γ^- in $\nabla(\mathscr{C}^{\mathrm{fib}}(\mathfrak{Grpd}))$) and its opposite Γ^{op} (i.e. Γ^- using the polarization of $\mathscr{C}^{\mathrm{fib}}(\mathfrak{Cat})$).

The fact that $\Gamma \cong \Gamma^-$ when Γ is a groupoid will be of central focus in section 2.3.

Let's establish one other fact about the relationship between CwF and PCwF. As GATs, $\mathfrak{PCwF}$ is an extension of $\mathfrak{CwF}$: we obtain the signature $\mathfrak{PCwF}$ in the onegat language by extending $\mathfrak{CwF}$ along a successive series of onegat types-in-context.[7] This means we automatically get a **forgetful functor** on their respective categories of algebras, $\mathrm{PCwF} \Rightarrow \mathrm{CwF}$. We'll address one of the most immediate category-theoretic questions prompted by the appearance of a forgetful functor: *does it have a left adjoint?*

Proposition 2.2.5 (Free Polarization)**.** *The forgetful functor* $\mathrm{PCwF} \to \mathrm{CwF}$ *has a left adjoint.*

In words: any CwF $\mathcal{C}$ can be upgraded to a PCwF by the *free addition* of the polarization; every context Γ, substitution σ, and type A is given a *formal negation* (Γ^-, σ^-, and A^-, respectively), satisfying the requirements of Definition 2.1.16—and *only* those requirements. We won't explore this construction further, other than to say that the polarizations we care about (on the category and preorder models, and consequently the groupoid and setoid models) are *not free*. We already established that $\Gamma \cong \Gamma^-$ for all Γ in the setoid and groupoid models; $\Gamma \cong \Gamma^-$ is definitely not provable from the laws of a PCwF (take the category model PCwF as a counterexample), hence it shouldn't hold in the free polarization.

Let's now pivot from the theory of the category PCwF to considering the structure of individual PCwFs. We remarked previously that there's nothing really connecting the deep and shallow polarities of a PCwF together—it makes perfect sense to define 'deeply-polarized CwFs' and 'shallowly-polarized CwFs' with just one kind of polarization. However, it's hard to say much about these in isolation from each other: the real intrigue in a PCwF lies in how the deep and shallow polarity *interact*. Mixing the context-, substitution-, and type-negation operators together, we find that each PCwF interprets *two* type theories: the usual one, and its negative 'alter ego'.

Definition 2.2.6. Consider a PCwF with underlying CwF $\mathcal{C} = (\mathsf{Con}, \mathsf{Sub}, \mathsf{Ty}, \mathsf{Tm}, \triangleright^+, \ldots)$. Define the **negative CwF structure** $\mathcal{C}^-$ to have

- the same underlying category-with-terminal-object $(\mathsf{Con}, \mathsf{Sub}, \ldots, \bullet, \ldots)$ as $\mathcal{C}$;

is essential to him; and for this reason we must add a little under every heading." [Har92, Sect. 26]

[7] A onegat telescope!

- $\mathsf{Ty}^- \ \Gamma$ defined to be $\mathsf{Ty}(\Gamma^-)$;
- $\mathsf{Tm}^-(\Gamma, A)$ defined to be $\mathsf{Tm}(\Gamma^-, A^-)$;[a]
- for $A : \mathsf{Ty}^- \ \Gamma$, the **negative context extension** $\Gamma \rhd^- A : \mathsf{Con}$ defined by

$$\Gamma \rhd^- A := (\Gamma^- \rhd^+ A^-)^- \tag{2.2.7}$$

[a] Check that this is well-formed: we want Tm^- to be a presheaf on the category of elements of Ty^-, so $A : \mathsf{Ty}^- \ \Gamma$, i.e. $A : \mathsf{Ty}(\Gamma^-)$. Hence $A^- : \mathsf{Ty}(\Gamma^-)$ too, and thus $\mathsf{Tm}(\Gamma^-, A^-)$ makes sense.

Recall that, in the category model, types $A : \mathsf{Ty} \ \Gamma$ are equivalent to *split opfibrations* over Γ (in the sense of [AL19]). As we might expect, types in context Γ^- (i.e. elements of $\mathsf{Ty}^- \ \Gamma$) are equivalent to split *fibrations*.[8] Thus, the negative CwF can be seen as a type theory of fibrations, to complement the positive CwF's type theory of opfibrations.

Consider Figure 2.4, which defines $\mathsf{Ty} \ \Gamma$ and $\mathsf{Ty}^- \ \Gamma$ for the preorder model[9] as displayed preorders over Γ with opposite fibrancy data, rather than defining Ty^- in terms of Ty. Notice that it's only in the fibrancy data that the two differ—displayed preorders over Γ^- and Γ are equivalent, but *fibrant* displayed preorders on Γ^- (that is, split *opfibrations* over Γ^-) correspond to split *fibrations* on Γ (and vice versa). This is one further reason that the preorder model is a more appropriate choice for studying polarity than the concrete CwF $\mathscr{C}(\mathfrak{PreOrd})$—if $\mathsf{Ty} \ \Gamma$ and $\mathsf{Ty}^- \ \Gamma$ are the same for all Γ, then our (shallow) polarity is trivial.

This figure also shows that Ty^- can be defined directly in the preorder model (and likewise in the category model), instead of defining it as the composition of Ty with the $(\_)^-$ endofunctor. We might wonder if "PCwF" could instead be articulated in terms of this dual-CwF structure instead of the endofunctor-and-endo-transformation definition given above. However, careful comparison of Definition 2.1.16 with Definition 2.2.6 will reveal that the former actually contains more data: given two CwFs $\mathcal{C}$ and $\mathcal{C}^-$ on the same underlying category $(\mathsf{Con}, \mathsf{Sub})$, there's no apparent way to recover the endofunctor and endo-natural transformation that define a PCwF. Indeed, we can come up with examples of two CwFs that share the underlying category but which don't arise as the positive and negative CwFs of some PCwF. We need to state some connection between the CwFs, such as the following.

Proposition 2.2.8. *The "positive" and "negative" CwFs, $\mathcal{C}$ and $\mathcal{C}^-$, of a PCwF are isomorphic: the maps*

$$\begin{aligned} \Gamma : \mathsf{Con} &\mapsto \Gamma^- & &: \mathsf{Con} \\ \sigma : \mathsf{Sub} \ \Delta \ \Gamma &\mapsto \sigma^- & &: \mathsf{Sub} \ \Delta^- \ \Gamma^- \\ A : \mathsf{Ty} \ \Gamma &\mapsto A^- & &: \mathsf{Ty}^-(\Gamma^-) \\ t : \mathsf{Tm}(\Gamma, A) &\mapsto t & &: \mathsf{Tm}^-(\Gamma^-, A^-) \end{aligned}$$

form a CwF-morphism $\mathcal{C} \to \mathcal{C}^-$, which has an inverse $\mathcal{C}^- \to \mathcal{C}$ (also given by $\Gamma \mapsto \Gamma^-$,

[8] For us, the covariant notion (Ty) is primary, and we attach modifiers like "negative" to indicate the contravariant notion (Ty^-). Unfortunately, this is the opposite of the convention in the fibrations literature (going back to Grothendieck), for whom the contravariant notion (fibration) is primary and the covariant notion (opfibration) is derived. We avoid "op" in our notation partially for this reason.

[9] The same can be done for the category model, but would require a lot more data.

pseudoAgda

```
record opfib ((Γ,≤,...) : 𝔓reℭrd-Alg) ((Γ^D,≤^D,...) : 𝔓reℭrd-DAlg
    (Γ,≤,...)) : Set
 where
  coe⁺  : {γ₀ γ₁ : Γ} → γ₀ ≤ γ₁ → Γ^D γ₀ → Γ^D γ₁
  coh⁺  : {γ₀ γ₁ : Γ}(γ₀^D : Γ^D γ₀) → (γ₀₁ : γ₀ ≤ γ₁) → γ₀^D ≤^D (coe⁺ γ₀₁ γ₀^D)
  cart⁺ : {γ₀ γ₁ γ₂ : Γ}(γ₀^D : Γ^D γ₀){γ₀₁ : γ₀ ≤ γ₁}{γ₀₂ : γ₀ ≤ γ₂} →
    (γ₂^D : Γ^D γ₂) → {γ₀₂^D : γ₀^D ≤^D γ₂^D} →
    coe⁺ γ₀₁ γ₀^D ≤^D γ₂^D

Ty (Γ : 𝔓reℭrd-Alg) : Set
Ty Γ = (Γ^D : 𝔓reℭrd-DAlg Γ) × opfib Γ Γ^D

record fib ((Γ,≤,...) : 𝔓reℭrd-Alg) ((Γ^D,≤^D,...) : 𝔓reℭrd-DAlg (Γ,≤,...)) :
    Set
 where
  coe⁻  : {γ₀ γ₁ : Γ} → γ₀ ≤ γ₁ → Γ^D γ₁ → Γ^D γ₀
  coh⁻  : {γ₀ γ₁ : Γ}(γ₁^D : Γ^D γ₁) → (γ₀₁ : γ₀ ≤ γ₁) → (coe⁻ γ₀₁ γ₁^D) ≤^D γ₁^D
  cart⁻ : {γ₀ γ₁ γ₂ : Γ}(γ₂^D : Γ^D γ₂){γ₀₁ : γ₀ ≤ γ₁}{γ₀₂ : γ₀ ≤ γ₂} →
    (γ₀^D : Γ^D γ₀) → {γ₀₂^D : γ₀^D ≤^D γ₂^D} →
    γ₀^D ≤^D coe⁻ γ₀₁ γ₂^D

Ty⁻ (Γ : 𝔓reℭrd-Alg) : Set
Ty⁻ Γ = (Γ^D : 𝔓reℭrd-DAlg Γ) × fib Γ Γ^D
```

Figure 2.4: Ty and Ty^- in the preorder model, defined directly in terms of (op)fibrancy data on displayed preorders.

pseudoAgda

```
_▹⁻_ : (Γ : Con) → Ty Γ⁻ → Con
 | Γ ▹⁻ A | = (γ : | Γ |) × | A(γ) |
 (Γ ▹⁻ A) [ (γ₀,a₀) , (γ₁,a₁) ] = (γ₀₁ : Γ [γ₀,γ₁]) × (A γ₀) [ a₀ , A γ₀₁ a₁ ]
```

Figure 2.5: $\triangleright^-$ in the category model, defined from scratch.

$A \mapsto A^-$, etc.).

Notice that this CwF morphism is the identity on terms; by the involutive property of negation, $\mathsf{Tm}^-(\Gamma^-, A^-) = \mathsf{Tm}(\Gamma, A)$, and thus it's well-typed to send t to itself. This corresponds to the fact that Definition 2.1.16 contains no operation on terms. Now, this statement could perhaps be the basis of an alternative definition of 'PCwF': we could say that a PCwF consists of a pair of CwFs on the same underlying category, connected by a self-inverse CwF morphism. There are some aspects of this definition which require further reflection (do we want to require that the CwF morphism is the identity on terms?), but perhaps these can be resolved. We suspect that some version of this formulation might make it possible to develop polarized type theory in the SOGAT setting, but we leave this to future work.

The formulation of 'PCwF' given in Definition 2.1.16 is better suited to our purposes anyways. Moving forward, we won't have much use for Ty^- and Tm^-—the only part of the negative CwF structure we'll be interested in is the context extension operator $\triangleright^-$. Rather than treating $\triangleright^-$ as an operation living in an entirely separate CwF as $\triangleright^+$, we'll want to be able to explore the interaction of the two. For instance, a simple consequence of Equation 2.2.7 (and the fact that negation is an involution) is that, for any $A : \mathsf{Ty}\ \Gamma$,

$$(\Gamma \triangleright^+ A)^- = \Gamma^- \triangleright^- A^-. \tag{2.2.9}$$

That is, the negation 'distributes' over context extension. In what follows, these will be the kind of 'polarity calculations' we need to perform; the negative context extension operator is ultimately just an abbreviation (Equation 2.2.7), but a convenient one.

To work with negative context extension, we need to develop its *local representability* data. The claim that Tm^- is locally representable with respect to Ty^-—which is part of the claim in Definition 2.2.6 that $\mathcal{C}^-$ is a CwF—is elaborated to the following.

Proposition 2.2.10. *In any PCwF, we have an isomorphism*

$$\mathsf{Sub}\ \Delta\ (\Gamma \triangleright^- A) \cong \sum_{\sigma :\ \mathsf{Sub}\ \Delta\ \Gamma} \mathsf{Tm}(\Delta^-, A[\sigma^-]^-)$$

natural in Δ.

Write $\mathsf{p}_{-,A} : \mathsf{Sub}\ (\Gamma \triangleright^- A)\ \Gamma$ and $\mathsf{v}_{-,A} : \mathsf{Tm}((\Gamma \triangleright^- A)^-, A[\mathsf{p}^-_{-,A}]^-)$ for the results of applying the left-to-right direction to $\mathsf{id}_{\Gamma \triangleright^- A}$. For any $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$ and $t : \mathsf{Tm}(\Delta^-, A[\sigma^-]^-)$, write $\sigma \mathbin{,_-} t$ for the right-to-left direction applied to (σ, t).

Proof. The key definitions are as follows.

$$\mathsf{p}_{-,A} := \mathsf{p}^-_{+,A^-} \tag{2.2.11}$$

$$\mathsf{v}_{-,A} := \mathsf{v}_{+,A^-} \tag{2.2.12}$$

$$\sigma \mathbin{,_-} t := (\sigma^- \mathbin{,_+} t)^- \tag{2.2.13}$$

The left-to-right map sends τ to $(\mathsf{p}_{-,A} \circ \tau, \mathsf{v}_{-,A}[\tau^-])$. The already-established laws of the polarity calculus (and the positive local representability) tell us that this is

natural and inverse to the $\_ \,,_- \,\_$ operation: for instance,

$$\begin{aligned}
\mathsf{p}_{-,A} \circ \tau \;\,,_- \;\, \mathsf{v}_{-,A}[\tau^-] &= \left((\mathsf{p}_{-,A} \circ \tau)^- \;\,,_+ \;\, \mathsf{v}_{-,A}[\tau^-]\right)^- \\
&= \left(\mathsf{p}^-_{-,A} \circ \tau^- \;\,,_+ \;\, \mathsf{v}_{-,A}[\tau^-]\right)^- \\
&= \left(\mathsf{p}_{+,A^-} \circ \tau^- \;\,,_+ \;\, \mathsf{v}_{+,A^-}[\tau^-]\right)^- \\
&= \left((\mathsf{p}_{+,A^-} \,,_+\, \mathsf{v}_{+,A^-}) \circ \tau^-\right)^- \\
&= (\tau^-)^- \\
&= \tau.
\end{aligned}$$

□

Proposition 2.2.14. *There is a bijection between terms in Γ^- of type A^- and sections of $\mathsf{p}_{-,A}$:*

$$\mathsf{Tm}(\Gamma^-, A^-) \underset{\mathsf{v}_{-,A}[(\_)^-]}{\overset{(\mathsf{id}_\Gamma \,,_- \,\_)}{\rightleftarrows}} (\tau \colon \mathsf{Sub}\ \Gamma\ (\Gamma \rhd^- A)) \times \mathsf{p}_{-,A} \circ \tau = \mathsf{id}_\Gamma.$$

Definition 2.2.15. Given $\sigma \colon \mathsf{Sub}\ \ \Delta\ \ \Gamma$ and a negative telescope $A_0 \colon \mathsf{Ty}\ \ \Gamma^-$, $A_1 \colon \mathsf{Ty}(\Gamma \rhd^- A_0)^-$, ...$A_n$, define the substitution

$$\begin{aligned}
\mathsf{q}^-(\sigma; A_0, \ldots, A_n) \quad : \quad & \mathsf{Sub}\ (\Delta \rhd^- A_0[\sigma^-] \rhd \cdots \rhd A_n[\mathsf{q}^-(\sigma; A_0, \ldots, A_{n-1})]) \\
& (\Gamma \rhd^- A_0 \rhd^- \cdots \rhd^- A_n)
\end{aligned}$$

by

$$\mathsf{q}^-(\sigma; A_0, \ldots, A_n) := (\mathsf{q}(\sigma^-, A_0^-, \ldots, A_n^-))^-.$$

Equivalently, define it by induction:

$$\begin{aligned}
\mathsf{q}^-(\sigma, A_0) &:= (\sigma \circ \mathsf{p}_- \,,_- \,\mathsf{v}_-) \\
\mathsf{q}^-(\sigma; A_0, \ldots, A_{n+1}) &:= (\mathsf{q}^-(\sigma; A_0, \ldots, A_n) \circ \mathsf{p}_- \,,_- \,\mathsf{v}_-).
\end{aligned}$$

Let's conclude this section with a treatment of *dependent types* in the category model. Defining dependent pair types, i.e. Σ-types, won't require any of the polarity mechanisms we've discussed: Σ-types are "positive" in the sense that both A and B in the type ΣAB are covariant in the context. Hofmann and Streicher do not explicitly spell out the Σ-types in the groupoid model, but if we do, we find that it's not ever necessary to invert any morphism. Thus there is no *polarity problem*; we can take the notion of Σ-types from undirected type theory [Hof97, Defn. 3.18][10] verbatim.

Definition 2.2.16. A (P)CwF **supports Σ-types** if the following data are given.

[10] Note that our notion of "supporting Σ-types" is what Hofmann calls "supporting Σ-types *in the strict sense*."

PSEUDOAGDA

```
Σ : {Γ : Con} ⟶ (A : Ty Γ) ⟶ Ty (Γ ▷⁺ A) ⟶ Ty Γ
Σ[] : {Δ Γ : Con}(A : Ty Γ)(B : Ty (Γ ▷⁺ A))(σ : Sub Δ Γ) ⟶
  Σ(A,B)[ σ ] = Σ {Γ=Δ} (A [ σ ]) (B [ q(σ;A) ])
pair : {Γ}{A}{B} ⟶ Sub (Γ ▷⁺ A ▷⁺ B) (Γ ▷⁺ Σ(A,B))
pair[] : {Δ Γ : Con}(A : Ty Γ)(B : Ty (Γ ▷⁺ A))(σ : Sub Δ Γ) ⟶
  q(σ;Σ(A,B)) ∘ pair = pair ∘ q(σ;A,B)
p∘pair : {Γ}{A}{B} ⟶ (p (Σ(A,B))) ∘ pair = (p A) ∘ (p B)
Σ-elim : {Γ}{A}{B}{M : Ty (Γ ▷⁺ Σ(A,B))} ⟶
  Tm(Γ ▷⁺ A ▷⁺ B, M[pair]) ⟶ Tm(Γ ▷⁺ Σ(A,B), M)
Σβ : {Γ}{A}{B}{M}(m : Tm(Γ ▷⁺ A ▷⁺ B, M[pair])) ⟶
  (Σ-elim m)[ pair ] = m
Ση : {Γ}{A}{B}{M}(m : Tm(Γ ▷⁺ A ▷⁺ B, M[pair])) ⟶
  (z : Tm(Γ ▷⁺ A ▷⁺ B, M[pair])) ⟶ (z[ pair ] = m) ⟶ z = Σ-elim m
Σ-elim[] : {Δ}{Γ}{A}{B}{M} ⟶
  (m : Tm(Γ ▷⁺ A ▷⁺ B, M[pair])) ⟶ (σ : Sub Δ Γ) ⟶
  (Σ-elim m)[ q(σ;Σ(A,B)) ] = Σ-elim {Γ=Δ} (m [ q(σ;A,B) ])
```

The construction of Σ-types in the category model is relatively straightforward, essentially the construction of positive context extension in a different guise: the category $\Sigma(A,B)\ \gamma$ has pairs $(a : |A\ \gamma|, b : |B(\gamma,a)|)$ as objects and morphisms given by

$$(\Sigma(A,B)\ \gamma)\ [(a,b),(a',b')] := (\overline{a} : (A\ \gamma)[a,a'])\times B(\gamma,a')\ [B(\mathrm{id}_\gamma,\overline{a})\ b,b'].$$

The morphism part of the $\Sigma(A,B)$ functor is spelled out explicitly in Figure 2.6 for the sake of completeness.

As usual, we will write $A\times B$ for $\Sigma(A,B[\mathsf{p}])$ when B doesn't depend on A, i.e. $A,B : \mathsf{Ty}\ \Gamma$. As we might expect, the type-negation distributes over non-dependent products:

$$\mathsf{Tm}(\Gamma,(A\times B)^-)\cong\mathsf{Tm}(\Gamma,A^-\times B^-).$$

However, it's not clear (with the current machinery) how to even *state* the dependent version: if $B : \mathsf{Ty}(\Gamma\rhd^+ A)$, then what's the $B' : \mathsf{Ty}(\Gamma\rhd^+ A^-)$ that should appear on the right-hand side of the 'dependent version':

$$\mathsf{Tm}(\Gamma,(\Sigma AB)^-)\cong\mathsf{Tm}(\Gamma,\Sigma\ A^-\ B')?$$

This is one of many problems which will prompt us to introduce *neutrality* in the next section; there, we'll be able to give a satisfactory answer (Proposition 2.3.13).

Defining the appropriate notion of Π-type for PCwFs requires a deeper engagement with the polarity calculus—indeed, they are our main motivation for studying negative context extension. If we examine the groupoid model semantics of Π-types [HS95, Section 4.6], we find that it makes use of the fact that contexts are groupoids and types are families of groupoids. Thus, we have a *polarity problem* preventing us from just copying the definition into the category model, and must work out the variances.

pseudoAgda

$$
\begin{array}{l}
\Sigma : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\ \Gamma) \longrightarrow \mathsf{Ty}(\Gamma \triangleright^{+} A) \longrightarrow \mathsf{Ty}\ \Gamma \\
\quad |\Sigma(A,B)\ \gamma| := (a : |A\ \gamma|) \times |B(\gamma,a)| \\
\quad (\Sigma(A,B)\ \gamma)[\ (a,b)\ ,\ (a',b')\ ] := (\overline{a} : A\gamma\ [\ a\ ,\ a'\ ]) \times B(\gamma,a')\ [\ B(\mathsf{id}_{\gamma}, \overline{a})\ b\ ,\ b'\ ] \\
\\
\quad \Sigma(A,B)\ \gamma_{01} : (\Sigma(A,B)\ \gamma_0) \Rightarrow (\Sigma(A,B)\ \gamma_1) \\
\quad \Sigma(A,B)\ \gamma_{01}\ (a_0,b_0) := (A\ \gamma_{01}\ a_0,\ B(\gamma_{01},\ \mathsf{id})\ b_0) \\
\quad \Sigma(A,B)\ \gamma_{01}\ \left(\overline{a}_0,\ \overline{b}_0\right) := \left(A\ \gamma_{01}\ \overline{a}_0,\ B(\gamma_{01},\ \mathsf{id})\ \overline{b}_0\right) \\
\\
\mathsf{pair} : \mathsf{Sub}\ (\Gamma \triangleright^{+} A \triangleright^{+} B)\ (\Gamma \triangleright^{+} \Sigma(A,B)) \\
\quad \mathsf{pair}\ (\gamma,a,b) := (\gamma,(a,b)) \\
\\
\quad \mathsf{pair} : \\
\qquad (\gamma_{01} : \Gamma\ [\ \gamma_0\ ,\ \gamma_1\ ]) \longrightarrow \\
\qquad (a_{01} : (A\ \gamma_1)\ [\ A\ \gamma_{01}\ a_0\ ,\ a_1\ ]) \longrightarrow \\
\qquad (b_{01} : B(\gamma_1,a_1,b_1)\ [\ B(\gamma_{01},a_{01})\ b_0\ ,\ b_1\ ]) \longrightarrow \\
\qquad (\Gamma \triangleright^{+} \Sigma(A,B))\ [\ (\gamma_0,(a_0,b_0))\ ,\ (\gamma_1,(a_1,b_1))\ ] \\
\quad \mathsf{pair}\ \gamma_{01}\ a_{01}\ b_{01} := (\gamma_{01},(a_{01},b_{01})) \\
\\
\Sigma\text{–elim}\ \{M\} : \mathsf{Tm}(\Gamma \triangleright^{+} A \triangleright^{+} B,\ M[\ \mathsf{pair}\ ]) \longrightarrow \mathsf{Tm}(\Gamma \triangleright^{+} \Sigma(A,B),\ M) \\
\quad \Sigma\text{–elim m} : (\gamma : |\Gamma|) \longrightarrow ((a,b) : |\Sigma(A,B)\ \gamma|) \longrightarrow |M(\gamma,(a,b))| \\
\quad \Sigma\text{–elim m}\ \gamma\ (a,b) := \mathsf{m}\ \gamma\ a\ b \\
\\
\quad \Sigma\text{–elim m} : \\
\qquad (\gamma_{01} : \Gamma\ [\ \gamma_0\ ,\ \gamma_1\ ]) \longrightarrow \\
\qquad ((a_{01},b_{01}) : (\Sigma(A,B)\ \gamma_1)\ [\ \Sigma(A,B)\ \gamma_{01}\ (a_0,b_0)\ ,\ (a_1,\ b_1)\ ]) \longrightarrow \\
\qquad (M(\gamma_1,(a_1,b_1)))\ [\ M(\gamma_{01},(a_{01},b_{01}))\ (\mathsf{m}\ \gamma_0\ a_0\ b_0)\ ,\ \mathsf{m}\ \gamma_1\ a_1\ b_1\ ] \\
\quad \Sigma\text{–elim m}\ \gamma_{01}\ (a_{01},b_{01}) := \mathsf{m}\ \gamma_{01}\ a_{01}\ b_{01}
\end{array}
$$

Figure 2.6: Semantics of Σ-types in the category model

Function types are the most essential place where variance becomes a concern. As is well-known in category theory and computer science, a set/type A *appears negatively* in the set/type $A \to B$. That is, $A \to B$ as a whole depends contravariantly on A: a function $A \to A'$ induces an operation $A' \to B$ to $A \to B$, not the other way around. The same goes for dependent functions: in the type $\Pi(A, B)$, it is the type A which *appears negatively* and B which appears positively. In our polarity calculus, we must capture this as dependence on the context: it makes sense to form $\Pi(A, B)$ when A is contravariant in the context, and B is covariant in the extended context. More precisely, we make the following definition of Π-types in a PCwF.[11]

Definition 2.2.17. A PCwF supports **polarized Π-types** if it is equipped with the following data.

PSEUDOAGDA

```
Π   : {Γ : Con} → (A : Ty Γ⁻) → Ty (Γ ▷⁻ A) → Ty Γ
lam : {Γ}{A}{B} → Tm (Γ ▷⁻ A, B) → Tm (Γ, Π(A,B))
app : {Γ}{A}{B} → Tm (Γ, Π(A,B)) → Tm (Γ ▷⁻ A, B)
Πβ  : {Γ}{A}{B} → (t : Tm (Γ ▷⁻ A, B)) → app(lam t) = t
Πη  : {Γ}{A}{B} → (f : Tm (Γ, Π(A,B))) → lam(app f) = f
Π[] : {Δ Γ}{A}{B} → (σ : Sub Δ Γ) →
  Π(A,B)[ σ ] = Π(A[ σ⁻ ], B [ q(σ⁻;A⁻)⁻ ])
lam[] : {Δ Γ}{A}{B}{t} → (σ : Sub Δ Γ) →
  (lam t)[ σ ] = lam (t [ q(σ⁻;A⁻)⁻ ] )
```

As usual, the category model semantics of polarized Π-types (given in Figure 2.7) are just the groupoid model semantics ([HS95, Section 4.6]), with proper care paid to the polarities. To define the object part, we make use of the fibrant-displayed nature of the category model: for each $\gamma : |\Gamma|$, we can form the context $A(\gamma)$ and the type $B_\gamma := B(\gamma, \_)$ in this context. The category $\Pi(A, B)\ \gamma$ has terms of B_γ as objects and, regarding such terms as functors $A(\gamma) \Rightarrow (A(\gamma) \triangleright^+ B_\gamma)$, the appropriate notion of *natural transformation* as morphisms. In the non-dependent case, we write $A \to B$ for $\Pi(A, B[\mathsf{p}\_])$; note that $A : \mathsf{Ty}\ \Gamma^-$ but $B : \mathsf{Ty}\ \Gamma$ in this case. As spelled out in Figure 2.9, terms of type $A \to B$ and terms of $(A \to B)^-$ both consist of families of functors $A(\gamma) \Rightarrow B(\gamma)$ indexed over objects γ, but are natural over $\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]$ in opposite ways—in the empty context, a function term $F : A \to B$ is literally a functor $A \Rightarrow B$. Later, we will view non-dependent function types as *synthetic functors*, so it is reassuring that they are indeed interpreted as functors.

We also have the following interaction between the polarity calculus and non-dependent function types.

Proposition 2.2.18. *In the category model, there is an operation for every* $\Gamma : \mathsf{Con}$,

[11] Note that this is the statement that appears in [LH11], modulo differences in notation.

pseudoAgda

```
Π : {Γ : Con}(A : Ty Γ⁻) ⟶ Ty(Γ ▷⁻ A) ⟶ Ty Γ
 |Π(A,B) γ| := Tm(A(γ), B_γ) where
    B_γ : Ty(A γ)
     B_γ a := B(γ,a)
     B_γ (x : (A γ)[ a , a' ]) := B(id_γ,x)

 (Π(A,B) γ)[ θ , θ' ] := Transform(θ,θ') where
    record Transform : (θ θ' : Tm(A(γ), B_γ)) ⟶ Set where
     component : (a : |A γ|) ⟶ (B(γ,a)) [ θ a , θ' a ]
     naturality : (x : (A γ) [ a , a' ]) ⟶
      (θ' x) ∘ B(γ,x)(component a) = (component a') ∘ (θ x)

 Π(A,B) γ₀₁ : Tm(A(γ₀), B_γ0) ⟶ Tm(A(γ₁), B_γ1)
  (Π(A,B) γ₀₁ θ₀) (a₁ : |A γ₁|) := B(γ₀₁, id_A γ01 a1) (θ₀(A γ₀₁ a₁))
  (Π(A,B) γ₀₁ θ₀) (x₁ : A γ₁ [a₁ , a₁']) := B(γ₀₁, id_A γ01 a'1) (θ₀(A γ₀₁ x₁))

lam : Tm(Γ ▷⁻ A, B) ⟶ Tm(Γ, Π(A,B))
 lam t' : (γ : |Γ|) ⟶ | (Π(A,B)) γ |
  lam t' γ : (a : |A γ|) ⟶ |B(γ,a)|
  lam t' γ a := t'(γ,a)

  lam t' γ : (x : (A γ)[ a , a' ]) ⟶ B(γ,a')[ B(id_γ,x) (lam t' γ a), lam t' γ a' ]
  lam t' γ x := t'(id_γ,x)

 lam t' : (γ₀₁ : Γ[ γ₀ , γ₁ ]) ⟶
     Transform {γ = γ₁} (Π(A,B) γ₀₁ (lam t' γ₀), lam t' γ₁)
  component(lam t' γ₀₁) : (a₁ : |A γ₁|) ⟶
     B(γ₁,a₁)[ B(γ₀₁, id_A γ01 a1) (lam t' γ₀ (A γ₀₁ a₁)) , lam t' γ₁ a₁ ]
  component(lam t' γ₀₁) a₁ := t'(γ₀₁, id_A γ01 a1)

  naturality(lam t' γ₀₁) (x₁ : (A γ₁)[ a₁ , a₁' ]) := (Figure 2.8)

app : Tm(Γ, Π(A,B)) ⟶ Tm(Γ ▷⁻ A, B)
 app f : (γ : |Γ|) ⟶ (a : |A γ|) ⟶ |B(γ,a)|
 app f γ a := f γ a                          -- (f γ) : (a : |A γ|) ⟶ |B_γ a|

 app f :
     (γ₀₁ : Γ[ γ₀ , γ₁ ]) ⟶
     (a₀₁ : (Aγ₀)[ a₀ , A γ₀₁ a₁ ]) ⟶
     B(γ₁,a₁)[ B(γ₀₁,a₀₁) (app f γ₀ a₀), app f γ₁ a₁ ]
 app f γ₀₁ a₀₁ := (component(f γ₀₁) a₁) ∘ B(γ₀₁, id) (f γ₀ a₀₁)
```

Figure 2.7: Semantics of Π-types in the category model

$$
\begin{aligned}
& t'(\gamma_1, x_1) \circ B(\mathsf{id}_{\gamma_1}, x_1)(t'(\gamma_{01}, \mathsf{id}_{A\ \gamma_{01}\ a_1})) \\
&= t'(\mathsf{id}_{\gamma_1} \circ \gamma_{01}, (A\ \gamma_{01}\ x_1) \circ \mathsf{id}_{A\ \gamma_{01}\ a_1}) \\
&= t'(\gamma_{01}, A\ \gamma_{01}\ x_1) \\
&= t'(\gamma_{01} \circ \mathsf{id}_{\gamma_0}, (A\ \mathsf{id}_{\gamma_0}\ \mathsf{id}_{A\ \gamma_{01}\ a'_1}) \circ A\ \gamma_{01}\ x_1) \\
&= t'(\gamma_{01}, \mathsf{id}_{A\ \gamma_{01}\ a'_1}) \circ B(\gamma_{01}, \mathsf{id}_{A\ \gamma_{01}\ a'_1})(t'(\mathsf{id}_{\gamma_0}, A\ \gamma_{01}\ x_1))
\end{aligned}
$$

Figure 2.8: A naturality calculation for the morphism part of lam t′.

PSEUDOAGDA

```
-- Natural transformation
record Nat {γ₁} : (θ θ' : A(γ₁) ⇒ B(γ₁)) ⟶ Set where
  component : (a : |A(γ₁)|) ⟶ B(γ₁) [ θ a , θ' a ]
  naturality : (x₁ : (A γ₁) [ a₁ , a₁' ]) ⟶
    (θ' x₁) ∘ (component a₁) = (component a₁') ∘ (θ x₁)

-- F : Tm(Γ, (A ⟶ B)⁻)
F : (γ : |Γ|) ⟶ (A(γ) ⇒ B(γ))
F : (γ₀₁ : Γ [γ₀,γ₁]) ⟶ Nat F(γ₁) (B(γ₀₁) ∘ F(γ₀) ∘ A(γ₀₁))

-- G : Tm(Γ, A ⟶ B)
G : (γ : |Γ|) ⟶ (A(γ) ⇒ B(γ))
G : (γ₀₁ : Γ [γ₀,γ₁]) ⟶ Nat (B(γ₀₁) ∘ G(γ₀) ∘ A(γ₀₁)) G(γ₁)
```

Figure 2.9: Non-dependent functions in the category model

pseudoAgda

```
(_)⁻ : {Γ}{A : Ty Γ⁻}{B : Ty Γ} → Tm(Γ, (A → B)⁻) → Tm(Γ, A⁻ → B⁻)
  F⁻ : (γ : |Γ|) → (A⁻(γ) ⇒ B⁻(γ))
  F⁻ γ := (F(γ))^op          -- op endofunctor on Cat

  F⁻ : (γ₀₁ : Γ [ γ₀ , γ₁ ]) → Nat (B⁻(γ₀₁) ∘ F⁻(γ₀) ∘ A⁻(γ₀₁)) F⁻(γ₁)
    component(F⁻ γ₀₁) : (a₁ : |A γ₁|) →
      B(γ₁) [ F⁻ γ₁ a₁ , (B⁻(γ₀₁) ∘ F⁻(γ₀) ∘ A⁻(γ₀₁)) a₁ ]
    component (F⁻ γ₀₁) := component(F γ₀₁)

    naturality(F⁻ γ₀₁) := naturality(F γ₀₁)
```

Figure 2.10: Terms $\mathsf{Tm}(\Gamma, (A \to B)^-)$ are terms $\mathsf{Tm}(\Gamma, A^- \to B^-)$ in the category model.

$A : \mathsf{Ty}\ \Gamma^-, B : \mathsf{Ty}\ \Gamma$

$$(\_)^- : \mathsf{Tm}(\Gamma, (A \to B)^-) \quad \to \quad \mathsf{Tm}(\Gamma, A^- \to B^-)$$

such that $((F^-)^- = F)$.

The construction is given in Figure 2.10; the fact that it is an involution follows from the fact that the $(\_)^{\mathrm{op}}$ endofunctor is. If we wish to view types as synthetic categories (with A^- the opposite category of A) and functions as synthetic functors, then this is not a surprising statement: a term of $(A \to B)^-$ is a synthetic functor from A to B, and that should be usable as a functor on their opposite categories $A^- \to B^-$. The only wrinkle here is that $F : \mathsf{Tm}(\Gamma, (A \to B)^-)$, not $\mathsf{Tm}(\Gamma, A \to B)$; this is just to get the variance on γ_{01} right—note that the term G in Figure 2.9 has the wrong shape. In order to turn a $A \to B$ term into a $A^- \to B^-$ term, then we somehow need a way to turn it into a $(A \to B)^-$ term first; this is also a problem which *neutrality* will solve for us.

Finally, let's obtain the application operator appropriate to this notion of Π-types, allowing us to apply a function term F to an argument term, t. As usual, this is done by substituting t into app F, except we must use the *negative* CwF facilities.

Definition 2.2.19. In a PCwF supporting Π-types, define the operator

$$\_\,\$\,\_ : \{\Gamma\}\{A\}\{B\} \to \mathsf{Tm}(\Gamma, \Pi(A, B)) \to (t : \mathsf{Tm}(\Gamma^-, A^-)) \to \mathsf{Tm}(\Gamma, B[\mathsf{id}\ ,\_\ t])$$
$$F\,\$\,t := (\mathsf{app}\ F)[\mathsf{id}\ ,\_\ t]$$

This is the appropriate application operator for our setting, but it has a bit of an unsatisfying shape: in order to get the polarities to work out, we need the function F and its argument to live in different contexts (Γ and Γ^-, respectively). Moreover, given the nature of negative context extension, we also need t to be of type A^- in order for $\mathsf{id}\ ,\_\ t$ to be well-typed.

2.3 Neutrality

The 'polarity calculus' of a PCwF has the same function as any type discipline: enforce a formal condition on types and terms, and rule out as 'ill-formed' or 'ill-typed' those terms which do not meet the condition. For the polarity calculus, the formal condition being enforced is the appropriate respect of *variances*; for instance, $\Gamma \rhd^- A$ is only well-formed if $A : \mathsf{Ty}(\Gamma^-)$, not if $A : \mathsf{Ty}\ \Gamma$.[12] The formal constraints imposed by this polarity calculus prohibit various constructions which otherwise would've been well-formed and well-typed, namely those which fail to respect variance.[13] However, when designing a type discipline, one must take care to not be *too restrictive*, lest the theory becomes unwieldy or downright unusable. As we've started to see with our discussion of dependent types in the category model, our polarity calculus might be more restrictive than we want: we can't characterize $(\Sigma\ A\ B)^-$ compositionally in terms of A^- and B^-, we have to take a function and its argument from opposite contexts, and we can't view functions of type $A \longrightarrow B$ as functions $A^- \longrightarrow B^-$.

It gets worse. To see how, let's consider the de Bruijn indices of the *negative* CwF of a PCwF.

Remark 2.3.1. Define the de Bruijn indices for the negative CwF as follows:

$$\begin{aligned} \mathsf{v}_0^-\{\Gamma\}\{A_0\} &:= \mathsf{v}_{-,A_0} && : \mathsf{Tm}((\Gamma \rhd^- A_0)^-, A_0[\mathsf{p}^-_{-,A_0}]^-) \\ \mathsf{v}^-_{n+1}\{\Gamma\}\{A_0\ \ldots\ A_{n+1}\} &:= \mathsf{v}^-_n[\mathsf{p}^-_{-,A_{n+1}}] \\ &&& : \mathsf{Tm}((\Gamma \rhd^- A_0 \rhd^- \cdots \rhd^- A_{n+1})^-, A_0[\mathsf{p}^-_{-,A_{n+1}} \circ \cdots \circ \mathsf{p}^-_{-,A_0}]^-) \end{aligned} \tag{2.3.2}$$

where $\{A_0\ \ldots\ A_{n+1}\}$ is a telescope over Γ with respect to the *negative* CwF, i.e. $A_0 : \mathsf{Ty}\ \Gamma^-$, $A_1 : \mathsf{Ty}\ (\Gamma \rhd^- A_0)^-$, and so on.[a] However, recalling Equation 2.2.11 and Equation 2.2.12, we see that these reduce to the positive CwF de Bruijn indices with the context and type negated:

$$\begin{aligned} \mathsf{v}_0^- &:= \mathsf{v}_{-,A_0} && : \mathsf{Tm}((\Gamma \rhd^- A_0)^-, A_0[\mathsf{p}^-_{-,A_0}]^-) \\ &= \mathsf{v}_{+,A_0^-} && : \mathsf{Tm}(\Gamma^- \rhd^+ A_0^-, A_0[\mathsf{p}_{+,A_0^-}]^-) \\ &= \mathsf{v}_0 && : \mathsf{Tm}(\Gamma^- \rhd^+ A_0^-, A_0^-[\mathsf{p}_{+,A_0^-}]) \end{aligned}$$

i.e. the de Bruijn index 0 for $\Gamma \rhd^- A_0$ is the de Bruijn index 0 for $\Gamma^- \rhd^+ A_0^-$. Likewise for 1:

$$\begin{aligned} \mathsf{v}_1^- &:= \mathsf{v}_0^-[\mathsf{p}^-_{-,A_1}] && : \mathsf{Tm}((\Gamma \rhd^- A_0 \rhd^- A_1)^-, A_0[\mathsf{p}^-_{-,A_1} \circ \mathsf{p}^-_{-,A_0}]^-) \\ &= \mathsf{v}_0[\mathsf{p}_{+,A_1^-}] && : \mathsf{Tm}(\Gamma^- \rhd^+ A_0^- \rhd^+ A_1^-, A_0[\mathsf{p}_{+,A_1^-} \circ \mathsf{p}_{+,A_0^-}]^-) \\ &= \mathsf{v}_1 && : \mathsf{Tm}(\Gamma^- \rhd^+ A_0^- \rhd^+ A_1^-, A_0^-[\mathsf{p}_{+,A_1^-} \circ \mathsf{p}_{+,A_0^-}]) \end{aligned}$$

and so on. So Equation 2.3.2 could be replaced with

$$\mathsf{v}_n^-\{\Gamma\}\{A_0\ \ldots\ A_n\} \quad := \quad \mathsf{v}_n\{\Gamma^-\}\{A_0^-\ \ldots\ A_n^-\}.$$

[12] Unless $\Gamma^- = \Gamma$, as is the case where Γ is $\bullet$.

[13] At this stage, the "variance" is purely formal, a calculus of empty symbolic decorations:. Only once we introduce directed equality and transport in the next chapter will we be able to attach the usual category-theoretic meaning of *co-* and *contra-variant* to these polarities.

[a] By Equation 2.2.7, the requirement that $A_1 : \mathsf{Ty}\ (\Gamma \rhd^- A_0)^-$ is the same as saying $A_1 : \mathsf{Ty}\ (\Gamma^- \rhd^+ A_0^-)$. Carrying this through, we find that the "negative telescope" $\{A_0\ \ldots\ A_{n+1}\}$ on Γ is the same as the "positive telescope" $\{A_0^-\ \ldots\ A_{n+1}^-\}$ on Γ^-.

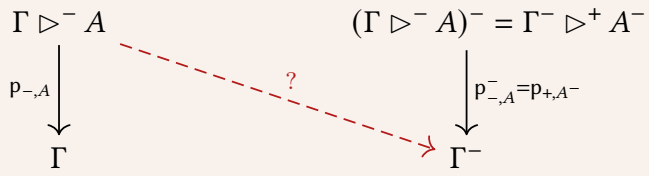
The variable terms v_n^- make sense and fit appropriately into the theory, but they don't mesh with the polarized Π-types in the way we'd hope. This can be seen most acutely in the case of the *identity function* (or lack thereof).

Remark 2.3.3. In *unpolarized* type theory, we can define the *identity function*

$$\mathsf{lam}\ \mathsf{v}_0 : \mathsf{Tm}(\Gamma, A \to A).$$

But this doesn't work with polarized type theory:

- $A \to A$ is not a well-formed type: in order to have A as the domain type, it must be the case that $A : \mathsf{Ty}\ \Gamma^-$. But to have A as the codomain type, we'd need a way to view A as a type in context $\Gamma \rhd^- A$, i.e. we'd need to substitute A along some substitution $\mathsf{Sub}\ (\Gamma \rhd^- A)\ (\Gamma^-)$. But we don't have such a substitution in general: both $\mathsf{p}_{-,A}$ and its negation, p_{+,A^-}, have the wrong shape.



- v_0^- lives in the wrong context to have lam applied: the negative variable term $\mathsf{v}_{-,A}$ is a term in the *negation* of $\Gamma \rhd^- A$, but lam expects a term in $\Gamma \rhd^- A$ itself.

Note that these are problems of *polarity*: by annotating our contexts with polarities—distinguishing Γ from Γ^- and $\Gamma \rhd^- A$ from $(\Gamma \rhd^- A)^-$, etc.—we have forbidden the identity function from our theory; our typing discipline is strong enough to declare $\mathsf{lam}\ \mathsf{v}_0^-$ and $A \to A$ is ill-formed. We would encounter similar problems if we were to try and define the composition of functions as well. As it stands, this type theory is simply *too polarized* for its function types to behave anything like how we want function types to behave.

The solution, then, is to *neutralize* some of the polarity. That is, we want some limited ways to overcome the difference between Γ and Γ^-, between $\Gamma \rhd^- A$ and $(\Gamma \rhd^- A)^-$, and so on. Let us emphasize that we want *limited* neutralization: we don't want to render the polarity calculus completely irrelevant, or else we'd be back to unpolarized type theory. Our solution will be to delineate a certain class of **neutral contexts**; these neutral contexts will still exist within a polarized type theory—indeed, it is only with respect to an ambient polarity calculus that "neutral" will have any meaning—but they will come equipped with further machinery that allows us to solve the above-mentioned problems (and further ones that will arise). Carefully maintaining this balance—keeping polarity around, but keeping it at bay—will create an environment where directed type theory is possible.

We will arrive at our notion of 'neutral-polarized CwF' (PCwFs with an appropriately-

chosen class of 'neutral contexts') in several stages. Throughout, the category model will be our guide, and its empty context the paradigm example of a neutral context. As our first approximation (or, rather, our *zeroth* approximation) of the NPCwF notion, we stipulate that our neutral contexts have enough structure to at least form the type of endofunctions—resolving the first issue from Remark 2.3.3. In the next stage, we add some machinery for manipulating contexts of the form $\Gamma \triangleright A$ where Γ is neutral but A is not. This will enable us to *write* the identity function, but the additional structure we require will let us do a number of intriguing and useful things. And then finally we can give the full definition of NPCwF and discuss the resulting type theory.

2.3.1 Phase Zero

As mentioned, our prototype neutral context will be the empty context. Recall that the empty context is *equal* to its own opposite, $\bullet = \bullet^-$. So, if $A : \mathsf{Ty}\ \bullet$, then A is also a type in $\bullet^-$, and thus the endofunction type $A \to A$ *is* well-formed. So, more generally, the first of the problems in Remark 2.3.3 disappears whenever $\Gamma = \Gamma^-$. But this is stronger than we need: so long as there's some substitution $\mathsf{e} : \mathsf{Sub}\ \Gamma^-\ \Gamma$, then $A[\mathsf{e}] \to A$ is a well-formed type in Γ; in the case of $\bullet$, the e in question just happens to be the identity $\mathsf{id}_\bullet$. Of course, the identity is particularly nice—if e is just *any* substitution, then $A[\mathsf{e}]$ might be a totally different type than A and calling terms of type $A[\mathsf{e}] \to A$ "endofunctions" would be inaccurate. But if e is at least an *isomorphism* (as $\mathsf{id}_\bullet$ is) then the term-substitution operation furnishes a bijection $\mathsf{Tm}(\Gamma, A) \cong \mathsf{Tm}(\Gamma^-, A[\mathsf{e}])$, making $A[e]$ a good-enough stand-in for A in the context Γ^-.

Remark 2.3.4. Given $A : \mathsf{Ty}\ \Gamma$ and $\mathsf{e} : \Gamma^- \cong \Gamma$ we can form the **endofunction** type

$$\dfrac{\dfrac{A : \mathsf{Ty}\ \Gamma \quad \mathsf{e} : \mathsf{Sub}\ \Gamma^-\ \Gamma}{A[\mathsf{e}] : \mathsf{Ty}(\Gamma^-)} \qquad A : \mathsf{Ty}\ \Gamma}{A[\mathsf{e}] \to A : \mathsf{Ty}\ \Gamma}$$

In the category model, the isomorphism e can be constructed not just for the empty context, but, in fact, for any *groupoid* context: if Γ is a groupoid, then $\mathsf{e} : \Gamma^- \cong \Gamma$ can be defined as the identity function on objects, and on morphisms the operation sending $\gamma_{10} : \Gamma^{\mathrm{op}}\ [\gamma_0, \gamma_1]$ to its inverse $\gamma_{10}^{-1} : \Gamma\ [\gamma_0, \gamma_1]$. So, while the endofunction type $A \to A$ is not well-formed in the category model when Γ is any *category*, the type $A[\mathsf{e}] \to A$ is well-formed when we require Γ to be a groupoid. Thus, "groupoid" is a sensible candidate for the notion of "neutral context" in the category model. Metatheoretically, this seems like a particularly elegant choice: the groupoid model is a paradigm model of *unpolarized* type theory and the category model a paradigm model of *polarized* type theory; if our goal is to articulate a *neutral* (i.e. unpolarized) fragment of polarized type theory, we should naturally start by exploring the role of groupoids in the category model. Accordingly, we'll take groupoids to be the neutral contexts of the category model.

Henceforth, we'll use the notation $\Gamma : \mathsf{NeutCon}$ to mean that Γ is neutral, i.e. a groupoid (or a setoid, when we're working with the preorder model). Now, remember that our ambient type theory is still polarized: even if Γ is a groupoid, the judgment

$A\colon \mathsf{Ty}\ \Gamma$ still means $A\colon \Gamma \Rightarrow \mathrm{Cat}$, not necessarily $A\colon \Gamma \Rightarrow \mathrm{Grpd}$. We'll also introduce notation for the latter: write $A\colon \mathsf{NeutTy}\ \Gamma$ to mean that $A\colon \Gamma \Rightarrow \mathrm{Grpd}$. Once again, the fibrancy of the category model means that context structure (the subcategory relation $\mathsf{NeutCon} \hookrightarrow \mathsf{Con}$) also plays out fiberwise (the sub-*presheaf* relation $\mathsf{NeutTy} \hookrightarrow \mathsf{Ty}$). The inclusion also gets reflected internally: later, we'll develop the types $A\colon \mathsf{Ty}\ \Gamma$ of the category model as *synthetic categories*; the neutral types, we'll find, are *synthetic groupoids*.

Though the set $\mathsf{NeutTy}\ \Gamma$ makes sense for any context Γ, we can say a bit more about *neutral types in a neutral context*, namely that

$$\frac{\Gamma\colon \mathsf{NeutCon} \quad A\colon \mathsf{NeutTy}\ \Gamma}{\Gamma \rhd^+ A\colon \mathsf{NeutCon}.} \tag{2.3.5}$$

If we did not know that *both* Γ and A were neutral, we would not be able to conclude that $\Gamma \rhd^+ A$ is. But if Γ is a groupoid and A is a family of groupoids over Γ, then we can show $\Gamma \rhd^+ A$ is a groupoid. This is just the context extension operation of the groupoid model: the subcategory inclusion $\mathrm{Grpd} \hookrightarrow \mathrm{Cat}$ is in fact a sub-*CwF* inclusion, of the groupoid model into (the *positive* CwF of) the category model—NeutCon and NeutTy are just our notations for the image of this inclusion. We do not need to write NeutSub and NeutTm, however, because the inclusion is both *full* and *locally full*.

Definition 2.3.6. A sub-CwF $\mathcal{D} \hookrightarrow \mathcal{C}$ is said to be

- **full** if the Sub-inclusion map $\mathsf{Sub}_{\mathcal{D}}\ \Delta\ \Gamma \to \mathsf{Sub}_{\mathcal{C}}\ \Delta\ \Gamma$ is surjective for all $\Delta, \Gamma\colon \mathsf{Con}_{\mathcal{D}}$
- **locally full** if the Tm-inclusion map $\mathsf{Tm}_{\mathcal{D}}\ (\Gamma, A) \to \mathsf{Tm}_{\mathcal{C}}\ (\Gamma, A)$ is surjective for all $\Gamma\colon \mathsf{Con}_{\mathcal{D}}$ and $A\colon \mathsf{Ty}_{\mathcal{D}}\ \Gamma$.

One other aspect worth mentioning about the e isomorphisms for groupoids is that the isomorphisms themselves are self-dual: since Γ being a groupoid implies that Γ^- is a groupoid too, we have the functors

$$\mathsf{e}_\Gamma\colon \Gamma^- \Rightarrow \Gamma \qquad \text{and} \qquad \mathsf{e}_{\Gamma^-}\colon \Gamma \Rightarrow \Gamma^-.$$

But notice that these are inverses: $e_\Gamma \circ e_{\Gamma^-} = \mathsf{id}_\Gamma$ and *vice versa*. Moreover, e_Γ^-—the application of the opposite endofunctor on Cat to e_Γ—is *also* equal to e_{Γ^-}. It's unclear whether these are coincidences of the category model, or some more fundamental fact. We'll take them as part of the definition of "NPCwF" because they are convenient technically—they save us from having to axiomatize the relationship between e_Γ, e_{Γ^-}, e_Γ^{-1}, and e_Γ^-—but it seems plausible that a weaker assumption than $\mathsf{e}_{\Gamma^-} = \mathsf{e}_\Gamma^{-1} = \mathsf{e}_\Gamma^-$ would suffice.

Let's now collect these observations into our initial version of "NPCwF". Our reason for calling it "zero-ary" will become more clear as we proceed.

Definition 2.3.7 (NPCwF—Version 0)**.** A (zero-ary) **neutral-polarized CwF** is a PCwF with

- a sub-collection of **neutral contexts** $\mathsf{NeutCon} \hookrightarrow \mathsf{Con}$ and
- a sub-presheaf[a] of **neutral types** $\mathsf{NeutTy} \hookrightarrow \mathsf{Ty}$

such that

- the empty context is neutral, $\bullet : \mathsf{NeutCon}$;
- the context extension $\Gamma \rhd^+ A$ of $\Gamma : \mathsf{NeutCon}$ by $A : \mathsf{NeutTy}\ \Gamma$ is a neutral context;
- NeutCon is closed under isomorphism; and
- for every $\Gamma : \mathsf{NeutCon}$, there is an isomorphism

$$\mathsf{e}_\Gamma : \Gamma^- \cong \Gamma$$

where
  - $\mathsf{e}_\Gamma^{-1} = \mathsf{e}_\Gamma^- = \mathsf{e}_{\Gamma^-}$ and
  - $\mathsf{e}_\Gamma \circ \sigma^- = \sigma \circ \mathsf{e}_\Delta$ for any $\Delta : \mathsf{NeutCon}$ and $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$.

[a]Note that this includes the requirement that neutral types are stable under substitution: if $A : \mathsf{NeutTy}\ \Gamma$, then $A[\sigma] : \mathsf{NeutTy}\ \Delta$ for all $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$.

The requirement that NeutCon is closed under isomorphism doesn't play much of a role in subsequent developments; we mainly include it as a more general fact (validated by the category model, of course) than the more specific instance that $\Gamma : \mathsf{NeutCon}$ implies $\Gamma^- : \mathsf{NeutCon}$. The fact that the sub-(P)CwF of neutral contexts and types is a full and locally full is made implicit in this definition: we don't define separate notions of "neutral substitution" nor "neutral term", so the only notion of substitution available between neutral contexts is the Sub of the ambient PCwF, and likewise for terms.

This definition begins to axiomatize the situation of the groupoid model as a neutral sub-PCwF of the category model, and, as we saw, begins to solve some of the issues raised at the beginning of the section about the usefulness of function types in polarized type theory. But it does not help us in actually *writing* functions, as this inherently involves extending the context, perhaps by a non-neutral type. So we must equip our neutral contexts with more machinery.

2.3.2 Phase One

What about the second issue of Remark 2.3.3? It's all well and good to be able to write down the type $A[\mathsf{e}] \to A$, but to actually be useful, we'll want to actually inhabit this type. Granting $\mathsf{e} : \Gamma^- \cong \Gamma$, we still have that

$$\begin{aligned} \mathsf{v}_0 &: \mathsf{Tm}(\Gamma \rhd^+ A, A[\mathsf{p}_{+,A}]) \\ \mathsf{v}_0^- &: \mathsf{Tm}((\Gamma \rhd^- A[\mathsf{e}])^-, A[\mathsf{e} \circ \mathsf{p}^-_{-,A[\mathsf{e}]}]). \end{aligned}$$

Neither of these can be lambda-abstracted to get a term of type $A[\mathsf{e}] \to A$, as both live in the wrong context. Well, let's return to the empty context of the category model. Notice that a closed type A in the category model is the same thing as a category, and that category is isomorphic to both $\bullet \rhd^+ A$ and $\bullet \rhd^- A[\mathsf{e}]$—since $\bullet$ has only the identity morphism, the morphisms of these categories are just given by the morphisms of A. Therefore, obtain an isomorphism

$$\bullet \rhd^- A[\mathsf{e}] \xrightarrow{\mathsf{ee}} \bullet \rhd^+ A.$$

LEAN–NOUGAT

```
def 𝔑₀𝔓ℭ𝔴𝔉 : GAT := {{
  include 𝔓ℭ𝔴𝔉;

    isNeut_Con : Con ⇒ U,
    isNeut_Con_prop : (Γ : Con) ⇒
        (nΓ nΓ' : isNeut_Con Γ) ⇒ nΓ ≡ nΓ',
    isNeut_Ty : {Γ : Con} ⇒ Ty Γ ⇒ U,
    isNeut_Ty_prop : {Γ : Con} ⇒ {A : Ty Γ} ⇒
        (nA nA' : isNeut_Ty A) ⇒ nA ≡ nA',

    neut_empty : isNeut_Con empty,
    neut_ext : {Γ : Con} ⇒ {A : Ty Γ} ⇒
        isNeut_Con Γ ⇒ isNeut_Ty A ⇒
        isNeut_Con (ext Γ A),
    neut_substTy : {Δ Γ : Con} ⇒ {A : Ty Γ} ⇒
        {σ : Sub Δ Γ} ⇒ isNeut_Ty A ⇒
        isNeut_Ty (substTy σ A),
    neut_iso : {Δ Γ : Con} ⇒
        (γ : Sub Δ Γ) ⇒ (δ : Sub Γ Δ) ⇒
        {_ : comp δ γ ≡ id Γ} ⇒
        {_ : comp γ δ ≡ id Δ} ⇒
        isNeut_Con Γ ⇒ isNeut_Con Δ,

    e : (Γ : Con) ⇒ {_ : isNeut_Con Γ} ⇒
        Sub (neg_Con Γ) Γ,
    e_sect : {Γ : Con} ⇒ {_ : isNeut_Con Γ} ⇒
        comp (neg_Sub (e Γ)) (e Γ) ≡ id (neg_Con Γ),
    e_retr : {Γ : Con} ⇒ {_ : isNeut_Con Γ} ⇒
        comp (e Γ) (neg_Sub (e Γ)) ≡ id Γ,
    e_dual : (Γ : Con) ⇒ {_ : isNeut_Con Γ} ⇒
        (neg_Sub (e Γ)) ≡ e (neg_Con Γ),
    e_square : {Γ : Con} ⇒ {_ : isNeut_Con Γ} ⇒
         {Δ : Con} ⇒ {_ : isNeut_Con Δ} ⇒
         (σ : Sub Δ Γ) ⇒
         comp (e Γ) (neg_Sub σ) ≡ comp σ (e Δ)
}}
```

Figure 2.11: The GAT $\mathfrak{N}_0\mathfrak{PCwF}$ of zero-ary NPCwFs.

So then observe $\mathsf{v}_0[\mathsf{ee}] : \mathsf{Tm}(\bullet \rhd^- A[\mathsf{e}], A[\mathsf{p}_{-,A[\mathsf{e}]}])$.[14] *This* we can lambda-abstract to a term of type $A[\mathsf{e}] \to A$, the identity function. If we spell out the category model semantics of $\mathsf{lam}(\mathsf{v}_0[\mathsf{ee}])$, we find that it is indeed the identity functor on the category A.

This isomorphism can *also* be defined in an arbitrary groupoid context.

Definition 2.3.8. For any $\Gamma : \mathsf{NeutCon}$ and $A : \Gamma \Rightarrow \mathrm{Cat}$, define

$$\mathsf{ee} : \Gamma \rhd^- A[\mathsf{e}] \cong \Gamma \rhd^+ A$$

to be

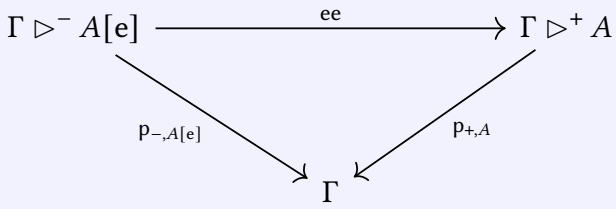
- the identity on objects (since $|\Gamma \rhd^- A[\mathsf{e}]| = |\Gamma \rhd^+ A|$);
- to have morphism part

$$\begin{aligned} \mathsf{ee} : \Big(& (\gamma_{01}, \mathsf{x}) : (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \times (A\ \gamma_0)\ [a_0, A\ \gamma_{01}^{-1}\ a_1] \Big) \\ \mapsto\ & (\gamma_{01}, A\ \gamma_{01}\ \mathsf{x}) : (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \times (A\ \gamma_1)\ [A\ \gamma_{01}\ a_0, a_1]; \end{aligned}$$

- to have inverse given by

$$\begin{aligned} \mathsf{ee}^{-1} : & ((\gamma_{01}, \mathsf{x}) : (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \times (A\ \gamma_1)\ [A\ \gamma_{01}\ a_0, a_1]) \\ \mapsto\ & (\gamma_{01}, A\ \gamma_{01}^{-1}\ \mathsf{x}) : (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \times (A\ \gamma_0)\ [a_0, A\ \gamma_{01}^{-1}\ a_1]. \end{aligned}$$

Proposition 2.3.9. *For the* ee *ismorphism defined in Definition 2.3.8, the following triangle commutes.*



What these ee isomorphisms allow us to do is sidestep the inconvenient negative CwF structure: we can operate the (dependent) function types—which are defined in terms of $\rhd^-$—by using the much more convenient *positive* CwF structure, and coercing back-and-forth with ee as needed.

Definition 2.3.10. Suppose $\Gamma : \mathsf{NeutCon}$ and $A : \mathsf{Ty}\ \Gamma$ is such that $\mathsf{ee} : \Gamma \rhd^- A[\mathsf{e}] \cong \Gamma \rhd^+ A$ satisfying the equality given in Proposition 2.3.9. Then, for any $B : \mathsf{Ty}(\Gamma \rhd^+ A)$,

[14] We silently use the fact that $\mathsf{p}_{+,A} \circ \mathsf{ee} = \mathsf{p}_{-,A[\mathsf{e}]}$. We know this because the codomain of these morphisms is $\bullet$, the terminal object.

define the following operations for working with Π-types.

$$\begin{aligned}
&\mathsf{lam}^+ \quad : \mathsf{Tm}(\Gamma \rhd^+ A, B) \to \mathsf{Tm}(\Gamma, \Pi(A[\mathsf{e}], B)) \\
&\mathsf{lam}^+\ b := \mathsf{lam}(b[\mathsf{ee}]) \\
&\mathsf{app}^+ \quad : \mathsf{Tm}(\Gamma, \Pi(A[\mathsf{e}], B)) \to \mathsf{Tm}(\Gamma \rhd^+ A, B) \\
&\mathsf{app}^+\ F : (\mathsf{app}\ F)[\mathsf{ee}^{-1}]
\end{aligned}$$

Write $\$^+$ for the operation

$$\_\ \$^+\ \_ : \{\Gamma\}\{A\}\{B\} \to \mathsf{Tm}(\Gamma, \Pi(A[\mathsf{e}], B)) \to (t' : \mathsf{Tm}(\Gamma, A)) \to \mathsf{Tm}(\Gamma, B[\mathsf{id} ,_+ t'])$$
$$F\ \$^+\ t' := (\mathsf{app}^+\ F)[\mathsf{id} ,_+ t'].$$

Definition 2.3.11. For $\Gamma : \mathsf{NeutCon}$ and $A : \mathsf{Ty}\ \Gamma$ as above, define the **identity function** on A:

$$I_A := \mathsf{lam}^+\mathsf{v}_0 \quad : \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to A).$$

Definition 2.3.12. For $\Gamma : \mathsf{NeutCon}$ and types $A, B, C : \mathsf{Ty}\ \Gamma$ such that lam^+ and app^+ are defined, define **function composition**:

$$\begin{aligned}
&_ \circ _ : \mathsf{Tm}(\Gamma, B[\mathsf{e}] \to C) \to \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to B) \to \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to C) \\
&G \circ F := \mathsf{lam}^+((\mathsf{app}^+\ G)[\mathsf{p} ,_+ \mathsf{app}^+\ F])
\end{aligned}$$

The ee isomorphisms allow us to overcome another roadblock we encountered: characterizing $(\Sigma\ A\ B)^-$. We're now able to state the following "negative version" of Proposition 1.2.14.

Proposition 2.3.13. *In the category model, there is a bijection*

$$\mathsf{Unpair}^- : \mathsf{Tm}(\Gamma, (\Sigma\ A\ B)^-) \ \cong\ (t : \mathsf{Tm}(\Gamma, A^-)) \times \mathsf{Tm}(\Gamma, B[\mathsf{ee}][\mathsf{id} ,_- t[\mathsf{e}]]^-) : \mathsf{Pair}^-.$$

Proof. First, observe the shape of a term $T : \mathsf{Tm}(\Gamma, (\Sigma\ A\ B)^-)$:

$$\begin{aligned}
T \quad &: (\gamma : |\Gamma|) \to (a : |A\ \gamma|) \times |B(\gamma, a)| \\
T \quad &: (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \to \\
&\qquad (a_{10} : (A\gamma_1)\ [(T\ \gamma_1)_{\#1},\ A\ \gamma_{01}\ (T\ \gamma_0)_{\#1}]) \\
&\qquad \times B(\gamma_1, A\ \gamma_{01}\ (T\ \gamma_0)_{\#1})\ [\\
&\qquad\quad B(\mathsf{id}_{\gamma_1}, a_{10})\ (T\ \gamma_1)_{\#2}, \\
&\qquad\quad B(\gamma_{01}, \mathsf{id})\ (T\ \gamma_0)_{\#2} \\
&\qquad]
\end{aligned}$$

So, given such a T, sending $\gamma \mapsto (T\ \gamma)_{\#1}$ and $\gamma_{01} \mapsto (T\ \gamma_{01})_{\#1}$ defines the required term $t : \mathsf{Tm}(\Gamma, A^-)$. So we just have to define a term

$$\psi : \mathsf{Tm}(\Gamma^-, B[\mathsf{ee}][\mathsf{id} ,_- t[\mathsf{e}]]^-)$$

i.e.

$$\begin{aligned}
\psi \quad &: (\gamma : |\Gamma|) \to |B(\gamma, (T\ \gamma)_{\#1})| \\
\psi \quad &: (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \to \\
&\quad B(\gamma_1, (T\ \gamma_1)_{\#1})\ [\\
&\qquad (T\ \gamma_1)_{\#2} \\
&\qquad B(\gamma_{01}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1})\ ((T\ \gamma_0)_{\#2}), \\
&\quad]
\end{aligned}$$

(see Equation 2.3.14 and Equation 2.3.15). For the object part, observe that $\gamma \mapsto (T\ \gamma)_{\#2}$ will work. To define the morphism part $\psi(\gamma_{01})$, the second component of $T(\gamma_{01})$ is almost right:

$$\begin{aligned}
(T\ \gamma_{01})_{\#2} \quad &: B(\gamma_1, A\ \gamma_{01}\ (T\ \gamma_0)_{\#1})\ [\\
&\qquad B(\mathsf{id}_{\gamma_1}, (T\ \gamma_{01})_{\#1})\ (T\ \gamma_1)_{\#2} \\
&\qquad B(\gamma_{01}, \mathsf{id})\ (T\ \gamma_0)_{\#2} \\
&\quad].
\end{aligned}$$

It just needs a small adjustment by the functor

$$B(\mathsf{id}_{\gamma_1}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1}) \quad : \quad B(\gamma_1, A\ \gamma_{01}\ (T\ \gamma_0)_{\#1}) \Rightarrow B(\gamma_1, (T\ \gamma_1)_{\#1}).$$

If we put

$$\psi(\gamma_{01}) := B(\mathsf{id}_{\gamma_1}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1})\ (T\ \gamma_{01})_{\#2}.$$

then this has the right domain by Equation 2.3.16 and the right codomain by Equation 2.3.17. The other direction is similar. □

Something interesting is happening here. B is a type in context $\Gamma \rhd^+ A$, that is, a type depending on a free variable of type A. But t is a term of type A^-. How did we manage to substitute t into B to make the above claim work? The composition $\mathsf{ee} \circ (\mathsf{id}\ ,_-\ t[\mathsf{e}])$ is where we make the sleight-of-hand: recall that terms of type A are in bijection with sections of $\mathsf{p} : \mathsf{Sub}\ (\Gamma \rhd^+ A)\ \Gamma$. By virtue of Proposition 2.3.9, the substitution $\mathsf{ee} \circ (\mathsf{id}\ ,_-\ t[\mathsf{e}])$ is such a section.

$$\Gamma \underset{\mathsf{p}}{\overset{\mathsf{id}\ ,_-\ t[\mathsf{e}]}{\rightleftarrows}} \Gamma \rhd^- A[\mathsf{e}] \xrightarrow{\mathsf{ee}} \Gamma \rhd^+ A$$

$$\begin{aligned}
\mathsf{p}_+ \circ \mathsf{ee} \circ (\mathsf{id}\ ,_-\ t[\mathsf{e}]) &= \mathsf{p}_- \circ (\mathsf{id}\ ,_-\ t[\mathsf{e}]) && \text{(Proposition 2.3.9)} \\
&= \mathsf{id}
\end{aligned}$$

So the corresponding term, $\mathsf{v}_+[\mathsf{ee}][\mathsf{id}\ ,_-\ t[\mathsf{e}]]$ is of type A. That is, we've turned a term of type A^- into a term of type A. This is quite a useful maneuver, so we give it its own notation.

$$\begin{aligned}(B[\mathsf{ee}][\mathsf{id} \mathbin{,\_} t[\mathsf{e}]])\ \gamma &= B[\mathsf{ee}]\ (\mathsf{id}\ \gamma, (T\ (\mathsf{e}\ \gamma))_{\#1}) \\ &= B[\mathsf{ee}]\ (\gamma, (T\ \gamma)_{\#1}) \\ &= B(\gamma, (T\ \gamma)_{\#1})\end{aligned} \tag{2.3.14}$$

$$\begin{aligned}&(B[\mathsf{ee}][\mathsf{id} \mathbin{,\_} t])\ (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1])\ (b_0 : |B(\gamma_0, (T\ \gamma_0)_{\#1})|) \\ &= B[\mathsf{ee}]\ (\mathsf{id}\ \gamma_{01}, (T\ (\mathsf{e}\ \gamma_{01}))_{\#1})\ b_0 \\ &= B[\mathsf{ee}]\ (\gamma_{01}, (T\ \gamma_{01}^{-1})_{\#1})\ b_0 \\ &= B(\gamma_{01}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1})\ b_0\end{aligned} \tag{2.3.15}$$

$$\begin{aligned}&B(\mathsf{id}_{\gamma_1}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1})\ \big(B(\mathsf{id}_{\gamma_1}, (T\ \gamma_{01})_{\#1})\ (T\ \gamma_1)_{\#2}\big) \\ &= B\big(\mathsf{id}_{\gamma_1}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1} \circ (T\ \gamma_{01})_{\#1}\big)\ (T\ \gamma_1)_{\#2} \\ &= B(\mathsf{id}_{\gamma_1}, \mathsf{id}_{(T\ \gamma_1)_{\#1}})\ (T\ \gamma_1)_{\#2} \\ &= (T\ \gamma_1)_{\#2}\end{aligned} \tag{2.3.16}$$

$$\begin{aligned}&B(\mathsf{id}_{\gamma_1}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1})\ \big(B(\gamma_{01}, \mathsf{id}_{A\ \gamma_{01}\ (T\ \gamma_0)_{\#1}})\ (T\ \gamma_0)_{\#2}\big) \\ &= B\big(\gamma_{01}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1} \circ \mathsf{id}_{A\ \gamma_{01}\ (T\ \gamma_0)_{\#1}}\big)\ (T\ \gamma_0)_{\#2} \\ &= B\big(\gamma_{01}, A\ \gamma_{01}\ (T\ \gamma_{01}^{-1})_{\#1}\big)\ (T\ \gamma_0)_{\#2}\end{aligned} \tag{2.3.17}$$

Figure 2.12: Auxiliary calculations for the proof of Proposition 2.3.13.

$$\dfrac{\mathsf{id} : \mathsf{Sub}\ \Gamma\ \Gamma \qquad \dfrac{\dfrac{\Gamma : \mathsf{NeutCon}}{e : \Gamma^- \to \Gamma} \qquad t' : \mathsf{Tm}(\Gamma, A)}{t'[\mathsf{e}] : \mathsf{Tm}(\Gamma^-, A[\mathsf{e}][\mathsf{id}^-]^{--})}}{(\mathsf{id} \,,_- \, t'[\mathsf{e}]) : \mathsf{Sub}\ \Gamma\ (\Gamma \rhd^- A[\mathsf{e}]^-)} \qquad \dfrac{\Gamma : \mathsf{NeutCon} \qquad A^- : \mathsf{Ty}\ \Gamma}{\mathsf{ee} : \mathsf{Sub}\ (\Gamma \rhd^- A[\mathsf{e}]^-)\ (\Gamma \rhd^+ A^-)}}{\dfrac{\mathsf{ee} \circ (\mathsf{id} \,,_- \, t'[\mathsf{e}]) : \mathsf{Sub}\ \Gamma\ (\Gamma \rhd^+ A^-)}{\dfrac{\mathsf{v}_0[\mathsf{ee}][\mathsf{id}_\Gamma \,,_- \, t'[\mathsf{e}]] : \mathsf{Tm}(\Gamma, A^-[\mathsf{p}_+][\mathsf{ee}][\mathsf{id}_\Gamma \,,_- \, t'[\mathsf{e}]])}{\mathsf{v}_0[\mathsf{ee}][\mathsf{id}_\Gamma \,,_- \, t'[\mathsf{e}]] : \mathsf{Tm}(\Gamma, A^-)}} \quad (2.3.19)}$$

$$\begin{aligned} A^-[\mathsf{p}_+][\mathsf{ee}][\mathsf{id}_\Gamma \,,_- \, t'[\mathsf{e}]] &= A^-[\mathsf{p}_+ \circ \mathsf{ee}][\mathsf{id}_\Gamma \,,_- \, t'[\mathsf{e}]] \\ &= A^-[\mathsf{p}_-][\mathsf{id}_\Gamma \,,_- \, t'[\mathsf{e}]] \\ &= A^-[\mathsf{id}_\Gamma] \\ &= A^- \end{aligned} \tag{2.3.19}$$

Figure 2.13: The definition given in Definition 2.3.18 has the appropriate type.

Definition 2.3.18. In any NPCwF with ee isomorphisms as in Proposition 2.3.9, define the **term negation** operation

$$\frac{\Gamma : \mathsf{NeutCon} \qquad t' : \mathsf{Tm}(\Gamma, A)}{-t' : \mathsf{Tm}(\Gamma, A^-)}$$

by

$$-t' := \mathsf{v}_0[\mathsf{ee}][\mathsf{id}_\Gamma \,,_- \, t'[\mathsf{e}]].$$

This has the correct type, by Figure 2.13

Example 2.3.20. In the category model, the operation $\mathsf{Tm}(\Gamma, A) \to \mathsf{Tm}(\Gamma, A^-)$ defined in Definition 2.3.18 has the following semantics.

PSEUDOAGDA

```
- : {Γ : NeutCon}{A : Ty Γ} → Tm(Γ,A) → Tm(Γ,A⁻)
-t' γ = t' γ
-t' γ₀₁ = A γ₀₁ (t'(γ₀₁⁻¹))
```

From the standpoint of synthetic category theory, this is a desirable outcome. If types A are supposed to represent categories, terms to represent objects, and A^- is supposed to be the opposite category of A, then we'd expect such an operation: a category and its opposite category have the same objects. In a neutral context, they do. This also addresses a significant issue with the type theory of [Nor19], namely that

there was no general way to produce terms of type A^- in that theory. If we accept a restriction to neutral contexts, then we have a way to produce terms of A^-: just convert from terms of A. It also addresses another issue from the previous section: recall that in an arbitrary context we could turn terms of type $(A \to B)^-$ into terms of $A^- \to B^-$, but the variances prevented a general method to turn terms of type $A \to B$ into terms of type $A^- \to B^-$. But now we've solved that: given $F : \mathsf{Tm}(\Gamma, A \to B)$, just consider $-F : \mathsf{Tm}(\Gamma, (A \to B)^-)$. Then we have $(-F)^- : \mathsf{Tm}(\Gamma, A^- \to B^-)$, as desired. This obeys the following law.

Proposition 2.3.21. *For $F : \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to B)$ and $t : \mathsf{Tm}(\Gamma, A^-)$ in the category model,*

$$-((-F)^- \,\$^+\, t) = F \,\$^+\, (-t).$$

Now, above we said that A and A^- have the *same terms* by virtue of this operation. We haven't proved that: given $t' : \mathsf{Tm}(\Gamma, A)$, we can negate it to get $-t' : \mathsf{Tm}(\Gamma, A^-)$ and then again: $--t' : \mathsf{Tm}(\Gamma, A)$, but we don't know yet that $--t' = t'$ as the notation might suggest. In the category model at least, it's a simple calculation from Example 2.3.20.

Proposition 2.3.22. *In the category model,*

$$-(-t') = t'$$

for every term t' in a neutral context.

Proof.

$$\begin{aligned} --t'\gamma_{01} &= A^- \,\gamma_{01}\, (-t'\,\gamma_{01}^{-1}) \\ &= A^- \,\gamma_{01}\, (A\,\gamma_{01}^{-1}\, (t'\,\gamma_{01})) \\ &= A\,(\gamma_{01} \circ \gamma_{01}^{-1})\, (t'\,\gamma_{01}) \\ &= A\,\mathsf{id}_{\gamma_1}\, (t'\,\gamma_{01}) \\ &= t'\,\gamma_{01} \end{aligned}$$

□

But what about in general? It's certainly not obvious that

$$t' = \mathsf{v}_+[\mathsf{ee}][\mathsf{id} \,,_- \, \mathsf{v}_+[\mathsf{ee}][\mathsf{id} \,,_- \, t'[\mathsf{e}]][\mathsf{e}]].$$

Indeed, it appears that we haven't asserted enough properties of ee in order to be able to prove this abstractly; we'll need to add an axiom to our NPCwF notion if we want this to be true in an arbitrary NPCwF. We could just directly assert the above equation (it holds in the category model, so there's no soundness concern), but it will be more helpful if we can find a slightly more general statement.

What we want to abstractly capture is that ee is implemented with involutive operations, so employing it twice is the same as employing it zero times. However, some gymnastics is required to state what "employing ee twice" means. Suppose we

have a composable pair of substitutions

$$\Theta \xrightarrow{\tau} \Delta \xrightarrow{\sigma} \Gamma$$

and a term $s : \mathsf{Tm}(\Theta, A[\sigma][\tau])$. Then, as usual, we can form

$$\Theta \xrightarrow{\sigma \circ \tau \,,_+\, s} \Gamma \rhd^+ A.$$

Here's the idea: if Δ and Γ are neutral, then we can use the ee maps to construct a substitution of this same shape by a different, more convoluted path: first we form

$$\Theta^- \xrightarrow{\tau^- \,,_-\, s} \Delta^- \rhd^- A[\sigma]^- \xrightarrow{\mathsf{ee}} \Delta^- \rhd^+ A[\sigma][\mathsf{e}_\Delta]^-$$

which works because s is an element of $\mathsf{Tm}(\Theta, A[\sigma][\tau]) = \mathsf{Tm}(\Theta^{--}, A[\sigma][\tau^{--}]^{--})$. If we take the variable term $\mathsf{v}_+ : \mathsf{Tm}(\Delta^- \rhd^+ A[\sigma][\mathsf{e}_\Delta]^-, A[\sigma][\mathsf{e}_\Delta][\mathsf{p}_+]^-)$ and transport it along this substitution, we get

$$\mathsf{v}_+[\mathsf{ee}][\tau^- \,,_-\, s] : \mathsf{Tm}(\Theta^-, A[\mathsf{e}_\Gamma][(\sigma \circ \tau)^-]^-)$$

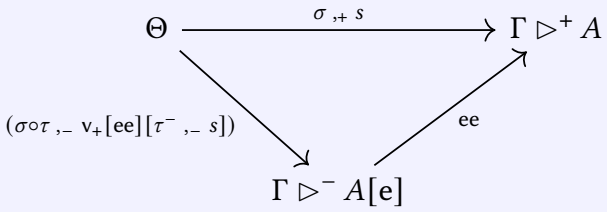
(see Figure 2.13). This has the right shape to be used as the term in a negative pairing:

$$\Theta \xrightarrow{\sigma \circ \tau \,,_-\, \mathsf{v}_+[\mathsf{ee}][\tau^- \,,_-\, s]} \Gamma \rhd^- A[\mathsf{e}].$$

At last, we can compose this with ee to get a substitution into $\Gamma \rhd^+ A$, as desired. Our assertion is that this is equal to the original $(\sigma \circ \tau \,,_+\, s)$.

Proposition 2.3.23. *In the category model, the following equation holds for all* $\Theta : \mathsf{Con}$, $\Delta, \Gamma : \mathsf{NeutCon}$, $\tau : \mathsf{Sub}\ \Theta\ \Delta$, $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$, $A : \mathsf{Ty}\ \Gamma$ *and* $s : \mathsf{Tm}(\Theta, A[\sigma][\tau])$,

$$\mathsf{ee} \circ (\sigma \circ \tau \,,_-\, \mathsf{v}_+[\mathsf{ee}][\tau^- \,,_-\, s]) = (\sigma \circ \tau \,,_+\, s).$$



As we'll see momentarily, this statement proves that $- - t' = t'$, but is general enough to be useful in other proofs too.

When introducing a new construct which operates on contexts, we have to axiomatize its interaction with the existing machinery, like p, v, q, and pairing. We can understand Proposition 2.3.9 as characterizing ee's interaction with the projection substitutions p. There's not an obvious way to axiomatize how ee interacts with v and pairing, but we have achieved that to some extent with the above law. However, for some of the proofs we want to do, that's still not quite enough. The following law, characterizing how ee interacts with the weakening operations q and q^-, will also help.

$$(\tau^- ,_- s) : \mathsf{Sub}\ \Theta^-\ (\Delta^- \rhd^- A[\sigma]^-)$$
$$\mathsf{ee} \circ (\tau^- ,_- s) : \mathsf{Sub}\ \Theta^-\ (\Delta^- \rhd^+ A[\sigma][\mathsf{e}_\Delta]^-)$$
$$\mathsf{v}_+[\mathsf{ee}][\tau^- ,_- s] : \mathsf{Tm}(\Theta^-, A[\sigma][\mathsf{e}_\Delta][\mathsf{p}_+][\mathsf{ee}][\tau^- ,_- s]^-)$$
$$A[\sigma][\mathsf{e}_\Delta][\mathsf{p}_+][\mathsf{ee}][\tau^- ,_- s]^- = A[\sigma][\mathsf{e}_\Delta][\tau]^- = A[\mathsf{e}_\Gamma][\sigma^-][\tau^-]^-$$

Figure 2.14: Types of some of the expressions appearing in Proposition 2.3.23.

Proposition 2.3.24. *For every* $\Gamma : \mathsf{NeutCon}$ *and* $A : \mathsf{Ty}\ \Gamma$ *in the category model, the following square commutes.*

$$\begin{array}{ccc} \Gamma^- \rhd^- A & \xrightarrow{\mathsf{q}^-(\mathsf{e})} & \Gamma \rhd^- A[\mathsf{e}] \\ \downarrow{\scriptstyle \mathsf{ee}} & & \downarrow{\scriptstyle \mathsf{ee}} \\ \Gamma^- \rhd^+ A[\mathsf{e}] & \xrightarrow[\mathsf{q}(\mathsf{e})]{} & \Gamma \rhd^+ A \end{array}$$

With that, we collect all these observations about ee into an abstract model notion. There are, in principle, various ways to do this. In [NA24], we asserted both ee and term-negation as primitives, including $- - t' = t'$ and a version of Corollary 2.3.28 as axioms. In the present work, we instead assert just ee with the above properties, from which we can define term-negation (as we did in Definition 2.3.18) and prove the salient properties (Corollary 2.3.28, Theorem 2.3.29, Theorem 2.3.30). We don't make any claim of completeness: there may very well be properties of the ee maps in the category model which are not provable from the presentation given here, but which are necessary for useful constructions in polarized and directed type theory. Only further work in the theory will reveal what these properties might be.

Definition 2.3.25. A (unary) **neutral-polarized CwF** is a PCwF with

- a sub-collection of **neutral contexts** $\mathsf{NeutCon} \hookrightarrow \mathsf{Con}$ and
- a sub-presheaf of **neutral types** $\mathsf{NeutTy} \hookrightarrow \mathsf{Ty}$

such that

- the empty context is neutral, $\bullet : \mathsf{NeutCon}$;
- the context extension $\Gamma \rhd^+ A$ of $\Gamma : \mathsf{NeutCon}$ by $A : \mathsf{NeutTy}\ \Gamma$ is a neutral context;
- NeutCon is closed under isomorphism; and
- for every $\Gamma : \mathsf{NeutCon}$, there is an isomorphism

$$\mathsf{e}_\Gamma : \Gamma^- \cong \Gamma$$

where

- $\mathsf{e}_\Gamma^{-1} = \mathsf{e}_\Gamma^- = \mathsf{e}_{\Gamma^-}$ and

  - $\mathsf{e}_\Gamma \circ \sigma^- = \sigma \circ \mathsf{e}_\Delta$ for any $\Delta : \mathsf{NeutCon}$ and $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$;
- for every $\Gamma : \mathsf{NeutCon}$ and $A : \mathsf{Ty}\ \Gamma$, there is an isomorphism

$$\mathsf{ee} : \Gamma \rhd^- A[\mathsf{e}] \cong \Gamma \rhd^+ A$$

  such that
  - $\mathsf{p}_+ \circ \mathsf{ee} = \mathsf{p}_-$,
  - $\mathsf{ee} \circ \mathsf{q}^-(e_\Gamma; A[\mathsf{e}]) = \mathsf{q}(e_\Gamma; A) \circ \mathsf{ee}$, and
  - for all $\Theta : \mathsf{Con}$, $\Delta, \Gamma : \mathsf{NeutCon}$, $\tau : \mathsf{Sub}\ \Theta\ \Delta$, $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$, $A : \mathsf{Ty}\ \Gamma$ and $s : \mathsf{Tm}(\Theta, A[\sigma][\tau])$,

$$\mathsf{ee} \circ (\sigma \circ \tau \mathbin{,_-} \mathsf{v}_+[\mathsf{ee}][\tau^- \mathbin{,_-} s]) = (\sigma \circ \tau \mathbin{,_+} s). \tag{2.3.26}$$

Lemma 2.3.27. *Given* $\Gamma : \mathsf{NeutCon}$ *and* $t' : \mathsf{Tm}(\Gamma, A)$ *in any NPCwF,*

$$-(t'[\mathsf{e}]) = (-t')[\mathsf{e}].$$

Proof. First, observe that $\mathsf{q}(\mathsf{e}_\Gamma)$ is the inverse of $\mathsf{q}(\mathsf{e}_{\Gamma^-})$:

$$\begin{aligned}
\mathsf{q}(\mathsf{e}_\Gamma) \circ \mathsf{q}(\mathsf{e}_{\Gamma^-}) &= (\mathsf{e}_\Gamma \circ \mathsf{p}_+ \mathbin{,_+} \mathsf{v}_+) \circ (\mathsf{e}_{\Gamma^-} \circ \mathsf{p}_+ \mathbin{,_+} \mathsf{v}_+) \\
&= (\mathsf{e}_\Gamma \circ \mathsf{p}_+ \circ (\mathsf{e}_{\Gamma^-} \circ \mathsf{p}_+ \mathbin{,_+} \mathsf{v}_+) \quad \mathbin{,_+} \quad \mathsf{v}_+[(\mathsf{e}_{\Gamma^-} \circ \mathsf{p}_+ \mathbin{,_+} \mathsf{v}_+)]) \\
&= (\mathsf{e}_\Gamma \circ \mathsf{e}_{\Gamma^-} \circ \mathsf{p}_+ \quad \mathbin{,_+} \quad \mathsf{v}_+) \\
&= (\mathsf{id}_\Gamma \circ \mathsf{p}_+ \quad \mathbin{,_+} \quad \mathsf{v}_+) \\
&= (\mathsf{p}_+ \quad \mathbin{,_+} \quad \mathsf{v}_+) \\
&= \mathsf{id}
\end{aligned}$$

and likewise for the other identity, and likewise $\mathsf{q}^-(e_\Gamma)$ and $\mathsf{q}^-(e_{\Gamma^-})$ are inverses.

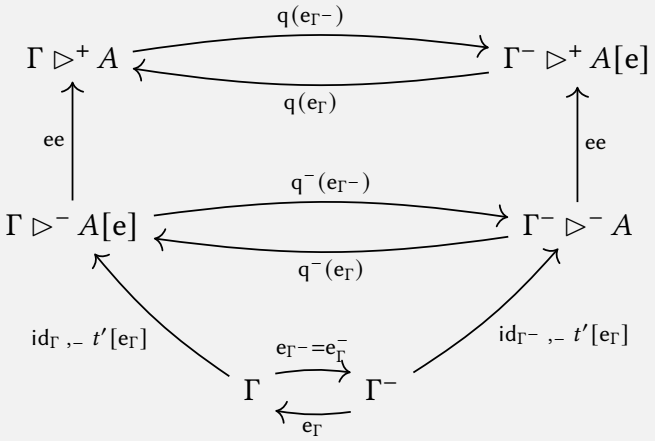
Next, for $t' : \mathsf{Tm}(\Gamma, A)$, observe that the two substitutions from Γ^- to $\Gamma^- \rhd^- A$ are equal:

$$(\mathsf{id}_{\Gamma^-} \mathbin{,_-} t') = \mathsf{q}^-(\mathsf{e}_{\Gamma^-}) \circ (\mathsf{id}_\Gamma \mathbin{,_-} t'[\mathsf{e}_\Gamma]) \circ \mathsf{e}_\Gamma$$

by:

$$\begin{aligned}
(\mathsf{id}_{\Gamma^-} \mathbin{,_-} t') &= (\mathsf{e}_{\Gamma^-} \mathbin{,_-} t'[\mathsf{e}_\Gamma]) \circ \mathsf{e}_\Gamma \\
&= (\mathsf{e}_{\Gamma^-} \circ \mathsf{p}_- \mathbin{,_-} \mathsf{v}_-) \circ (\mathsf{id}_\Gamma \mathbin{,_-} t'[\mathsf{e}_\Gamma]) \circ \mathsf{e}_\Gamma \\
&= \mathsf{q}^-(\mathsf{e}_{\Gamma^-}) \circ (\mathsf{id}_\Gamma \mathbin{,_-} t'[\mathsf{e}_\Gamma]) \circ \mathsf{e}_\Gamma.
\end{aligned}$$

This is all depicted in the following commutative diagram.



Write ($\star$) for the commutativity of the lower square (which was proved above), and ($\star\star$) for the commutativity of the upper square (which is one of the defining properties of ee given in Definition 2.3.25, plus the above observation about the q's being inverses).

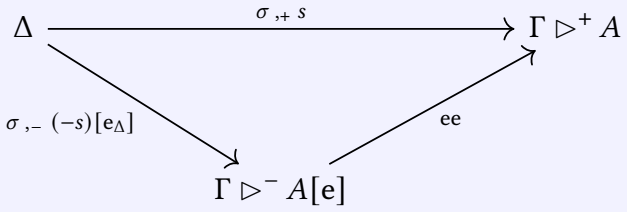
The proof of the claim is done by taking the definition of term negation (Definition 2.3.18) and moving across this diagram: $-(t'[\mathsf{e}_\Gamma])$ is the variable term in the top-right, $\mathsf{v}_+ : \mathsf{Tm}(\Gamma^- \triangleright^+ A[\mathsf{e}], A[\mathsf{e}][\mathsf{p}])$, substituted down to Γ^- along the right. On the other hand, $(-t')[\mathsf{e}_\Gamma]$ is the variable term in the top-left, $\mathsf{v}_+ : \mathsf{Tm}(\Gamma \triangleright^+ A, A[\mathsf{p}])$, substituted down the left to Γ (obtaining $-t'$), and then substituted across e_Γ into Γ. These are equal by the commutativity of the diagram:

$$\begin{aligned}
-(t'[\mathsf{e}_\Gamma]) &= \mathsf{v}_+[\mathsf{ee}][\mathsf{id}_{\Gamma^-} ,_- t'] \\
&= \mathsf{v}_+[\mathsf{ee}][\mathsf{q}^-(\mathsf{e}_{\Gamma^-})][\mathsf{id}_\Gamma ,_- t'[\mathsf{e}_\Gamma]][\mathsf{e}_\Gamma] && (\star) \\
&= \mathsf{v}_+[\mathsf{q}(\mathsf{e}_{\Gamma^-})][\mathsf{ee}][\mathsf{id}_\Gamma ,_- t'[\mathsf{e}_\Gamma]][\mathsf{e}_\Gamma] && (\star\star) \\
&= \mathsf{v}_+[\mathsf{ee}][\mathsf{id}_\Gamma ,_- t'[\mathsf{e}_\Gamma]][\mathsf{e}_\Gamma] \\
&= (-t')[\mathsf{e}_\Gamma].
\end{aligned}$$

The penultimate step uses that $\mathsf{q}(\mathsf{e}_\Gamma) := (\mathsf{e}_\Gamma \circ \mathsf{p} ,_+ \mathsf{v}_+)$, so $\mathsf{v}_+[\mathsf{q}(\mathsf{e}_\Gamma)] = \mathsf{v}_+$.

□

Corollary 2.3.28. *For any* $\Delta, \Gamma : \mathsf{NeutCon}$ *and* $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$ *and* $s : \mathsf{Tm}(\Delta, A[\sigma])$*, the following triangle commutes.*



Proof. Apply Equation 2.3.26 with $\Theta := \Delta$ and $\tau := \mathrm{id}_\Delta$. This tells us that

$$(\sigma \,,_+\, s) \quad = \quad \mathrm{ee} \circ (\sigma \,,_-\, \mathrm{v}_+[\mathrm{ee}][\mathrm{id} \,,_-\, s]).$$

The $\mathrm{v}_+[\mathrm{ee}]\mathrm{id} \,,_-\, s$ term *almost* looks like the negation of a term, but missing a substitution by e. To express it as a negation, we write

$$(\sigma \,,_+\, s) \quad = \quad \mathrm{ee} \circ (\sigma \,,_-\, \mathrm{v}_+[\mathrm{ee}][\mathrm{id} \,,_-\, s[\mathrm{e}_\Delta][\mathrm{e}_\Delta{}^-]]),$$

so we can say

$$(\sigma \,,_+\, s) \quad = \quad \mathrm{ee} \circ (\sigma \,,_-\, -(s[\mathrm{e}_\Delta])).$$

Then we can replace $-(s[\mathrm{e}_\Delta])$ by $(-s)[\mathrm{e}_\Delta]$, by Lemma 2.3.27 (since $\Delta\colon \mathsf{NeutCon}$), and we have the claim. □

Theorem 2.3.29. *In an arbitrary NPCwF, term-negation is involutive:*

$$- - t' = t'.$$

Proof.

$$\begin{aligned} - - t' &:= \mathrm{v}_+[\mathrm{ee}][\mathrm{id} \,,_-\, (-t')[\mathrm{e}]] && \text{(Defn.)} \\ &= \mathrm{v}_+[\mathrm{id} \,,_+\, t'] && \text{(Corollary 2.3.28)} \\ &= t' \end{aligned}$$

□

Theorem 2.3.30. *For any $\sigma\colon \mathsf{Sub}\ \Delta\ \Gamma$ between neutral contexts $\Delta, \Gamma\colon \mathsf{NeutCon}$, and any term $t'\colon \mathsf{Tm}(\Gamma, A)$,*

$$(-t')[\sigma] = -(t[\sigma]).$$

Proof.

$$\begin{aligned} (-t')[\sigma] &= \mathrm{v}_+[\mathrm{ee}][\mathrm{id} \,,_-\, t'[\mathrm{e}_\Gamma]][\sigma] \\ &= \mathrm{v}_+[\mathrm{ee}][\sigma \,,_-\, t'[\mathrm{e}_\Gamma \circ \sigma^-]] \\ &= \mathrm{v}_+[\mathrm{ee}][\sigma \,,_-\, t'[\sigma][\mathrm{e}_\Delta]] \\ &= \mathrm{v}_+[\mathrm{ee}][\sigma \,,_-\, (-(-(t'[\sigma])))[\mathrm{e}_\Delta]] && \text{(Theorem 2.3.29)} \\ &= \mathrm{v}_+[\sigma \,,_+\, -(t'[\sigma])] && \text{(Corollary 2.3.28)} \\ &= -(t'[\sigma]) \end{aligned}$$

□

Finally, let us note that Corollary 2.3.28 allows us to restate Proposition 2.3.13 in the following nice form.

Remark 2.3.31. In any NPCwF supporting Σ-types, we can define the Pair^- and Unpair^- functions of Proposition 2.3.13 as the composite of the other three sides of the following square.

$$\begin{array}{ccc}
\mathsf{Tm}(\Gamma, \Sigma AB) & \underset{-T \leftarrow\!\shortmid T}{\overset{T' \mapsto -T'}{\rightleftarrows}} & \mathsf{Tm}(\Gamma, (\Sigma AB)^-) \\
\mathsf{Pair} \uparrow \;\downarrow \mathsf{Unpair} & & \mathsf{Pair}^- \uparrow \;\downarrow \mathsf{Unpair}^- \\
(s' : \mathsf{Tm}(\Gamma, A)) \times \mathsf{Tm}(\Gamma, B[\mathsf{id}\ ,_+\ s']) & \underset{(-s,-t) \leftarrow\!\shortmid (s,t)}{\overset{(s',t') \mapsto (-s',-t')}{\rightleftarrows}} & (s : \mathsf{Tm}(\Gamma, A^-)) \times \mathsf{Tm}(\Gamma, B[\mathsf{id}\ ,_+\ -s])
\end{array}$$

Analogously to Definition 1.2.15, write $\mathsf{pr}_1^-(S)$ and $\mathsf{pr}_2^-(S)$ for the two components of $\mathsf{Unpair}^-(S)$:

$$\frac{S : \mathsf{Tm}(\Gamma, (\Sigma AB)^-)}{\mathsf{pr}_1^-(S) : \mathsf{Tm}(\Gamma, A^-)} \qquad \frac{S : \mathsf{Tm}(\Gamma, (\Sigma AB)^-)}{\mathsf{pr}_2^-(S) : \mathsf{Tm}(\Gamma, B[\mathsf{id}\ ,_+\ -\mathsf{pr}_1^-(S)])}$$

2.3.3 Phase Two+

But what about multi-variable functions? If we wanted to write a function of the form $A'[\mathsf{e}] \to A[\mathsf{e}] \to B$, the same trick wouldn't work: we don't know that the context $\Gamma \rhd^- A'[\mathsf{e}]$ is neutral, so we wouldn't get ee as an isomorphism between $\Gamma \rhd^- A'[\mathsf{e}] \rhd^- A[\mathsf{e} \circ \mathsf{p}_-^-]$ and $\Gamma \rhd^- A'[\mathsf{e}] \rhd^+ A[\mathsf{p}_-]$). What we need is a substitution

$$\Gamma \rhd^- A'[\mathsf{e}] \rhd^- A[\mathsf{e} \circ \mathsf{p}_-^-] \xrightarrow{\mathsf{ee}_2} \Gamma \rhd^+ A' \rhd^+ A[\mathsf{p}_{+,A'}].$$

The category model supports this, so long as Γ is neutral and, crucially, both A' and A are types in Γ (i.e. A doesn't depend on a variable of type A'). The object part of this functor continues to be the identity, and the morphism part is analogous to the definition of ee given above:

$$(\gamma_{01}, \mathsf{x}', \mathsf{x}) \quad \mapsto \quad (\gamma_{01}, A'\ \gamma_{01}^{-1}\ \mathsf{x}', A\ \gamma_{01}^{-1}\ \mathsf{x}).$$

This satisfies the analogous properties to the ee introduced above. For instance, the following square commutes.

$$\begin{array}{ccc}
\Gamma \rhd^- A'[\mathsf{e}] \rhd^- A[\mathsf{e} \circ \mathsf{p}_-^-] & \xrightarrow{\mathsf{ee}\ \{A',A\}} & \Gamma \rhd^+ A' \rhd^+ A[\mathsf{p}_+] \\
\mathsf{p}_- \downarrow & & \downarrow \mathsf{p}_+ \\
\Gamma \rhd^- A'[\mathsf{e}] & \xrightarrow[\mathsf{ee}\ \{A'\}]{} & \Gamma \rhd^+ A'
\end{array} \tag{2.3.32}$$

As here, we'll begin view ee as a family of isomorphisms indexed over lists of types: the binary version on the top for A', A, and the unary version on the bottom for just A'; we'll make this precise shortly.

We can also perform the analogous construction to Proposition 2.3.23: as there, suppose $\tau\colon \mathsf{Sub}\ \Theta\ \Delta$ and $\sigma\colon \mathsf{Sub}\ \Delta\ \Gamma$ with Δ, Γ neutral, and $s'\colon \mathsf{Tm}(\Theta, A'[\sigma][\tau])$, $s\colon \mathsf{Tm}(\Theta, A[\sigma][\tau])$. Then we can take the positive route from Θ to $\Gamma \rhd^+ A' \rhd^+ A[\mathsf{p}]$, i.e. form the substitution

$$(\sigma \circ \tau \mathbin{,_+} s' \mathbin{,_+} s) \quad : \quad \mathsf{Sub}\ \Theta\ (\Gamma \rhd^+ A' \rhd^+ A[\mathsf{p}]).$$

But we can also take the route involving the binary ee: observe that we have

$$\begin{array}{ccc} \Theta^- & \xrightarrow{(\tau^- \mathbin{,_-} s' \mathbin{,_-} s)} & \Delta^- \rhd^- A'[\sigma]^- \rhd^- A[\sigma][\mathsf{p}_+]^- \\ & & \Big\downarrow{\scriptstyle \mathsf{ee}} \\ & & \Delta^- \rhd^+ A'[\sigma][\mathsf{e}]^- \rhd^+ A[\sigma][\mathsf{e}][\mathsf{p}_+] \end{array}$$

where the ee on the right is the one for the neutral context Δ^- and types $A'[\sigma][\mathsf{e}_\Delta]^-$ and $A[\sigma][\mathsf{e}_\Delta]^-$ (check that $A[\sigma][\mathsf{e}_\Delta]^-[\mathsf{e}_{\Delta^-}][\mathsf{p}^-_-] = A[\sigma][\mathsf{p}_+]^-$). Then we can form the terms

$$\begin{aligned} \mathsf{v}_1[\mathsf{ee}][\tau^- \mathbin{,_-} s' \mathbin{,_-} s] \quad &: \mathsf{Tm}(\Theta^-, A'[\mathsf{ee}_\Gamma][(\sigma \circ \tau)^-]^-) \\ \mathsf{v}_0[\mathsf{ee}][\tau^- \mathbin{,_-} s' \mathbin{,_-} s] \quad &: \mathsf{Tm}(\Theta^-, A[\mathsf{ee}_\Gamma][(\sigma \circ \tau)^-]^-) \end{aligned}$$

and thus it makes sense to assert

$$(\sigma \circ \tau \mathbin{,_+} s' \mathbin{,_+} s) \quad = \quad \mathsf{ee} \circ (\sigma \circ \tau \mathbin{,_-} \mathsf{v}_1[\mathsf{ee}][\tau^- \mathbin{,_-} s' \mathbin{,_-} s] \mathbin{,_-} \mathsf{v}_0[\mathsf{ee}][\tau^- \mathbin{,_-} s' \mathbin{,_-} s]) \tag{2.3.33}$$

With this binary ee in hand, we can reason effectively about 2-variable functions.

Proposition 2.3.34. *Given $A, B, C\colon \mathsf{Ty}\ \Gamma$ for $\Gamma\colon \mathsf{NeutCon}$ with ee_2 as described above, there is an isomorphism*

$$\mathsf{Tm}(\Gamma, (A \times B)[\mathsf{e}] \to C) \quad \cong \quad \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to B[\mathsf{e}] \to C).$$

Proof. We have the chain of isomorphisms

$$\begin{aligned} &\mathsf{Tm}(\Gamma, (A \times B)[\mathsf{e}] \to C) \\ &\cong \mathsf{Tm}(\Gamma \rhd^+ (A \times B), C[\mathsf{p}_+]) && (\mathsf{app}^+ \ \&\ \mathsf{lam}^+) \\ &\cong \mathsf{Tm}(\Gamma \rhd^+ A \rhd^+ B[\mathsf{p}_+], C[\mathsf{p}_+ \circ \mathsf{pair}]) && (\_[\mathsf{pair}] \ \&\ \Sigma\text{-}\mathsf{elim}) \\ &\cong \mathsf{Tm}(\Gamma \rhd^- A[\mathsf{e}] \rhd^- B[\mathsf{e} \circ \mathsf{p}^-_-], C[\mathsf{p}_+][\mathsf{pair}][\mathsf{ee}]) && (\_[\mathsf{ee}] \text{ and } \_[\mathsf{ee}^{-1}]) \\ &= \mathsf{Tm}(\Gamma \rhd^- A[\mathsf{e}] \rhd^- B[\mathsf{e} \circ \mathsf{p}^-_-], C[\mathsf{p}_- \circ \mathsf{p}_-]) && (\star) \\ &\cong \mathsf{Tm}(\Gamma \rhd^- A[\mathsf{e}], (B[\mathsf{e}] \to C)[\mathsf{p}_-]) && (\mathsf{lam}\ \&\ \mathsf{app}) \\ &\cong \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to B[\mathsf{e}] \to C) && (\mathsf{lam}\ \&\ \mathsf{app}) \end{aligned}$$

where (⋆) comes from the equality

$$\begin{aligned} \mathsf{p}_{+,A\times B} \circ \mathsf{pair} \circ (\mathsf{ee}\ \{A,B\}) &= \mathsf{p}_{+,A} \circ \mathsf{p}_{+,B[\mathsf{p}_+]} \circ (\mathsf{ee}\ \{A,B\}) && \text{(Definition 2.2.16)} \\ &= \mathsf{p}_{+,A} \circ (\mathsf{ee}\ \{A\}) \circ \mathsf{p}_{-,B[\mathsf{e}\circ\mathsf{p}_-^-]} && \text{(Equation 2.3.32)} \\ &= \mathsf{p}_{-,A[\mathsf{e}]} \circ \mathsf{p}_{-,B[\mathsf{e}\circ\mathsf{p}_-^-]} && \text{(Proposition 2.3.9)} \end{aligned}$$

□

So, at least for the purposes of writing *non-dependent* functions, we have an analogous construction to Definition 2.3.10 for 2-variable functions: we can write a function of shape $A[\mathsf{e}] \to B[\mathsf{e}] \to C$ by supplying a term $c : \mathsf{Tm}(\Gamma \triangleright^+ A \triangleright^+ B[\mathsf{p}], C[\mathsf{p}_+ \circ \mathsf{p}_+])$, substituting it by the binary ee, and then applying the original lambda abstraction operator twice. Note that we *do* rely on both A and B being types in Γ here: we (apparently) don't have a similar ee for when $B : \mathsf{Ty}(\Gamma \triangleright^+ A)$, and so this technique doesn't permit writing dependent functions—if we want to write something of the form $\Pi(A, \Pi(B, C))$, then we have to deal with the variance of B's dependence on A.

What about 3-variable functions? And 4-variable functions? We make the following definitions to generalize the above specification of ee to arbitrary arity.

Definition 2.3.35. A (positive) **flat telescope** in context Γ is a list of types in Γ:

$$\mathsf{Tel}^\flat\ \Gamma := \mathsf{List}(\mathsf{Ty}\ \Gamma).$$

Rather than the customary notation for lists, where new elements are added on the *left* by an operator cons, we'll instead write them with new elements added on the *right* by an operator snoc, i.e. $\mathsf{Tel}^\flat\ \Gamma$ is generated by

$$\begin{aligned} \mathsf{nil} &: \mathsf{Tel}^\flat\ \Gamma \\ \mathsf{snoc} &: \mathsf{Tel}^\flat\ \Gamma \to \mathsf{Ty}\ \Gamma \to \mathsf{Tel}^\flat\ \Gamma. \end{aligned}$$

Given $\mathsf{Ts} : \mathsf{Tel}^\flat\ \Gamma$ and $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$, write $\mathsf{Ts}[\sigma] : \mathsf{Tel}^\flat\ \Delta$ for the result of substituting every type in Ts by σ:

$$\begin{aligned} \mathsf{nil}[\sigma] &:= \mathsf{nil} \\ (\mathsf{snoc}\ \mathsf{Ts}\ \mathsf{T})[\sigma] &:= \mathsf{snoc}\ \mathsf{Ts}[\sigma]\ \mathsf{T}[\sigma]. \end{aligned}$$

Likewise, define Ts^- to be the element-wise negation of the types in Ts.

Definition 2.3.36. The context extension operator $\triangleright^+$ of any CwF can be lifted to an operation extending a context Γ by a *flat telescope* in Γ.

PSEUDOAGDA

```
_▷⁺_ : (Γ : Con) ⟶ List(Ty Γ) ⟶ Con
p₊* : {Γ : Con} ⟶ (Ts : List(Ty Γ)) ⟶ Sub (Γ ▷⁺ Ts) Γ
Γ ▷⁺ nil = Γ
Γ ▷⁺ (snoc Ts T) = (Γ ▷⁺ Ts) ▷⁺ T[ p₊* (Ts) ]
p₊* nil = id Γ
p₊* (snoc Ts T) = (p₊* Ts) ∘ (p₊ {Γ=Γ▷⁺Ts} {A=T[ p₊* (Ts) ]})
```

Definition 2.3.37. Given any Γ and $\mathsf{Ts} : \mathsf{Tel}^\flat(\Gamma^-)$, we can define the *negative* extension of Γ by Ts, as follows.

PSEUDOAGDA

```
_▷⁻_ : (Γ : Con) ⟶ List(Ty Γ⁻) ⟶ Con
p₋* : {Γ : Con} ⟶ (Ts : List(Ty Γ⁻)) ⟶ Sub (Γ ▷⁻ Ts) Γ
Γ ▷⁻ nil = Γ
Γ ▷⁻ (snoc Ts T) = (Γ ▷⁻ Ts) ▷⁻ T[ (p₋*(Ts))⁻ ]
p₋* nil = id Γ
p₋* (snoc Ts T) = (p₋* Ts) ∘ (p₋ {Γ=Γ▷⁻Ts} {A=T[ (p₋* (Ts))⁻ ]})
```

Definition 2.3.38. Given $\mathsf{Ts} : \mathsf{Tel}^\flat\ \Gamma$, write $\mathsf{Tms}(\Gamma, \mathsf{Ts})$ for the **set of terms of** Ts:

$$\begin{aligned} \mathsf{Tms}(\Gamma, \mathsf{nil}) &:= \{\star\} \\ \mathsf{Tms}(\Gamma, \mathsf{snoc\ Ts\ T}) &:= \mathsf{Tms}(\Gamma, \mathsf{Ts}) \times \mathsf{Tm}(\Gamma, \mathsf{T}). \end{aligned}$$

Given $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$ and $\vec{r} : \mathsf{Tms}(\Gamma, \mathsf{Ts})$, define $\vec{r}[\sigma] : \mathsf{Tms}(\Delta, \mathsf{Ts}[\sigma])$ by element-wise substitution by σ.

When $n + 1 = |\mathsf{Ts}|$, write $\vec{\mathsf{v}_+}$ for

$$(\mathsf{v}_n, \mathsf{v}_{n-1}, \ldots, \mathsf{v}_0) : \mathsf{Tms}(\Gamma \triangleright^+ \mathsf{Ts}, \mathsf{Ts}[\mathsf{p}_+^*]).$$

Definition 2.3.39. Given $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$, $\mathsf{Ts}' : \mathsf{Tel}^\flat\ \Gamma$, and $\vec{s} : \mathsf{Tms}(\Delta, \mathsf{Ts}'[\sigma])$, define

$$(\sigma \,,_+\, \vec{s}) \quad : \mathsf{Sub}\ \Delta\ (\Gamma \triangleright^+ \mathsf{Ts}')$$

by

$$(\sigma \,,_+\, (s_1, \ldots, s_n)) := (\sigma \,,_+\, s_1 \,,_+\, \ldots \,,_+\, s_n).$$

Similarly, given $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$, $\mathsf{Ts} : \mathsf{Tel}^\flat(\Gamma^-)$, and $\vec{t} : \mathsf{Tms}(\Delta^-, \mathsf{Ts}[\sigma^-]^-)$, define

$$(\sigma \,,_-\, \vec{t}) : \mathsf{Sub}\ \Delta\ (\Gamma \triangleright^- \mathsf{Ts}).$$

Proposition 2.3.40 (Ladder Isomorphisms in Category Model). *For any groupoid* Γ *and any list* $\mathsf{Ts} : \mathsf{Tel}^\flat\ \Gamma$*, there is an isomorphism in the category model*

$$\mathsf{ee}\ \{\mathsf{Ts}\} \quad : \quad \Gamma \rhd^- \mathsf{Ts}[\mathsf{e}] \quad \cong \quad \Gamma \rhd^+ \mathsf{Ts}$$

such that

- $\mathsf{ee}\ \{\mathsf{nil}\} = \mathsf{id}\ \Gamma$
- *for all* $\mathsf{T} : \mathsf{Ty}\ \Gamma$*, the following square commutes.*

$$\begin{array}{ccc} \Gamma \rhd^- (\mathsf{snoc}\ \mathsf{Ts}\ \mathsf{T})[\mathsf{e}] & \xrightarrow{\mathsf{ee}\ \{\mathsf{snoc}\ \mathsf{Ts}\ \mathsf{T}\}} & \Gamma \rhd^+ (\mathsf{snoc}\ \mathsf{Ts}\ \mathsf{T}) \\ \downarrow{\scriptstyle \mathsf{p}_{-,\mathsf{T}[\mathsf{e}]}} & & \downarrow{\scriptstyle \mathsf{p}_{+,\mathsf{T}}} \\ \Gamma \rhd^- \mathsf{Ts} & \xrightarrow[\mathsf{ee}\ \{\mathsf{Ts}\}]{} & \Gamma \rhd^+ \mathsf{Ts} \end{array} \tag{2.3.41}$$

- *for every* Ts*, the following square commutes;*

$$\begin{array}{ccc} \Gamma^- \rhd^- \mathsf{Ts} & \xrightarrow{\mathsf{q}^-(\mathsf{e};\mathsf{Ts})} & \Gamma \rhd^- \mathsf{Ts}[\mathsf{e}] \\ \downarrow{\scriptstyle \mathsf{ee}\ \{\mathsf{Ts}[\mathsf{e}]\}} & & \downarrow{\scriptstyle \mathsf{ee}\ \{\mathsf{Ts}\}} \\ \Gamma^- \rhd^+ \mathsf{Ts}[\mathsf{e}] & \xrightarrow[\mathsf{q}(\mathsf{e};\mathsf{Ts})]{} & \Gamma \rhd^+ \mathsf{Ts} \end{array} \tag{2.3.42}$$

- *for all* $\Theta : \mathsf{Con}$, $\Delta, \Gamma : \mathsf{NeutCon}$, $\tau : \mathsf{Sub}\ \ \Theta\ \ \Delta$, $\sigma : \mathsf{Sub}\ \ \Delta\ \ \Gamma$, *and* $\vec{s} : \mathsf{Tms}(\Theta, \mathsf{Ts}[\sigma][\tau])$,

$$(\mathsf{ee}\ \{\mathsf{Ts}\}) \circ (\sigma \circ \tau \mathbin{,_-} \vec{\mathsf{v}}_+[\mathsf{ee}\ \{(\mathsf{Ts}[\sigma][\mathsf{e}])^-\}][\tau^- \mathbin{,_-} \vec{s}]) = (\sigma \circ \tau \mathbin{,_+} \vec{s}). \tag{2.3.43}$$

We call these the "ladder isomorphisms" in reference to Equation 2.3.41: the ee's are the rungs of the ladder, with the positive and negative p's as the sides; this was depicted as a triangle in Proposition 2.3.9 because the bottom rung (ee {nil}) is the identity morphism on Γ. It's unclear whether the other properties, Equation 2.3.42 and Equation 2.3.43, are practically useful outside the unary case discussed above; we include them simply for the sake of uniformity.

At last, this supplies us with our full notion of NPCwF.

Definition 2.3.44. A **neutral-polarized CwF** is a PCwF with

- a sub-collection of **neutral contexts** NeutCon $\hookrightarrow$ Con and
- a sub-presheaf of **neutral types** NeutTy $\hookrightarrow$ Ty

such that

- the empty context is neutral, $\bullet$: NeutCon;
- the context extension $\Gamma \rhd^+ A$ of Γ : NeutCon by A: NeutTy Γ is a neutral context;
- NeutCon is closed under isomorphism; and

- for every $\Gamma : \mathsf{NeutCon}$, there is an isomorphism

$$\mathsf{e}_\Gamma : \Gamma^- \cong \Gamma$$

  where
  - $\mathsf{e}_\Gamma^{-1} = \mathsf{e}_\Gamma^- = \mathsf{e}_{\Gamma^-}$ and
  - $\mathsf{e}_\Gamma \circ \sigma^- = \sigma \circ \mathsf{e}_\Delta$ for any $\Delta : \mathsf{NeutCon}$ and $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$;
- for every $\Gamma : \mathsf{NeutCon}$ and every $\mathsf{Ts} : \mathsf{Tel}^\flat\ \Gamma$, an isomorphism

$$\mathsf{ee}\ \{\mathsf{Ts}\} \quad : \quad \Gamma \rhd^- \mathsf{Ts}[\mathsf{e}] \quad \cong \quad \Gamma \rhd^+ \mathsf{Ts}$$

  such that
  - $\mathsf{ee}\ \{\mathsf{nil}\} = \mathsf{id}\ \Gamma$;
  - for all $\mathsf{T} : \mathsf{Ty}\ \Gamma$, the following square commutes;

$$\begin{array}{ccc} \Gamma \rhd^- (\mathsf{snoc}\ \mathsf{Ts}\ \mathsf{T})[\mathsf{e}] & \xrightarrow{\mathsf{ee}\ \{\mathsf{snoc}\ \mathsf{Ts}\ \mathsf{T}\}} & \Gamma \rhd^+ (\mathsf{snoc}\ \mathsf{Ts}\ \mathsf{T}) \\ \downarrow{\scriptstyle \mathsf{p}_{-,\mathsf{T}[\mathsf{e}]}} & & \downarrow{\scriptstyle \mathsf{p}_{+,\mathsf{T}}} \\ \Gamma \rhd^- \mathsf{Ts} & \xrightarrow[\mathsf{ee}\ \{\mathsf{Ts}\}]{} & \Gamma \rhd^+ \mathsf{Ts} \end{array} \tag{2.3.45}$$

  - for every Ts, the following square commutes;

$$\begin{array}{ccc} \Gamma^- \rhd^- \mathsf{Ts} & \xrightarrow{\mathsf{q}^-(\mathsf{e};\mathsf{Ts})} & \Gamma \rhd^- \mathsf{Ts}[\mathsf{e}] \\ \downarrow{\scriptstyle \mathsf{ee}\ \{\mathsf{Ts}[\mathsf{e}]\}} & & \downarrow{\scriptstyle \mathsf{ee}\ \{\mathsf{Ts}\}} \\ \Gamma^- \rhd^+ \mathsf{Ts}[\mathsf{e}] & \xrightarrow[\mathsf{q}(\mathsf{e};\mathsf{Ts})]{} & \Gamma \rhd^+ \mathsf{Ts} \end{array} \tag{2.3.46}$$

  - for all $\Theta : \mathsf{Con}$, $\Delta, \Gamma : \mathsf{NeutCon}$, $\tau : \mathsf{Sub}\ \Theta\ \Delta$, $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$, and $\vec{s} : \mathsf{Tms}(\Theta, \mathsf{Ts}[\sigma][\tau])$,

$$(\mathsf{ee}\ \{\mathsf{Ts}\}) \circ (\sigma \circ \tau ,_- \vec{\mathsf{v}}_+[\mathsf{ee}\ \{(\mathsf{Ts}[\sigma][\mathsf{e}])^-\}][\tau^- ,_- \vec{s}]) = (\sigma \circ \tau ,_+ \vec{s}). \tag{2.3.47}$$

There's one important caveat with this definition: *it's not a GAT*. The issue is that there aren't finitely-many components: in order to introduce the ladder isomorphisms (and assert their properties), we must quantify over flat telescopes, which are not an element of a previous sort but *lists* of elements of a previous sort. There's no machinery in the definition of 'GAT' to be able to do this; we must pass to *indexed GATs/QIIT signatures*. We can do it with just one metatheoretic set—the natural numbers—by quantifying over the *length* of the flat telescope, but for full generality it seems we *do* need at least that. Perhaps some clever encoding as a GAT could exist, but by inspection it certainly seems impossible. Of course, this is a theoretical rather than practical limitation: the notion of 'NPCwF' with the ladder isomorphisms specified for

only flat telescopes of length less than, say, one million *is* a GAT. For practically all the constructions in the present work, the unary NPCwF notion given in Definition 2.3.25 is adequate.[15]

[15] In [NA24], we didn't even contemplate binary ee.

Chapter 3

Directed Type Theory

We now turn to the central topic of our study: *hom-types*. These types are to furnish a directed notion of equality, a directed analogue of the identity types characteristic of (undirected) Martin-Löf Type Theory. To guide our analysis, it is helpful to consider what the judgment $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t'))$ is supposed to *mean*. In particular, we want to interpret our theory's *rejection* of the following principles.

- **Symmetry**: as we've discussed, *undirected* type theories are characterized by the symmetry of their identity-types: from a term of $\mathsf{Id}(t, t')$, obtain a term of $\mathsf{Id}(t', t)$. We distinguish our theories (and other theories) as *directed* in rejecting symmetry: $\mathsf{Hom}(t, t')$ could be inhabited, but $\mathsf{Hom}(t', t)$ could be *uninhabited*. Any interpretation of what hom-types *mean* will need to explain this fact.
- **Uniqueness of Homs Principle**: how do we make sense of the fact that, in models like the category model, the directed analogue of the *uniqueness of identity principle* fails? That is, what does it mean for the hom type $\mathsf{Hom}(t, t')$ to contain multiple, distinct inhabitants?

As with undirected type theory, we can give several compelling answers to this question, which will help illuminate the shape of our eventual theory

- **Category interpretation**: our predominant understanding so far has been that we wish to endow our types with the synthetic structure of a category. The terms furnish the synthetic *objects*, and we must complete the picture by giving synthetic *hom-sets* between the objects. As discussed, this is simply the directed analogue of the well-established interpretation of types in undirected type theory as representing (higher) groupoids.
  In this view, the failure of UHP amounts to the fact that *not all categories are preorders*; that is, a given hom-set in a category can contain numerous elements—distinct, parallel arrows. It's no wonder, then, that the preorder model *will* validate UHP, the same way that the setoid model validates UIP.
- **Homotopy interpretation**: the keen insight of Voevodsky was that the *homotopy types* of algebraic topology could literally be *types* in the computer-scientific

sense; upon this rock, homotopy type theorists have built their discipline. Though it takes considerable work to precisely interpret type theory in homotopy spaces, the basic intuition is straightforward: the terms of a type are the *points* of a space, the terms of type $\mathsf{Id}(t, t')$ are *paths* in the space from t to t', paths between *those* paths are *homotopies*, and so on into arbitrary dimensions.
With directed type theory, we simply add a *direction* onto these paths and homotopies: a *directed path* f, i.e. a term of type $\mathsf{Hom}(t, t')$, will take you from t to t', but *there might not be a path back*. Our rejection of symmetry opens the possibility that $\mathsf{Hom}(t, t')$ might be inhabited but $\mathsf{Hom}(t', t)$ uninhabited. Instead of giving a type theory corresponding to homotopy spaces, we are instead giving a type theory corresponding to *directed homotopy spaces* (see e.g.).
UHP, in this view, is the assertion that all parallel directed paths are homotopic, i.e. that there are no "holes" in our directed spaces. In rejecting this principle, we allow our directed homotopy spaces to have holes in them, where the meaning of "having a hole" is formally cashed out in the existence of distinct directed paths with the same start-point and end-point, which "go around" the hole on different sides.

- **Computational interpretation**: we can understand directed paths, i.e. hom-terms, as reflecting the temporal evolution of a computer process: a term f of $\mathsf{Hom}(t, t')$ means that the computation could *start* at t and *result* in t', by way of the evaluation trace f. In much the same way that undirected identity types make metatheoretic judgmental equality tangible within the object theory—and the *intensional/extensional* distinction indicates how *exactly* judgmental equality is so rendered—we can understand directed hom-types as internalizing the *reduction* relation of the metatheory.
  Our denial of UHP has an intriguing application: the analysis of concurrency. As thoroughly treated in , careful steps must be taken when dealing with concurrent computer processes to ensure that processes don't attempt to simultaneously modify the same memory location, lest the computational results become unpredictable; this is managed with a system of "locking" and "freeing" of resources. A "hole" in our directed homotopy space corresponds to a *deadlock*, when two processes lock the same resource, preventing either from utilizing it. Any *successful* evaluation trace must *go around* the hole, i.e. schedule operations between the processors that avoids deadlocks. The existence of holes, i.e. the fact that our directed homotopy spaces have nonzero dimension, corresponds to the existence of multiple concurrent processors whose operation can be scheduled in different ways; our denial of UHP thus represents the presence of concurrency in our evaluation system.
- **Logical-Temporal interpretation**: More abstractly,[1] we can understand hom-types as encoding an equality relation which is modulated by time: an inhabitant of $\mathsf{Hom}(t, t')$ is a witness to the assertion that t *becomes* t'. Such a relation is inherently directed, because *time itself is directed*: t computes to t' over time, and to go back would involve *turning time back*. Just like how a directed path is a *path with direction*, we can think of a directed equality as an *equality with a temporal order*: it is not just the case that $7 + 5$ *is* 12, but rather that $7 + 5$ *becomes*

[1] Or, perhaps it would be better to say, more *philosophically*.

12 *through time*.

Though the category interpretation is our primary understanding in the present work, directed type theory should properly engage with all of these: not only do each of these perspectives provide a different way of intuiting what each principle of directed type theory means, but also each of them potentially serves as a source of new ideas to be transported to the others, utilizing directed type theory as the *rosetta stone* between them.

Let us conclude this introduction with the following observation: in the above interpretations, we were ambiguous about whether undirected equality was *stronger* or *weaker* than directed equality. In the case of categories and groupoids, it seems that the directed notion is weaker: undirected identity is interpreted as *isomorphism*, which consists of directed equality, i.e. morphisms, in either direction. In the other interpretations, directed equality seemed to be *more* than undirected equality: directed equality is equality, but with a *direction*, an *orientation*, a *temporal order*. In the present work, we won't embark on a precise comparison of these two possible relationships between identity types and hom-types—identity as the *core* of directed equality versus identity as the *closure* of directed equality—but we will speculate somewhat in the final chapter about what such a treatment might look like. In particular, we'll find that the category model allows us to take both the *core* A^0 and the *localization* A^* of a type A, which capture these two notions of identity generated by the system of hom-types on the directed type A. Depending on the intended application, one might want directed type theory with one notion of identity or the other, or both.

3.1 Directed Equality Types

Let's develop hom-types in the category model. Once again, our methodology will be to (1) examine the groupoid model's definition of the undirected notion, in this case identity types; (2) to solve the "polarity problem", i.e. appropriately annotate the contexts, types, etc. with polarities so that the definition makes sense in the category model (possibly in a neutral context); and then (3) articulate the resulting structure as an extension of the GAT $\mathfrak{CwF}$.

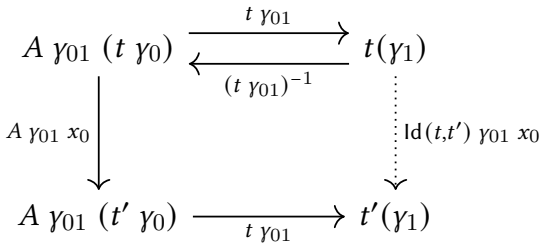
The formation rule of identity types gives rise to a relatively simple polarity problem. Recall that the type $\mathsf{Id}(t, t')$ is interpreted in the groupoid model as follows.

$$
\begin{aligned}
&\mathsf{Id} && : \{\Gamma : \mathsf{Con}\}\{A : \mathsf{Ty}\ \Gamma\}(t\ t' : \mathsf{Tm}(\Gamma, A)) \to \mathsf{Ty}\ \Gamma \\
&\mathsf{Id}(t, t')\ \gamma && := (A\ \gamma)\ [t(\gamma), t'(\gamma)] \\
&\mathsf{Id}(t, t')\ \gamma_{01} && : (A\ \gamma_0)\ [t(\gamma_0), t'(\gamma_0)] \Rightarrow (A\ \gamma_1)\ [t(\gamma_1), t'(\gamma_1)] \\
&\mathsf{Id}(t, t')\ \gamma_{01}\ x_0 && := t'(\gamma_{01}) \circ A\ \gamma_{01}\ x_0 \circ (t\ \gamma_{01})^{-1}
\end{aligned}
$$

For the object part, the set $(A\ \gamma)\ [t(\gamma), t'(\gamma)]$ is regarded as a *discrete groupoid*, i.e. the only morphisms are the identity morphism on each object. This doesn't need to change when we pass to the category model.[2] As we see, there is only one use of the inverse

[2] Though, since discrete categories are inherently groupoids, hom-types in the category model will always end up being neutral types themselves. We'll make this explicit later in this section.

operation in this definition: in the morphism part, we must invert the morphism $t(\gamma_{01})$ so that we can navigate around this square (in $A(\gamma_1)$)



and define $\mathsf{Id}(t, t')\ \gamma_{01}\ x_0$ as required on the right. This is a *shallow* assumption of invertibility: we don't use the fact that Γ is a groupoid, only that $A(\gamma_1)$ is. Accordingly, we'll make this suitable for the category model using the latter's shallow polarity: if we just require the term t to be of type A^- instead of A, then its morphism part $t(\gamma_{01})$ will go the desired direction.

We therefore follow [Nor19] and adopt the following Hom-formation rule.

Definition 3.1.1. Define the Hom-**formation** operation of the category model, which is expressed in rule form as

$$\frac{\Gamma : \mathsf{Con} \quad t : \mathsf{Tm}(\Gamma, A^-) \quad t' : \mathsf{Tm}(\Gamma, A)}{\mathsf{Hom}_A(t, t') : \mathsf{Ty}\ \Gamma} \qquad \text{(Hom formation)}$$

by the following.

PSEUDOAGDA

```
Hom : {Γ : Con}{A : Ty Γ} → Tm(Γ, A⁻) → Tm(Γ, A) → Ty Γ

Hom(t,t') : |Γ| → Cat
| Hom(t,t') γ | := (A γ) [ t γ , t' γ ]
(Hom(t,t') γ) [ i , j ] := (i = j)

Hom(t,t') : (γ₀₁ : Γ [ γ₀ , γ₁ ]) → (Hom(t,t') γ₀) ⇒ (Hom(t,t') γ₁)
Hom(t,t') γ₀₁ f₀ := t'(γ₀₁) ∘ A γ₀₁ f₀ ∘ t(γ₀₁)
```

Remark 3.1.2. Note that the hom-types in the category model are defined as a family of *discrete* categories. So, if a context contains variables of hom-types, e.g.

$$\Gamma \rhd^+ A \rhd \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0) \qquad \text{for some } t : \mathsf{Tm}(\Gamma, A^-)$$

then the morphisms of this category will include meta-theoretic identity terms:

$$\begin{aligned}&\big(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0)\big)\ [\ (\gamma_0, a_0, x_0),\ (\gamma_1, a_1, x_1)\ ] \\ &= (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \times (a_{01} : (A\ \gamma_1)\ [A\ \gamma_{01}\ a_0, a_1]) \times (a_{01} \circ A\ \gamma_{01}\ x_0 \circ t(\gamma_{01}) = x_1).\end{aligned}$$

It will often be convenient to "singleton contract" such expressions to remove the occurrence of the identity. That is, instead of taking morphisms to be triples

$(\gamma_{01}, a_{01}, x_{01})$ where x_{01} is a proof in the metatheory that $a_{01} \circ A\ \gamma_{01}\ x_0 \circ t(\gamma_{01}) = x_1$, we'll instead understand the morphisms in the category to be *pairs*

$$(\gamma_{01}, a_{01}) \quad : \big(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0)\big)[\ (\gamma_0, a_0, x_0),\ (\gamma_1, a_1, a_{01} \circ A\ \gamma_{01}\ x_0 \circ t(\gamma_{01}))\ ].$$

Therefore a type $M : \mathsf{Ty}(\Gamma \rhd^+ A \rhd \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0))$ will have shape

$$\begin{aligned} M_{\#1} &: (\gamma : |\Gamma|) \to (a : |A\ \gamma|) \to (x : (A\ \gamma)\ [t\ \gamma, a]) \to \mathrm{Cat} \\ M_{\#2} &: (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \to (a_{01} : (A\ \gamma_1)[A\ \gamma_{01}\ a_0, a_1]) \to \\ &\qquad \big(M_{\#1}(\gamma_0, a_0, x_0) \Rightarrow M_{\#1}(\gamma_1, a_1, a_{01} \circ A\ \gamma_{01}\ x_0 \circ t(\gamma_{01}))\big) \end{aligned}$$

And a term $J : \mathsf{Tm}(\Gamma \rhd^+ A \rhd \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0), M)$ will have shape

$$\begin{aligned} J_{\#1} &: (\gamma : |\Gamma|) \to (a : |A\ \gamma|) \to (x : (A\ \gamma)\ [t\ \gamma, a]) \to |M(\gamma, a, x)| \\ J_{\#2} &: (\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]) \to (a_{01} : (A\ \gamma_1)[A\ \gamma_{01}\ a_0, a_1]) \to \\ &\qquad (M(\gamma_1, a_1, a_{01} \circ A\ \gamma_{01}\ x_0 \circ t(\gamma_{01}))) \ [\\ &\qquad\quad M\ (\gamma_{01}, a_{01})\ (J_{\#1}(\gamma_0, a_0, x_0)), \\ &\qquad\quad J_{\#1}(\gamma_1, a_1, a_{01} \circ A\ \gamma_{01}\ x_0 \circ t(\gamma_{01})) \\ &\qquad]. \end{aligned}$$

Besides being required by our semantics, these polarity annotations—negative for the domain, positive for the codomain—are desirable for our syntax. For one, they seem to provide a guarantee that our directed type theory will not "collapse" into undirected type theory (as described in the introduction to chapter 2): we know that a term $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}_A(t, t'))$ cannot generally be turned into a term $f^{-1} : \mathsf{Tm}(\Gamma, \mathsf{Hom}_A(t', t))$ because the latter is not even well-formed (t' is not a term of A^- and t is not a term of A).[3] Now, this manner of preventing symmetry will shortly turn out to be far too crude—sometimes we *do* want hom-terms to have inverses, so a theory that syntactically forbids such a possibility isn't much good. But even after we refine our theory to account for this, the positive- and negative-type annotations will still turn out to play a vital role in preventing symmetry from being provable in general.

We can see that the syntactic restrictions of hom-types in arbitrary contexts are too strong by considering how to carry out hom-*introduction*. In undirected type theory, identity-introduction asserts the existence of a term refl_t of type $\mathsf{Id}(t, t)$ witnessing the reflexivity of identity. This is what we want in directed type theory too: in order to think of our types as synthetic categories, we must have *identity morphisms* of the form $\mathsf{Hom}(t, t)$ for every object (i.e. term) t; in order to think of our types as directed homotopy spaces, we must have the trivial directed path; in order to think of hom-types as temporally-modulated equality, we ought to include the degenerate case that every term becomes itself in a process lasting zero seconds. But the reflexivity assertion is forbidden by our typing discipline as it currently stands: t cannot serve as both the domain and codomain, as it would need to be a term of A^- *and* A. So we have a new polarity problem, which is much more imposing than the polarity problem for Hom.

[3] Note that t' and t *do* have the right shape for the A^--hom type $\mathsf{Hom}_{A^-}(t', t)$ to be well-formed. This accords with our desire to think of A^- as the *opposite category* of A.

Before presenting our general solution, let us note that this is no problem at all in the empty context. Recall that a type A in the empty context is the same thing as a category, and a term $t : \mathsf{Tm}(\bullet, A)$ is the same thing as an object of A. It's therefore no trouble for a term to be both a term of A and A^-, because the two have exactly the same objects. So it suffices to define $\mathsf{refl}_t : \mathsf{Tm}(\bullet, \mathsf{Hom}_A(t,t))$ as just the identity morphism on t, and everything works out fine. So the question is: how do we generalize this solution beyond the empty context?

This is the point where our development of hom-types in the category model diverges from North's. In order to type refl, she employs *core types*. The category model supports another operation on types, which we'll denote $(\_)^0$, defined in just the same way as $(\_)^-$, but this time using the *core groupoid* operation (as an endofunctor on Cat) instead of the $(\_)^{\mathrm{op}}$ endofunctor. That is, we can compose the functor $\mathrm{core} : \mathrm{Cat} \Rightarrow \mathrm{Grpd}$ which takes a category $\mathcal{C}$ to its maximal subgroupoid $\mathrm{core}(\mathcal{C})$ with the inclusion $\mathrm{Grpd} \Rightarrow \mathrm{Cat}$ to get an endofunctor on Cat. This endofunctor—like the $(\_)^{\mathrm{op}}$ endofunctor—induces an operation on the types of the category model, defined by post-composition. Types of the form A^0 have a very useful property: a term of type A^0 can be coerced into either a term of type A or into a term of type A^-. This solves the problem of refl's polarity: just restrict the statement of refl_t to only terms t of type A^0. Then $\mathsf{Hom}(t,t)$ is well-formed, because we coerce t into A^- to fit into the domain position and into A to fit into the codomain position. North goes on to state the elimination rules for hom-types (i.e. directed path induction), and use them to prove a transport law and a composition operation for hom-types.

North's solution has the benefit of working in arbitrary contexts: by treating the mixed-variance of refl as a *shallow* polarity problem (that is, solving it by restricting the *type* to be groupoidal), there's no need for deep polarity; indeed, North's theory[4] doesn't include deep polarity at all. We, however, take an alternative approach: if we're willing to accept a restriction to neutral contexts, then we can employ the machinery developed in section 2.3 to solve this polarity problem without the use of core types. If we're in a neutral context, *any* term can be coerced into the opposite type, so we can take refl_t to be a term of type $\mathsf{Hom}(t, -t)$ and proceed from there.

Definition 3.1.3. Define the Hom-**introduction** operation of the category model, which is expressed in rule form as

$$\frac{\Gamma : \mathsf{NeutCon} \quad t : \mathsf{Tm}(\Gamma, A^-)}{\mathsf{refl}_t : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, -t))}$$

by the following.

[4] At least as stated in [Nor19].

$$
\begin{aligned}
\mathsf{Hom}(t, -t)\ \gamma_{01}\ (\mathsf{refl}_t\ \gamma_0) &= (-t)\ \gamma_{01} \circ A\ \gamma_{01}\ (\mathsf{refl}_t\ \gamma_0) \circ t(\gamma_{01}) \\
&= (-t)\ \gamma_{01} \circ A\ \gamma_{01}\ \mathsf{id}_{t\ \gamma_0} \circ t(\gamma_{01}) \\
&= A\ \gamma_{01}\ (t\ \gamma_{01}^{-1}) \circ \mathsf{id}_{A\ \gamma_{01}(t\ \gamma_0)} \circ t(\gamma_{01}) \\
&= A\ \gamma_{01}\ (t\ \gamma_{01}^{-1}) \circ t(\gamma_{01}) \\
&= t(\gamma_{01} \circ \gamma_{01}^{-1}) \\
&= t(\mathsf{id}_{\gamma_1}) \\
&= \mathsf{id}_{t\ \gamma_1} \\
&= \mathsf{refl}_t\ \gamma_1
\end{aligned}
$$

Figure 3.1: Calculation of the morphism part of refl. Recall that the morphisms of the category $\mathsf{Hom}(t, t')\ \gamma_1$ are identities.

PSEUDOAGDA

```
refl : {Γ : NeutCon}{A : Ty Γ} → (t : Tm(Γ, A⁻)) → Tm(Γ, Hom(t,–t))

(refl t) : (γ : |Γ|) → |Hom(t,–t) γ|
refl t γ := id_{t γ}

(refl t) : (γ₀₁ : Γ [ γ₀ , γ₁ ]) →
  (Hom(t,–t) γ₁) [ Hom(t,–t) γ₀₁ (refl t γ₀) , refl t γ₁ ]
refl t γ₀₁ := Figure 3.1
```

It's worth remarking that *both* of these solutions generalize the construction in the empty context: both coercions (North's coercion from A^0 to A^- and A, and ours from A to A^-) do nothing to terms in the empty context, as the closed types (i.e. categories) A^0, A^-, and A have precisely the same terms (objects). And it seems that our approaches can be viewed as equivalent in *any* neutral context. If we include *both* the neutral-context machinery *and* core types, then we can coerce between terms of A and A^0 in a neutral context as well. So the restriction that t be a term of A^0 in order to come equipped with refl_t is no restriction at all, and the different possible Hom-introduction rules come into alignment.

We choose to pursue the neutral-context approach—excluding any reference to core types—because we are convinced that it's essential to work in a neutral context anyways. As covered in the previous chapter, the Π-types of the category model seem practically useless in an arbitrary context, forbidding us from even *forming the type of the identity function* (let alone performing meaningful work). Thus, the benefit that North's refl is stated in an arbitrary context is not compelling to us. Moreover, it's unclear how to work with negative and core types outside a neutral context anyways: North doesn't include any mechanism for actually obtaining terms of type A^0 (unless there's one in the context) or for converting between A and A^-, because these kinds

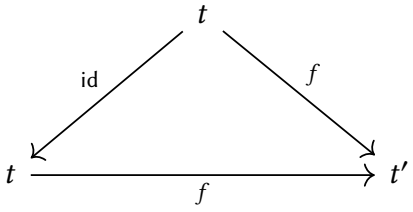
of operations depend critically on the context being neutral. To our understanding, current work to extend North's theory to one capable of synthetic category theory (including Π-types) involves equipping it with a deep polarity calculus after all; we suspect that a mature development of such a theory will overlap significantly with the present work, and will ultimately need to make essential use of (something like) neutral contexts. Given that, we find it more elegant (syntactically and semantically) to forego core types in the statement of the introduction and elimination of hom-types, and to embrace the restriction to neutral contexts.

This also clears the way for stating the elimination principle of hom-types, *directed path induction*. In the directed setting, there are two ways of performing path induction: a *forwards* path induction that proves a statement about arbitrary *coslices* by proving it for refl, and a *backwards* path induction that does the same for *slices*. As we might expect, these are dual to each other: doing slice path induction in A is the same thing as doing coslice path induction in A^-, and vice-versa. We'll make this precise by just introducing one of them—we've arbitrarily selected *coslice* path induction—and then in the next section we'll obtain the other as a consequence.

Following [HS95, Section 4.10], let's consider this in the empty context of the category model. Coslice path induction is based at a term $t : \mathsf{Tm}(\bullet, A^-)$, which is just an object of the category A. A *motive* M for coslice path induction is a type family depending on variables x' of type A and u of type $\mathsf{Hom}(t, x'))$. That is, it's a functor

$$M : t/A \Rightarrow \mathrm{Cat}.$$

Then the principle of coslice path induction says that, to inhabit M abstractly over x', u, i.e. to construct a term in $\mathsf{Tm}(t/A, M)$, it suffices to supply some $m : \mathsf{Tm}(\bullet, M[-t, \mathsf{refl}_t])$, i.e. an object of the category $M(t, \mathsf{id})$. The reason why this works is that identity morphisms are initial in the coslice category: given arbitrary t' and f, the following triangle commutes.



I.e. f is a coslice morphism $(t, \mathsf{id}_t) \to (t', f)$. This gives us the 'object part' of our desired term in $\mathsf{Tm}(t/A, M)$: just send (t', f) to $M\ f\ m : |M(t', f)|$. The morphism part is defined analogously, just using the functoriality of M. This construction generalizes to non-empty neutral contexts: everything is parametrized over some context Γ, but the idea remains the same.

Definition 3.1.4 (Coslice Path Induction)**.** Figure 3.2 constructs in the category model an operation J^+ of the form

$$\frac{\begin{array}{cc}\Gamma : \mathsf{NeutCon} & A : \mathsf{Ty}\ \Gamma \\ t : \mathsf{Tm}(\Gamma, A^-) & M : \mathsf{Ty}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0)) \\ & m : \mathsf{Tm}(\Gamma, M[\mathsf{id} ,_+ -t ,_+ \mathsf{refl}_t]\end{array}}{\mathsf{J}^+\ m : \mathsf{Tm}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0), M)} \quad \text{(Coslice Path Induction)}$$

PSEUDOAGDA

```
J⁺ : (t : Tm(Γ, A⁻))
  ⟶ (M : Ty (Γ ▷⁺ A ▷⁺ Hom(t[ p ], v₀)))
  ⟶ Tm(Γ, M[ id ,₊ −t ,₊ reflₜ ])
  ⟶ Tm(Γ ▷⁺ A ▷⁺ Hom(t[ p ], v₀), M)

(J⁺t,M m) : (γ : |Γ|) ⟶ (a : |A γ|) ⟶ (x : (A γ) [t γ , a]) ⟶ | M(γ, a, x) |
(J⁺t,M m) γ a x = M (id_γ, x) (m γ)  -- x = x ∘ A id_γ id_tγ ∘ t id_γ

(J⁺t,M m) : (γ₀₁ : Γ [γ₀,γ₁])
    ⟶ (a₀₁ : Aγ₁ [A γ₀₁ a₀, a₁])
    ⟶ M(γ₁, a₁, a₀₁ ∘ A γ₀₁ x₀ ∘ t(γ₀₁) )[
          M(γ₀₁,a₀₁) ((J⁺t,M m) γ₀ a₀ x₀),
          ((J⁺t,M m) γ₁ a₁ (a₀₁ ∘ A γ₀₁ x₀ ∘ t(γ₀₁)))
      ]
(J⁺t,M m) γ₀₁ a₀₁ = M (id_γ₁, a₀₁ ∘ A γ₀₁ x₀ ∘ t(γ₀₁)) (m γ₀₁)
```

Figure 3.2: Construction of the J^+-rule (coslice path induction) in the category model.

satisfying the equation

$$(\mathsf{J}^+\ m)\ [\mathsf{id} ,_+ -t ,_+ \mathsf{refl}_t] = m. \qquad \text{(Coslice } \beta\text{)}$$

Remark 3.1.5. When the base point t and/or the motive M can be inferred from context, we will omit them from the J^+ subscript.

Sometimes when using Coslice Path Induction, we'll use a motive $M : \mathsf{Ty}(\Gamma \triangleright^+ A)$ that only depends on the codomain of the slice, not the hom-term itself; formally, this is just doing coslice path induction with motive $M[\mathsf{p}]$. The resulting term $\mathsf{J}^+\ m$ is then a term in context $\Gamma \triangleright^+ A \triangleright^+ \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0)$, of type $M[\mathsf{p}]$.

Since this has the form of a logical rule, probably the most natural interpretation with which to read this principle is the logical-temporal one. Under that reading, Coslice Path Induction says that predicates like M are functorial in the future progress of our base-point t: we can prove $M(t', f)$ for any "future" of t (some t' which t might become, and an arbitrary process f by which t becomes t') just by proving it (supplying the method m) for the "trivial future" (t is rendered as a "future state" trivially, $-t$, and the process by which t becomes itself is the trivial process refl_t). As t develops into t' by f, m develops into $(\mathsf{J}^+\ m)(t', f)$ along with it. As Coslice β reminds us, if we instantiate this to the trivial future, i.e. $(\mathsf{J}^+\ m)(-t, \mathsf{refl}_t)$, then we get back our input m: if t doesn't develop, then neither does m.

As with its undirected cousin, directed path induction will prove a powerful tool

for type-theoretic reasoning. As a first example, we can make good on the promise that *directed types are synthetic categories* by constructing the composition operation among hom-terms. For us, f and g will be said to be **composable** when the domain of g is the *negation* of the codomain of f, i.e. $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t'))$ and $g : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', t''))$ for some $t : \mathsf{Tm}(\Gamma, A^-)$ and $t', t'' : \mathsf{Tm}(\Gamma, A)$. If this is so, then we can define their composite $f \cdot g$[5] by path induction: $f \cdot \mathsf{refl}_{-t'}$ ought to be just f, and this suffices to define $f \cdot g$ for arbitrary g. This is done more precisely in Figure 3.3. Of course, composition of hom-terms is an essential component of all the interpretations listed at the top of the chapter.

- **Category**: composition of morphisms in the synthetic category: if the codomain of f matches the domain of g, then $f \cdot g$ is a morphism with the domain of f and the codomain of g.
  As we'll demonstrate as part of the proof of Proposition 3.2.2, the interpretation of the term $f \cdot g$ in the category model indeed sends $\gamma : |\Gamma|$ to $g(\gamma) \circ f(\gamma) : |\mathsf{Hom}(t, t'')\ \gamma|$. So, as with various constructs we're adding to directed type theory, this construction of synthetic composition is internalizing the *actual* category-theoretic composition; these kinds of things happen when mathematical objects give semantics for synthetic theories of themselves. We defer the proof of the category laws for now, but we can already see that one unit law—that $f \cdot \mathsf{refl}$ is f—holds definitionally (by Coslice β), and the rest is easy directed path induction.
- **Homotopy**: composition of directed paths: given a directed path from t to t' and one from t' to t'', then they can be composed to get a directed path from t to t''.
- **Computational**: composition of evaluations: if t computes to t', and t' computes to t'', then these computations can be stuck together, witnessing that t computes to t''. If hom-types are understood as the internalization of the reduction relation of the metatheory, then this internalizes that said relation is transitive.
- **Logical-Temporal** composition of 'becoming': if t becomes t' in some period of time, and t' becomes t'' in some other period of time, then t becomes t'' in the sum of those time periods.

The only oddity to make sense of is that our notion of 'composability' comes with polarity annotations: as mentioned, what it means for the codomain of f to "match" the domain of doesn't mean that the former term and the latter term are the *same*—that wouldn't make sense in the polarity calculus—it means that *one is the negation of the other*; in the directed homotopy interpretation, t' can serve as the *endpoint* of the directed path f, but we must recast it as $-t$ for it to stand as the *starting point* of the directed path g; and so on.

Notice that Figure 3.3 is not a construction in the category model specifically; rather, it defines composition in the *syntax of directed type theory*. To make precise what this is, we introduce *directed CwFs*, an (indexed) GAT $\mathfrak{DCwF}$ extending the (indexed) GAT $\mathfrak{MPCwF}$. Like any indexed GAT, this admits an initial model, the syntax of directed type theory. Thus, the construction of Figure 3.3 doesn't just work in the category model, but in any $\mathfrak{DCwF}$-algebra. For instance, in the preorder model, it encodes the fact that the synthetic preorders of that model are indeed transitive. Here is our notion.

[5]Note that we switch to the diagrammatic order for writing compositions, as is frequently the convention for working with identity types in undirected type theory.

Definition 3.1.6. A **directed CwF (DCwF)** is a NPCwF equipped with the following structure:

- a type former

$$\mathsf{Hom} : \{\Gamma : \mathsf{Con}\}\{A : \mathsf{Ty}\ \Gamma\} \to \mathsf{Tm}(\Gamma, A^-) \to \mathsf{Tm}(\Gamma, A) \to \mathsf{Ty}\ \Gamma$$

  which is stable under substitution:

$$\mathsf{Hom}(t, t')[\sigma] = \mathsf{Hom}(t[\sigma], t'[\sigma])$$

  and such that the hom-types of neutral types are neutral;
- in any $\Gamma : \mathsf{NeutCon}$, a term $\mathsf{refl}_t : \mathsf{Hom}(t, -t)$ for each term $t : \mathsf{Tm}(\Gamma, A^-)$, also stable under substitution by $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$ for $\Delta : \mathsf{NeutCon}$; and
- a term former J^+ as given in Definition 3.1.4, also appropriately stable under substitution: for $\Delta, \Gamma : \mathsf{NeutCon}$ and $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$,

$$\mathsf{J}^+(m[\sigma]) = (\mathsf{J}^+\ m)[\mathsf{q}(\sigma; A, \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0))] \qquad \mathsf{J}^+[]$$

Notice that, unlike the stability-under-substitution properties for type-formers in CwFs (the kind we considered in section 1.2), refl and J^+ are only stipulated to be stable under substitution between *neutral* contexts. It is, indeed, only with respect to such substitutions that we can state stability-under-substitution. In the present work, we don't elaborate on what happens if, say, one forms refl_t in a neutral context Γ, and then substitutes that term along $\sigma : \mathsf{Sub}\ \Delta\ \Gamma$ where Δ is *not* neutral; the only such situation which we'll encounter presently (particularly in chapter 4) is where Δ is an extension of Γ and σ is the composition of the projection morphisms p. In this case, there's no need to characterize what $\mathsf{refl}_t[\mathsf{p} \circ \cdots \circ \mathsf{p}]$ is: we'll ultimately be working in Γ anyways, and so any instance of $\mathsf{refl}_t[\mathsf{p} \circ \cdots \circ \mathsf{p}]$ will eventually be substituted back into Γ by some morphism of the form $(\mathsf{id} ,_+ s_1 ,_+ \ldots ,_+ s_n)$, giving us back refl_t again. However, we defer a careful consideration of this issue (and its possible impacts on the canonicity of the system) to future work.

DIRECTEDTT

```
Csl : {t : Tm(Γ, A⁻)} ⟶ Ty (Γ ▷⁺ A)
Csl {t} := Hom(t[ p ], v₀)

_·v₀ : Tm(Γ, Hom(t,t')) ⟶ Tm(Γ ▷⁺ A ▷⁺ Csl{-t'}, Hom(t[ p ∘ p ], v₁)
f ·v₀ := (J⁺₋t',Csl{t} f)

_·_ : Tm(Γ, Hom(t,t')) ⟶ Tm(Γ, Hom(-t',t'')) ⟶ Tm(Γ, Hom(t,t''))
f · g := (f ·v₀) [ id ,₊ t'' ,₊ g ]
```

Figure 3.3: Composition of hom-terms, in the syntax of directed type theory.

Directed path induction also allows us to establish our principle of *transport*, which, recall, was a reflection of the *fibrancy* of our model into the syntax. As nicely summarized by [AN22], *un*directed transport realizes the principle of *indiscernibility of identicals*, that identical terms can be used in place of each other, with equivalent results. In the directed setting, this has the same intuition as the one given above for J^+:[6] if t can become/be transformed into t', then the properties of t transform functorially into properties of t'. Thus, we replace the "the indiscernibility of identicals" with something like "the analogy of transformables"—the word "analogy" here meaning that terms t and t' connected by a directed equality do not necessarily have the *same* properties, but that transport allows us to convert the properties of t into properties of t', *mutatis mutandis*.

Definition 3.1.7. In the syntax of directed type theory, we have an operator

$$\frac{f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t')) \quad b : \mathsf{Tm}(\Gamma, B[\mathsf{id} ,_+ -t])}{\mathsf{tr}_B\ f\ b := (\mathsf{J}^+\ b)[\mathsf{id} ,_+ t' ,_+ f] : \mathsf{Tm}(\Gamma, B[\mathsf{id} ,_+ t'])}$$

This is the form of transport which will be more useful for our purposes. However, with some extra effort, we can recover a version which is functorial in the term b.

Definition 3.1.8. Let $B : \mathsf{Ty}(\Gamma \rhd^+ A)$ and $t : \mathsf{Tm}(\Gamma, A^-)$ as before. Then recall the identity function (Definition 2.3.11)

$$I_{B[\mathsf{id} ,_+ -t]} : \mathsf{Tm}(\Gamma, B[\mathsf{id} ,_+ -t][\mathsf{e}] \to B[\mathsf{id} ,_+ -t]).$$

This is a term of type $M[\mathsf{id} ,_+ -t ,_+ \mathsf{refl}]$ where

$$M := \Pi(B[\mathsf{id} ,_+ -t][\mathsf{e}][\mathsf{p}_- \circ \mathsf{p}_-],\ B[\mathsf{p}_+ \circ \mathsf{p}_-]) : \mathsf{Ty}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0))$$

(see Figure 3.4 and Figure 3.5). Thus, for any $t' : \mathsf{Tm}(\Gamma, A)$ and $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t')$, obtain

$$\mathsf{transport}_B\ f := (\mathsf{J}^+\ I_{B[\mathsf{id} ,_+ -t]})[\mathsf{id} ,_+ t' ,_+ f] \quad : \mathsf{Tm}(\Gamma, B[\mathsf{id} ,_+ -t] \to B[\mathsf{id} ,_+ t']) \tag{3.1.9}$$

Let's now return to a vital question for any putative directed type theory: *why doesn't this collapse into undirected type theory?* That is, what prevents our new J-rule from being able to prove the invertibility of hom-terms? Now that we're working in neutral contexts, the type of symmetry is well-formed:

$$\mathsf{symm} : \mathsf{Tm}(\Gamma, \mathsf{Hom}_A(t, t')) \to \mathsf{Tm}(\Gamma, \mathsf{Hom}_A(-t', -t)).$$

And, if we follow the usual definition of this operation in undirected type theory, it seems like it might be possible: could we not just put

$$\mathsf{symm}\ \mathsf{refl}_t := \mathsf{refl}_t : \mathsf{Tm}(\Gamma, \mathsf{Hom}(- - t, -t))$$

[6] Transport is, after all, just a specialization of directed path induction.

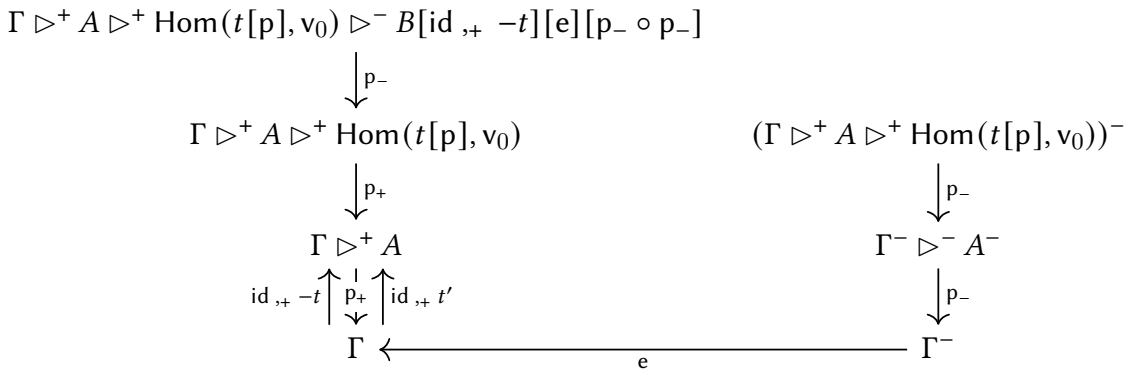


Figure 3.4: The contexts and substitutions involved in Definition 3.1.8.

$$
\begin{aligned}
&M[\mathsf{id} ,_+ t' ,_+ f] \\
&= (\Pi(B[\mathsf{id} ,_+ -t][\mathsf{e}][\mathsf{p}_- \circ \mathsf{p}_-],\ B[\mathsf{p}_+ \circ \mathsf{p}_-]))\ [\mathsf{id} ,_+ t' ,_+ f] \\
&= \Pi\ (B[\mathsf{id} ,_+ -t][\mathsf{e}][\mathsf{p}_- \circ \mathsf{p}_-][(\mathsf{id} ,_+ t' ,_+ f)^-])\ \ (B[\mathsf{p}_+ \circ \mathsf{p}_-][\mathsf{q}((\mathsf{id} ,_+ t' ,_+ f)^-; \ldots^-)^-]) \\
&= \Pi\ (B[\mathsf{id} ,_+ -t][\mathsf{e}][\mathsf{p}_- \circ \mathsf{p}_-][\mathsf{id} ,_- t' ,_- f])\ \ (B[\mathsf{p}_+ \circ \mathsf{p}_-][((\mathsf{id} ,_- t' ,_- f) \circ \mathsf{p}_+ ,_+ \mathsf{v}_0)^-]) \\
&= \Pi\ (B[\mathsf{id} ,_+ -t][\mathsf{e}])\ \ (B[\mathsf{p}_+ \circ \mathsf{p}_-][(\mathsf{id} ,_+ t' ,_+ f) \circ \mathsf{p}_- ,_- \mathsf{v}_0]) \\
&= \Pi\ (B[\mathsf{id} ,_+ -t][\mathsf{e}])\ \ (B[\mathsf{p}_+][\mathsf{id} ,_+ t' ,_+ f][\circ \mathsf{p}_-]) \\
&= \Pi\ (B[\mathsf{id} ,_+ -t][\mathsf{e}])\ \ (B[\mathsf{id} ,_+ t'][\mathsf{p}_-]) \\
&= B[\mathsf{id} ,_+ -t][\mathsf{e}] \to B[\mathsf{id} ,_+ t']
\end{aligned}
$$

Figure 3.5: Substituting the motive M in Definition 3.1.8 with some $t' : \mathsf{Tm}(\Gamma, A)$ and $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t'))$. Note that $I_{B[\mathsf{id} ,_+ -t]}$ has this type when $t' = -t$ and $f = \mathsf{refl}_t$,

and then obtain symm f for arbitrary f by directed path induction? The reason this does not work is subtle, but reveals itself if we try to write down the motive for this putative directed path induction:

$$M := \mathsf{Hom}_{A[\mathsf{p}\circ\mathsf{p}]}(-\mathsf{v}_1, -t[\mathsf{p}\circ\mathsf{p}]).$$

This is *not* a valid type in context $\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0)$: we must negate the variable v_1, because it is a term of type $A[\mathsf{p}\circ\mathsf{p}]$ and the domain of a hom must be negative. But the context $\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0)$ is not neutral, and therefore does not permit term-negation; nor is $\Gamma \rhd^+ A$ neutral, so we couldn't replace $-\mathsf{v}_1$ with $(-\mathsf{v}_0)[\mathsf{p}]$ either. This attempt at proving symmetry fails; two critical aspects of our polar typing discipline—the polarity requirements of Hom formation and the neutrality requirement for term-negation—have worked in tandem to make this a genuine *directed* type theory, with hom composition provable by J but not symmetry. Of course, we haven't shown that there's *no* proof of symm—that requires a metatheoretic argument, which we'll carry out at the end of this section.

Observe that if A *is* a neutral type, then $\Gamma \rhd^+ A$ *is* a neutral context, and thus we can write $-\mathsf{v}_0 : \mathsf{Tm}(\Gamma \rhd^+ A, A[\mathsf{p}]^-)$. Generally speaking, variables (or terms containing them) can be negated whenever their type is a neutral type in a neutral context. For instance, in context $\Gamma \rhd^+ A_2 \rhd^+ A_1[\mathsf{p}] \rhd^+ A_0$ with Γ neutral and $A_1 : \mathsf{NeutTy}\ \Gamma$, we can negate the variable v_1 regardless of whether A_2 and A_0 are neutral: just regard $-\mathsf{v}_1$ as an abbreviation for

$$(-\mathsf{v}_0)[\mathsf{q}(\mathsf{p}_{A_2})][\mathsf{p}_{A_0}] : \mathsf{Tm}(\Gamma \rhd^+ A_2 \rhd^+ A_1[\mathsf{p}] \rhd^+ A_0, A_1[\mathsf{p}\circ\mathsf{p}\circ\mathsf{p}])$$

where the negation of v_0 happens in $\Gamma \rhd^+ A_1$, which is neutral. In the next chapter, we adopt an informal principle codifying this practice.

So, returning to symmetry, if $A : \mathsf{NeutTy}\ \Gamma$, then the construction of symm for the hom-types of A works because we *can* negate the relevant variable—see Figure 3.6. This makes good on the idea that neutral types are *synthetic groupoids*; if A is neutral (which, recall, means that it's interpreted in the category model as a family of groupoids), then we can prove internally that it's a groupoid. We therefore adopt the following notation.

Remark 3.1.10. If A is a neutral type, then we'll write $\mathsf{Id}(t, t')$ for the hom-type $\mathsf{Hom}(t, t')$. We will also avoid making negations explicit, e.g. writing $\mathsf{refl}_t : \mathsf{Id}(t, t)$ instead of $\mathsf{Id}(t, -t)$, when dealing with neutral types.

Remark 3.1.11. Let us briefly touch on a point brought up in the beginning of this chapter. Suppose A is a type, which we do *not* assume to be neutral. Then we can talk about directed equality of terms of type A, i.e. form the type $\mathsf{Hom}(t, t')$ for $t : \mathsf{Tm}(\Gamma, A^-)$ and $t' : \mathsf{Tm}(\Gamma, A)$. But we cannot speak of *undirected* equality between terms of type A, that is, the type $\mathsf{Id}(t, t')$ is *meaningless*. This type *only* means something when A is neutral, in which case it's notation for $\mathsf{Hom}(t, t')$.

However, we might want to endow the type A with identity types *in addition* to its hom-types. A way to do so would be to have another type A' which has the exact same terms as A but is *neutral*. Then we could say $\mathsf{Id}_A(t, t')$ to actually mean $\mathsf{Id}_{A'}(t, t')$.

directedTT

```
symm : {A : NeutTy Γ}{t : Tm(Γ, A⁻)}{t' : Tm(Γ, A)} →
  (f : Tm(Γ, Id(t,t'))) → Tm(Γ, Id(-t',-t))
symm f := (J⁺t,S [ id ,+ t' ,+ f ]
  where
    S : Ty (Γ ▷⁺ A ▷⁺ Hom(t, v₀))
    S := Id(-v₀[ p ], -t[ p ∘ p ])
```

Figure 3.6: Symmetry of identity types of neutral types. Alternatively, symm $f := \mathrm{tr}_{\mathsf{Hom}(-\mathsf{v}_0,-t[\mathsf{p}])}\ f\ \mathrm{refl}_t$ arrives at the same result.

Thus A would have both an identity type structure *and* a hom-type structure, and we could study their interaction. Now, the category model actually supports *two* canonical choices for how to do this.

- As mentioned, we have the **core** construction, an endofunctor on Cat whose object part sends a category to its maximal sub-groupoid (taken as a category). This becomes an operation on types by post-composition, so for every type (in any context) A, we get a neutral type A^0 in the same context. As North observes, the terms of A^0 can be coerced into terms of A (and A^-) in an arbitrary context; but in a neutral context, this coercion is reversible: A^0 and A have the *same terms*:
$$\mathsf{Tm}(\Gamma, A^0) \cong \mathsf{Tm}(\Gamma, A).$$
So, if we follow the above procedure, understanding $\mathsf{Id}_A(t,t')$ to mean $\mathsf{Id}_{A^0}(t,t')$—coercing silently across the above isomorphism—then we get a notion of identity for every type.[a]
This notion of identity is *stronger* than the hom-types: a term ϕ of type $\mathsf{Id}_{A^0}(t,t')$ is interpreted as a family of *isomorphisms* between the interpretations of t and t'—for every object γ of Γ, $\phi(\gamma)$ is an $A(\gamma)$-isomorphism between $t(\gamma)$ and $t'(\gamma)$. So we could split this into its two directions, obtaining $\phi^{\rightarrow} : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t,t'))$ and $\phi^{\leftarrow} : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', -t))$.
- Dually, we can consider the groupoid **closure**, i.e. the Cat-endofunctor sending a category to its *minimal super-groupoid*, which we obtain by freely adding inverses to every morphism. This too becomes a type-to-neutral-type operation, denote it A^*. The attending notion of identity we'd get by the procedure above is *weaker* than directed equality: for every $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t,t'))$, we could obtain an identity $\mathsf{Id}_{A^*}(t,t')$ just by pairing the components of f with their formal inverses.

So we have, in principle, two ways of equipping a directed type with undirected identity types related to its hom-types, one stronger and one weaker, one 'right' and one 'left' (referring to the fact that core and closure are defined with the right- and left-adjoints, respectively, to the forgetful functor $\mathrm{Grpd} \Rightarrow \mathrm{Cat}$). Depending on our desired interpretation, we may want one notion of identity, or the other, or perhaps

both.

[a]Since a groupoid is its own maximal sub-groupoid, this gives us back the same identity types if A is already neutral.

Recall that the groupoid model was originally devised to establish the independence of UIP: it showed that the ordinary rules of Martin-Löf Type Theory permit identity types to have multiple elements which are not themselves identical. In the language of homotopy type theory, the groupoid model includes types which are not *0-truncated* (also known as *h-sets*), but are instead *1-truncated types* (i.e. *h-groupoids*). However, unlike general homotopy type theory, the groupoid model doesn't permit truncation levels above 1: while it doesn't validate UIP, it does validate "UIP one level up", that is, the identity types of the groupoid model *are* h-sets—two identity proofs, if identical, are identical by a unique identity proof. Semantically, this corresponds to the fact that identity types are interpreted as families of *discrete* groupoids, i.e. sets treated as groupoids—there's no way for the identity types of identity types to have nontrivial structure.

A priori, the syntax of directed type theory—the initial DCwF—allows us to express all levels of higher categories: we can form the type of homs-between-homs, and homs-between-homs-between-homs, and so on forever. But, as with the groupoid model, in the category model this trivializes after the first step. However, in directed type theory, things are a bit more complex. If we iterate homs-between-homs-between-..., there are *two* ways in which things can "trivialize": beyond a certain level, the types in question can become *subsingletons*, i.e. h-props (the point at which this happens is the *truncation level* mentioned above), but, independently, all the hom-types above a certain level can become *neutral*, i.e. symmetric. These two "dimensions of truncation" are well-known in the category theoretic literature, in the notion of an (m, n)-category (where m and n are natural numbers or ∞). We give names to the first few steps in this hierarchy.

Definition 3.1.12. A type A is called a

- **proposition** if it is neutral and, for every $t, t' : \mathsf{Tm}(\Gamma, A)$, there is a term of type $\mathsf{Id}(t, t')$.
- **poset**, or **(0,1)-truncated type**, if all its hom-types are propositions.
- **set**, or **(0,0)-truncated type**, if it is a neutral poset.
- **category**, or **(1,1)-truncated type**, if all its hom-types are sets.
- **groupoid**, or **(1,0)-truncated type**, if it is a neutral category.

We don't consider (m, n)-truncation in general, but the pattern is clear: propositions—which we could take as the notion of "$(-1, 0)$-truncated type"—are the base case, neutral types always have 0 as the second truncation parameter, and an $(m+1, n+1)$-truncated type has (m, n)-truncated types as its hom-types. Interpreting types as directed homotopy spaces, the (m, n) parameters reflect the usual *homotopy dimension*, m, of the space (the same way an m-truncated type in HoTT is understood as an m-dimensional homotopy space) and the *category-theoretic dimension*, n, of its directed paths.

Let's make a few comments about this definition. First, note that we do much of our quantification at the metalanguage ("there is a term...", "all its hom-types...") instead of internalizing the quantification into the object language using dependent

types. It's possible that some or all of this quantification could be moved inside the object language, as is the practice in undirected type theory (homotopy type theory in particular). However, this is more difficult in directed type theory, because we have to account for variances; universal quantification has to be *functorial* with the right *variance* in order to be admitted into the polarized syntax. Our purpose in making this definition is to delineate different varieties of DCwF, so metalanguage quantification is alright for us. But a more mature version of the present theory aiming to articulate higher category theory would need to substantively address this issue.

Second, notice that we say "poset" instead of "preorder". In the present theory, there's not a precise statement of antisymmetry: only neutral types have identity types, so, for a non-neutral poset, we can't actually say that hom-terms in either direction between two terms implies those terms are *identical*. But we adopt this terminology in anticipation of the above-mentioned method for extending identity types to non-neutral types by way of core types. If, say, we had core types with the appropriate observational laws—namely that a hom in A^0 between t and t' is given by a synthetic isomorphism between t and t', i.e. homs in A from t to t' and from $-t'$ to $-t$ that compose to the respective refl's—and defined $\mathsf{Id}(t,t')$ to mean $\mathsf{Id}_{A^0}(t,t')$, then *posets* in the sense above really would be antisymmetric: if we had a witnesses that "$t \leq t'$", i.e. some $f\colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(t,t'))$ and that "$t' \leq t$", i.e. some $g\colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t',-t))$, then these would necessarily compose to the respective refl terms (since the hom-types of A are propositions), thus yielding a hom-term in A^0 between t and t', i.e. an identity $\mathsf{Id}(t,t')$. There are more details to attend to than we care to consider presently, but this is the motivation for saying "poset", and likewise with "set" instead of "setoid".[7]

Our reason for introducing these distinctions is to make the following definition, specializing the notion of 'DCwF' based on its maximum truncation level.

Definition 3.1.13.
- A **(1,1)-DCwF** is a DCwF all of whose types are categories.
- A **(1,0)-DCwF** is a DCwF all of whose types are groupoids.
- A **(0,1)-DCwF** is a DCwF all of whose types are posets.
- A **(0,0)-DCwF** is a DCwF all of whose types are sets.

Example 3.1.14. The category model is a (1,1)-DCwF: all of its hom-types are neutral and hence we write Id for the hom-types between terms of a hom-type; moreover, the hom-types satisfy UIP, that is, for $f, f'\colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(t,t'))$ and $\alpha, \beta\colon \mathsf{Tm}(\Gamma, \mathsf{Id}(f,f'))$, we have a term of type $\mathsf{Id}(\alpha,\beta)$.

Example 3.1.15. The preorder model is a (0,1)-DCwF.

Notice that an ordinary, *undirected* CwF with intensional identity types is essentially an $(\infty, 0)$-DCwF in this notation,[8] with the ∞ indicating that no particular level of

[7] To carry this terminology throughout, we should also say "univalent category" and "univalent groupoid" above, but we omit this distinction for simplicity.

[8] We're implicitly using a more advanced analogue of Definition 2.2.3, which views any CwF as a NPCwF with trivial polarization, every context and type neutral, and the e and ee isomorphisms being just identities. If the CwF is further equipped with intensional identity types, then those serve as the hom-types, making this into a DCwF as well.

truncation is assumed.[9] And likewise for finite truncation levels: analogously to Example 3.1.14 and Example 3.1.15 respectively, the groupoid model would be a $(1,0)$-DCwF and the setoid model a $(0,0)$-DCwF.

Definition 3.1.16. The **syntax (or language) of (1,1)-directed type theory** ((1,1)-directedTT for short) is the initial (1,1)-DCwF.

To be fully precise, we understand (1,1)-DCwFs an (indexed) GAT: we modify the (indexed) GAT $\mathfrak{DCwF}$ by making the Hom type constructor go into *neutral* types[10] and by adding a constructor witnessing UIP for hom-types. Analogous moves can be made to render, $(0,1)$-DCwFs, $(2,1)$-DCwFs, etc. as (indexed) GATs. In what follows, we won't explicitly refer to these constructors, rather taking it for granted that hom-types are sets.

Remark 3.1.17. Observe that the GAT of (1,1)-DCwFs is a *purely structural* type theory: its initial algebra (the syntax of (1,1)-directedTT) has only the empty context, no types, and no terms. However, we can arrive at a nontrivial directed type theory by appending type- and term-constructors. We will choose only those constructors which are validated by the category model, so the category model continues to be a model of (1,1)-directedTT+(the constructors).

Example 3.1.18. For any set X (in particular the finite sets , $\mathbb{1}$, $\mathbb{2}$, ...), the category model supports a closed type, interpreted as the discrete category on X.

Example 3.1.19. The category model supports a closed type $\vec{\mathbb{2}}$, which is constructed by two points $i_0 : \mathsf{Tm}(\bullet, \vec{\mathbb{2}}^-)$ and $i_1 : \mathsf{Tm}(\bullet, \vec{\mathbb{2}})$ and a term $i_{01} : \mathsf{Tm}(\bullet, \mathsf{Hom}(i_0, i_1))$.

A future version of this theory could perhaps include a framework for introducing and reasoning about *directed higher-inductive types*, i.e. types like $\vec{\mathbb{2}}$ which are generated by term-constructors *and* hom-constructors. However, we leave this to future work.

The walking arrow is among the simplest possible types which is genuinely directed, i.e. is resolutely *not* neutral. This can be seen most acutely in the following independence result, which at long last establishes that our type theory is indeed a *directed* type theory.

Theorem 3.1.20. *The syntax of (1,1)-directed TT + the walking arrow type $\vec{\mathbb{2}}$ cannot prove the symmetry of arbitrary hom-types:*

$$\mathsf{symm} : \{\Gamma\}\{t\}\{t'\} \to \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t')) \to \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', -t)).$$

[9] Likewise, the notion of 'DCwF' given in Definition 3.1.6 could be called (∞,∞)-DCwF, since it doesn't proscribe any truncation.

[10] Since we implement the NeutTy $\hookrightarrow$ Ty subpresheaf in the GAT $\mathfrak{MPCwF}$ as a predicate `isNeut_Ty` : (Γ : Con) ⇒ Ty Γ ⇒ U (see Figure 2.11), this amounts to extending $\mathfrak{DCwF}$ with a witness that `isNeutTy(hom(t, t'))`.

Proof. Suppose such a symm could be constructed in the syntax. Then obtain

$$\mathsf{symm}\ i_{01} : \mathsf{Tm}(\bullet, \mathsf{Hom}(-i_1, -i_0))$$

in the syntax model. By initiality, we would have the same in the category model. But this is interpreted as an element of the set

$$\vec{2}\ [i_1, i_0]$$

of which there are none. Contradiction. □

Remark 3.1.21. If we extend (1,1)-directed TT + $\vec{2}$ with further constructs validated by the category model, the result asserted in Theorem 3.1.20 continues to hold: by hypothesis, the category model will continue to be a model of whatever type theory we obtain, so the initiality map used in the proof of Theorem 3.1.20 will exist for that theory.

In many respects, this result is the culmination of our choice to work in the category model: we wanted to develop a *directed* dependent type theory, i.e. intensional Martin-Löf Type Theory but "dropping the assumption of symmetry" on the identity types to obtain hom-types. But symmetry is a difficult thing to "drop": it's not an axiom of MLTT but rather a *theorem* provable from the J-rule. So we had to impose further discipline on the type theory—the polarity calculus—in hopes that the J-rule would be unable to prove symmetry. But, of course, proving that a formal theory *cannot prove* something requires a metatheoretic argument, so we developed the formal language of (1,1)-directed type theory by abstracting the features of the category model. Theorem 3.1.20 demonstrates that we succeeded in getting rid of *symmetry* as an inherent theorem of type theory.

3.2 Observations and Universes

> "...even the most trivial actions should be undertaken in reference to the end. And the end for rational creatures is to follow the reason and the rule of that most venerable archetype of a governing state—the Universe."
>
> Marcus Aurelius, *Meditations*

The present work sits firmly in the "extensional type theory in intensional type theory" tradition [Hof95b; Alt99]: we seek to build a directed type theory which is *intensional* (in the sense that our hom-types are coarser than definitional directed equality, i.e. reduction), but we also wish to include (the directed analogue of) *extensionality* principles. As we have throughout, we most closely adhere to the development of the groupoid model [HS95] which is resolutely a model of intensional equality—extensional equality must validate the uniqueness of identity proofs, which the groupoid model was specifically designed to refute—but nonetheless validates axioms such as function extensionality. Furthermore, the universe of sets interpreted by the groupoid model also admits a *universe extensionality* principle: the identity type between two terms of the set universe is equivalent to the type of *bijections* between the corresponding

sets. The groupoid model thus prefigures the development of *homotopy type theory* and *univalent foundations*: taking the groupoid model's rejection of UIP to its logical extreme, we obtain the *higher types* characteristic of HoTT, and the attending notion of *universe extensionality* for the universes of higher types is Voevodsky's *univalence axiom*.

As with the rest of directed type theory, the historical development of directed univalence follows the opposite progression of undirected univalence: the groupoid model and its universe extensionality came *before* full HoTT and its univalence axiom, but the present development of directed universe extensionality in the category model is coming *after* the appearance of full *directed* HoTT (in the form of simplicial type theory) and its notion of directed univalence [GWB24]. As noted in the beginning of this thesis, we don't view the success of simplicial type theory as making the present investigation irrelevant: type theory need not be a one-size-fits-all affair—it is often beneficial to utilize a theory with more modest mathematical scope (such as the present theory's restriction to 1-categories) when greater generality (e.g. the ∞-categories of simplicial type theory) is not needed.

3.2.1 Opposites

In the current section, we highlight some of the 'observational' laws that hold in the category model and note how they bolster the informal interpretations of 'directed equality' expounded above. This will take the form of several axioms, which we add to the existing syntax (the syntax of of (1,1)-directedTT + Π + Σ + the types mentioned in the previous section), which we're justified in doing because the category model supports all these constructs and because all these axioms can be written down as GAT extensions of the GAT of (1,1)-DCwFs. First off, we characterize the interaction between hom-types and opposites.

Axiom. For every context Γ, every $A : \mathsf{Ty}\ \Gamma$, and every $t : \mathsf{Tm}(\Gamma, A^-)$, $t' : \mathsf{Tm}(\Gamma, A)$,

$$\mathsf{Hom}_{A^-}(t', t) = \mathsf{Hom}_A(t, t'). \qquad \text{(Op-Observ.)}$$

This axiom finally makes explicit our contention that A^- is supposed to represent the *dual* of A. For instance, if A is a poset, then a term of type $\mathsf{Hom}_A(t, t')$ witnessing $t \leq t'$ is the same as a term $\mathsf{Hom}_{A^-}(t', t)$ witnessing $t' \geq t$. In the directed homotopy space interpretation, this is saying that A^- is the same underlying space as A, but with the direction on all the paths reversed. In the logical-temporal and computational interpretations, this tells us that A^- represents the same relation as A, but with time running backwards.

As is standard in category theory, Op-Observ. opens up the possibility of every construct having a *dual*, obtained by turning all the arrows around. We will see numerous examples of this in chapter 4. Here's our first instance: a principle of *slice path induction*, dual to the *coslice path induction* introduced before.

Definition 3.2.1 (Slice Path Induction)**.** Instantiating Definition 3.1.4 with $A = A^-$ and transforming by Op-Observ., we obtain the following induction principle. A

pseudoAgda

```
J⁻ : (t' : Tm(Γ, A))
  ⟶ (M : Ty (Γ ▷⁺ A⁻ ▷⁺ Hom(v₀, t'[ p ])))
  ⟶ Tm(Γ, M[ id ,₊ −t' ,₊ refl₋ₜ' ])
  ⟶ Tm(Γ ▷⁺ A⁻ ▷⁺ Hom(v₀ , t'[ p ]), M)

(J⁻t',M m) : (γ : |Γ|) ⟶ (a : |A γ|) ⟶ (x : (A γ) [a , t' γ]) ⟶ | M(γ, a, x) |
(J⁻t',M m) γ a x = M (id_γ, x) (m γ)  -- x = t' id_γ ∘ A id_γ id_tγ ∘ x

(J⁻t',M m) : (γ₀₁ : Γ [γ₀,γ₁])
    ⟶ (a₁₀ : Aγ₁ [a₁ , A γ₀₁ a₀])
    ⟶ M(γ₁, a₁, t'(γ₀₁) ∘ A γ₀₁ x₀ ∘ a₁₀ )[
        M(γ₀₁,a₁₀) ((J⁻t,M m) γ₀ a₀ x₀),
        ((J⁻t,M m) γ₁ a₁ (t'(γ₀₁) ∘ A γ₀₁ x₀ ∘ a₁₀))
      ]
(J⁻t',M m) γ₀₁ a₁₀ = M (id_γ1, t'(γ₀₁) ∘ A γ₀₁ x₀ ∘ a₁₀) (m γ₀₁)
```

Figure 3.7: Direct construction of the J^--rule (slice path induction) in the category model.

direct construction in the category model is given by Figure 3.7

$$\frac{\begin{array}{ll}\Gamma : \mathsf{NeutCon} & A : \mathsf{Ty}\ \Gamma \\ t' : \mathsf{Tm}(\Gamma, A) & M : \mathsf{Ty}(\Gamma \rhd^+ A^- \rhd^+ \mathsf{Hom}(\mathsf{v}_0, t'[\mathsf{p}])) \\ & m : \mathsf{Tm}(\Gamma, M[\mathsf{id} ,_+ -t' ,_+ \mathsf{refl}_{-t'}]\end{array}}{(\mathsf{J}^-\ m : \mathsf{Tm}(\Gamma \rhd^+ A^- \rhd^+ \mathsf{Hom}(\mathsf{v}_0, t'[\mathsf{p}])), M)} \quad \text{(Slice Path Induction)}$$

$$(\mathsf{J}^-\ m)\ [\mathsf{id} ,_+ -t' ,_+ \mathsf{refl}_{-t'}] = m \quad \text{(Slice } \beta\text{)}$$

We designate this path induction principle J^- because it is "looking backwards" from the base point, that is, it proves a claim about arbitrary homs whose codomain/destination is the fixed point t'. In terms of category theory, this is just saying that the identity morphism is the *terminal slice*, dual to the assertion (made formal with the J^+ rule) that identity morphisms are also the *initial coslice*. The observation that these *universal mapping properties* are rendered as induction principles will be a central theme of the next chapter.

From the computational and logical-temporal interpretations, we can read J^- as saying that any property $m : M(-t', \mathrm{refl})$ of the current state t'—trivially rendered as the *past* state $-t'$ of zero seconds ago, which trivially becomes t' by refl—has an analogous antecedent property $\mathsf{J}^-\ m : M(t, f)$ for any past state t which becomes t' by any process f. As t evolved into t' by f, $\mathsf{J}^-\ m$ evolved into m.

Now, there are circumstances where either J^+ or J^- could be deployed to prove a claim.

This is particularly the case when defining operations on hom-terms, such as in Figure 3.3. There, we have composable homs $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t'))$ and $g : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', t''))$. We could define their composite $f \cdot g$ as we did above, by coslice path induction with f as the method and g substituted in after; but we could *also* go the other way around: do *slice* path induction with g as the method, and then substitute in f. In the category model, both yield the same thing.

directedTT

```
Sl : {t' : Tm(Γ, A)} ⟶ Ty (Γ ▷⁺ A⁻)
Sl {t'} := Hom(v₀, t'[ p ])

v₀·_ : Tm(Γ, Hom(-t',t'')) ⟶ Tm(Γ ▷⁺ A⁻ ▷⁺ Sl{t'}, Hom(v₁, t''[ p ∘ p ])
v₀· g := (J⁻_{t',Sl{t''}} g)

_·_ : Tm(Γ, Hom(t,t')) ⟶ Tm(Γ, Hom(-t',t'')) ⟶ Tm(Γ, Hom(t,t''))
f · g := (v₀· g) [ id ,+ t ,+ f ]
```

Figure 3.8: Composition of hom-terms, by slice path induction

Proposition 3.2.2. *Given composable hom-terms* $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t'))$ *and* $g : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', t''))$ *in the category model, there is a (metatheoretic) equality*

$$(\mathsf{J}^+ f)[\mathsf{id} ,_+ t' ,_+ g] = (\mathsf{J}^- g)[\mathsf{id} ,_+ t ,_+ f].$$

Proof. Let $\gamma : |\Gamma|$ be arbitrary.

$$\begin{aligned}
(\mathsf{J}^+ f)[\mathsf{id} ,_+ t' ,_+ g]\ \gamma &= ((\mathsf{J}^+ f) \circ (\mathsf{id} ,_+ t' ,_+ g))\ \gamma \\
&= (\mathsf{J}^+ f)\ \gamma\ (t'\ \gamma)\ (g\ \gamma) \\
&= \mathsf{Hom}(t[\mathsf{p} \circ \mathsf{p}], \mathsf{v}_1)\ \mathsf{id}_\gamma\ (t'\ \gamma)\ (g\ \gamma) && \text{(Figure 3.2)} \\
&= (g\ \gamma) \circ (f\ \gamma) && \text{(Figure 3.9)} \\
&= \mathsf{Hom}(\mathsf{v}_1, t''[\mathsf{p} \circ \mathsf{p}])\ \mathsf{id}_\gamma\ (t\ \gamma)\ (f\ \gamma) && \text{(Figure 3.9)} \\
&= (\mathsf{J}^- g)\ \gamma\ (t\ \gamma)\ (f\ \gamma) && \text{(Figure 3.7)} \\
&= ((\mathsf{J}^- g) \circ (\mathsf{id} ,_+ t ,_+ f))\ \gamma \\
&= (\mathsf{J}^- g)[\mathsf{id} ,_+ t ,_+ f]\ \gamma.
\end{aligned}$$

So, by (metatheoretic) function extensionality, these two terms have equal object parts. Their morphism parts send morphisms $\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]$ to morphisms in the category $\mathsf{Hom}(t, t'')\ \gamma_1$. But this is a discrete category: its morphisms are metatheoretic identity terms. Thus, by metatheoretic UIP, these morphism parts must also agree. So conclude that the two terms are equal. □

As with the other axioms/constructs we've introduced, we're justified in asserting

$$\begin{aligned}
\mathsf{Hom}(t[\mathsf{p}\circ\mathsf{p}],\mathsf{v}_1) &: (\gamma_{01}:\Gamma[\gamma_0,\gamma_1]) \to (a_{01}:(A\gamma_1)\ [A\ \gamma_{01}\ a_0, a_1]) \to \\
&\qquad (A\gamma_0)\ [t\ \gamma_0, a_0] \to (A\gamma_1)[t\ \gamma_1, a_1] \\
\mathsf{Hom}(t[\mathsf{p}\circ\mathsf{p}],\mathsf{v}_1)\ \gamma_{01}\ a_{01}\ x_0 &:= a_{01} \circ A\ \gamma_{01}\ x_0 \circ t(\gamma_{01}) \\
\mathsf{Hom}(\mathsf{v}_1,t''[\mathsf{p}\circ\mathsf{p}]) &: (\gamma_{01}:\Gamma[\gamma_0,\gamma_1]) \to (a_{10}:(A\gamma_1)[a_1, A\ \gamma_{01}\ a_0]) \to \\
&\qquad (A\gamma_0)\ [a_0, t''\ \gamma_0] \to (A\gamma_1)\ [a_1, t''\ \gamma_1] \\
\mathsf{Hom}(\mathsf{v}_1,t''[\mathsf{p}\circ\mathsf{p}])\ \gamma_{01}\ a_{10}\ x_0 &:= t''(\gamma_{01}) \circ A\ \gamma_{01}\ x_0 \circ a_{10}
\end{aligned}$$

Figure 3.9: The relevant morphism parts for the proof of Proposition 3.2.2. Note that these are terms in $\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(-t'[\mathsf{p}], \mathsf{v}_0)$ and $\Gamma \rhd^+ A^- \rhd^+ \mathsf{Hom}(\mathsf{v}_0, t''[\mathsf{p}])$, respectively; the morphism part of these terms take one fewer argument than would be expected, per Remark 3.1.2.

this as an axiom of directed type theory.

Axiom (Composition Coincidence). Given composable hom-terms $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t'))$ and $g : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', t''))$, the following judgmental equality holds.

$$(\mathsf{J}^+ f)[\mathsf{id} ,_+ t' ,_+ g] = (\mathsf{J}^- g)[\mathsf{id} ,_+ t ,_+ f]$$

As an upshot, this makes *both* the left- and right-unit laws of composition hold judgmentally (Figure 3.10): to prove $\mathsf{refl} \cdot g$ equals g, we take composition to be defined by slice path induction, so $\mathsf{refl} \cdot g = g$ by Slice β; and likewise we treat composition as defined by slice path induction to get that $f \cdot \mathsf{refl} = f$ by Coslice β. This is perhaps "too nice", and might have to be removed in order for this directed type theory to have favorable computational properties. But for the sake of pen-and-paper reasoning (particularly some of the constructions of the next chapter), it is a convenient principle to adopt.

3.2.2 Σ-Types

Let's now shift our attention to Σ-types, recalling the Pair/Unpair notation of Proposition 1.2.14 and the negative counterpart from Remark 2.3.31. By the latter, we know that the negation operation distributes over Pair:

$$-(\mathsf{Pair}(s', t')) = \mathsf{Pair}^-(-s', -t')$$

and over the projection term formers, e.g.

$$\mathsf{pr}_1(-S) = -\mathsf{pr}_1^-(S)$$

for $S : \mathsf{Tm}(\Gamma, (\Sigma AB)^-)$. Our goal will be to develop the directed analogue of the characterization of the identity types of Σ-types in undirected type theory. That is, we want to take the statement

directedTT

```
Csl : {t : Tm(Γ, A⁻)} ⟶ Ty (Γ ▷⁺ A)
Csl {t} := Hom(t[ p ], v₀)

Sl : {t' : Tm(Γ, A)} ⟶ Ty (Γ ▷⁺ A⁻)
Sl {t'} := Hom(v₀, t'[ p ])

_·_ : Tm(Γ, Hom(t,t')) ⟶ Tm(Γ, Hom(-t',t'')) ⟶ Tm(Γ, Hom(t,t''))
f · g := (f ·v₀) [ id ,+ t'' ,+ g ]   -- or  (v₀· g) [ id ,+ t ,+ f ]

r-unit : (g : Tm(Γ, Hom(-t',t''))) ⟶ Tm(Γ, Id(refl_{-t'} · g, g))
r-unit g := refl_g     -- by J⁻β, refl · g = g

l-unit : (f : Tm(Γ, Hom(t,t'))) ⟶ Tm(Γ, Id(f · refl_{t'}, f))
l-unit f := refl_f     -- by J⁺β, f · refl = f

assoc : (f : Tm(Γ, Hom(t,t')))
   ⟶ (g : Tm(Γ, Hom(-t',t'')))
   ⟶ (h : Tm(Γ, Hom(-t'',t''')))
   ⟶ Tm(Γ, Id(f · (g · h), (f · g) · h))
assoc f g h := (J⁺_{-t',S} refl_{f·g})[ id ,+ t''' ,+ h ]
  where
    S : Ty (Γ ▷⁺ A ▷⁺ Hom(t[ p ], v₀))
    S := Id( (J⁺_{-t',Csl{t}} f) [ p ,+ (J⁺_{-t'',Csl{-t'}} g) ] ,
             (J⁺_{-t'',Csl{t}} (f · g))
           )
```

Figure 3.10: Synthetic category laws for composition.

$$\mathsf{Id}(S, S') \simeq \sum_{\alpha :\, \mathsf{Id}(\mathsf{pr}_1(S), \mathsf{pr}_1(S'))} \mathsf{Id}(\mathsf{tr}\ \alpha\ (\mathsf{pr}_2\ S), \mathsf{pr}_2(S')) \tag{3.2.3}$$

and come to an appropriate analogue in our directed type theory. We'll loosely follow the presentation of Rijke [Rij22, Section 9.3]. In that presentation, the left-to-right direction of the above equivalence is defined by path induction, sending refl to the pair (refl, refl), and then the tools of Homotopy Type Theory are used to construct an inverse for this map. We'll also define the one map by *directed* path induction, but, given our more impoverished setting, we'll instead just axiomatically assert an inverse in the metatheory (justified by the category model, of course).

The precise formulation is given in Figure 3.11, but we explain it more carefully here. Fix some $S\colon \mathsf{Tm}(\Gamma, (\Sigma AB)^-)$; we then construct a type $\mathsf{ObsHom}(S, \mathsf{v}_0)$ in context $\Gamma \rhd^+ \Sigma AB$ of "observational homs"[11] between S and the variable v_0. This type is a

[11] By analogy to how $\sum_{\alpha :\, \mathsf{Id}(\mathsf{pr}_1(S), \mathsf{pr}_1(S'))} \mathsf{Id}(\mathsf{tr}\ \alpha\ (\mathsf{pr}_2\ S), \mathsf{pr}_2(S'))$ is the type of witnesses to the

Σ-type, the type of dependent pairs whose first component is a hom between the first components of S and v_0, and whose second component is a hom between their second components, *over* the first hom. That is, we have to transport $\mathsf{pr}_2^-(S)$, a term of type along the first component (the hom between $\mathsf{pr}_1^-(S)$ and $\mathsf{pr}_1(\mathsf{v}_0)$) to get it into the same type as $\mathsf{pr}_2(\mathsf{v}_0)$ so we can speak of homs between them. This is not a transport in the precise sense of Definition 3.1.7, as the hom we're transporting along is a variable;[12] but nonetheless we can perform this transport: since $\mathsf{pr}_2^-(S)$ is a term in the neutral context Γ (of type $B[\mathsf{id} \,,_+ \, -\mathsf{pr}_1^-(S)]^-$), we can use it as the method for (coslice) directed path induction based at $\mathsf{pr}_1^-(S)$:

$$\mathsf{J}^+ \, \mathsf{pr}_2^-(S) \quad : \mathsf{Tm}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}((\mathsf{pr}_1^- S)[\mathsf{p}], \mathsf{v}_0), \quad B[\mathsf{p}]^-).$$

Then, we can substitute in $\mathsf{pr}_1(\mathsf{v}_0)$ in for the A term and the first component in for the hom:

$$\begin{array}{ccc} (\mathsf{J}^+ \, \mathsf{pr}_2^-(S))[\ldots] : B[\mathsf{p} \circ \mathsf{p} \,,_+ \, (\mathsf{pr}_1 \mathsf{v}_0)[\mathsf{p}]]^- & & \Gamma \rhd^+ \Sigma AB \rhd^+ \mathsf{Hom}((\mathsf{pr}_1^-(S))[\mathsf{p}], \mathsf{pr}_1 \mathsf{v}_0) \\ \uparrow & & \downarrow{\scriptstyle (\mathsf{p} \circ \mathsf{p} \,,_+ \, (\mathsf{pr}_1 \mathsf{v}_0)[\mathsf{p}] \,,_+ \, \mathsf{v}_0)} \\ \mathsf{J}^+ \, \mathsf{pr}_2^-(S) : B[\mathsf{p}]^- & & \Gamma \rhd^+ A \rhd^+ \mathsf{Hom}((\mathsf{pr}_1^-(S))[\mathsf{p}], \mathsf{v}_0). \end{array}$$

Since $(\mathsf{pr}_2 \mathsf{v}_0)[\mathsf{p}]$ has type $B[\mathsf{p} \,,_+ \, \mathsf{pr}_1 \mathsf{v}_0][\mathsf{p}] = B[\mathsf{p} \circ \mathsf{p} \,,_+ \, (\mathsf{pr}_1 \mathsf{v}_0)[\mathsf{p}]]$, it makes sense to form the type

$$\mathsf{Hom}((\mathsf{J}^+ \, \mathsf{pr}_2^- S)[\mathsf{p} \circ \mathsf{p} \,,_+ \, (\mathsf{pr}_1 \mathsf{v}_0)[\mathsf{p}] \,,_+ \, \mathsf{v}_0], \; (\mathsf{pr}_2 \mathsf{v}_0)[\mathsf{p}])$$

and use this as the second component of the type $\mathsf{ObsHom}(S, \mathsf{v}_0)$.

Given some term $S' : \mathsf{Tm}(\Gamma, \Sigma AB)$, we'll write $\mathsf{ObsHom}(S, S')$ for $\mathsf{ObsHom}(S, \mathsf{v}_0)[\mathsf{id} \,,_+ \, S']$. Observe[13] that, once we supply S', the "transport" we did above indeed becomes a genuine transport in the sense of Definition 3.1.7:

$$\mathsf{ObsHom}(S, S') = \Sigma \; (\mathsf{Hom}(\mathsf{pr}_1^- \; S, \mathsf{pr}_1 \; S')) \; (\mathsf{Hom}(\mathsf{tr} \; \mathsf{v}_0 \; ((\mathsf{pr}_2^- \; S)[\mathsf{p}]), \mathsf{pr}_2 \; S'))$$

where the transport is happening in the context $\Gamma \rhd^+ \mathsf{Hom}(\mathsf{pr}_1^- S, \mathsf{pr}_1 S')$, which is neutral because hom-types are neutral. Then, if $\mathsf{Pair}(\varphi, \psi) : \mathsf{Tm}(\Gamma, \mathsf{ObsHom}(S, S'))$, this means that

$$\begin{aligned} \varphi &: \mathsf{Tm}(\Gamma, \mathsf{Hom}(\mathsf{pr}_1^- S, \mathsf{pr}_1 S')) \\ \psi &: \mathsf{Tm}(\Gamma, \mathsf{Hom}(\mathsf{tr} \; \varphi \; (\mathsf{pr}_2^- \; S), \mathsf{pr}_2 \; S') \end{aligned}$$

"observational equality" between S and S' above.

[12] We're forming the type of the second component in the context

$$\Gamma \rhd^+ \Sigma AB \rhd^+ \mathsf{Hom}((\mathsf{pr}_1^- \; S)[\mathsf{p}], \mathsf{pr}_2(\mathsf{v}_0))$$

and trying to transport along the de Bruijn index v_0 in this context.

[13] Care of the calculation in Figure 3.12.

In particular, consider $S' = -S$. Since tr refl $x = x$ by Coslice β, we observe that Pair(refl, refl) is a valid term of type ObsHom$(S, -S)$. We can use this to define a map

$$\mathsf{Hom{-}to{-}pair} \colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(S, S')) \to \mathsf{Tm}(\Gamma, \mathsf{ObsHom}(S, S'))$$

by directed path induction on S'; in fact, we can use Definition 3.1.7 itself this time: given some $h \colon \mathsf{Tm}(\Gamma, \mathsf{Hom}(S, S'))$, we have

$$\mathsf{Hom{-}to{-}pair}\ h := \mathsf{tr}\ h\ (\mathsf{Pair}(\mathsf{refl}, \mathsf{refl})) \quad : \mathsf{Tm}(\Gamma, \mathsf{ObsHom}(S, S')).$$

Our characterization of the hom-types of Σ-types is the assertion[14] that Hom−to−pair is a bijection, whose inverse is called, appropriately, pair−to−Hom. That is, the *hom-types of Σ-types are given observationally*—to construct a directed path in ΣAB between two points, it is necessary and sufficient to construct a directed path in A between their first components and a "heterogeneous directed path in B *over the first path*" between their second components.

Axiom (Sigma Observation). The map Hom−to−pair sending terms h of type $\mathsf{Hom}_{\Sigma AB}(S, S')$ to tr h (Pair(refl, refl)) is a bijection.

There are a number of different ways we could have stated this principle, some weaker and some stronger. We could, for instance, have made the much stronger assertion that $\mathsf{Hom}(S, S')$ and $\mathsf{ObsHom}(S, S')$ are *definitionally* equal types. Or we could've pushed more of this statement into the object language, such as by requiring pair−to−Hom to be the inverse of Hom−to−pair only up to *propositional* equality,[15] or perhaps by trying to internalize Hom−to−pair as a function in the object theory rather than the metatheory. However, we think that the statement as given strikes the appropriate object language/metalanguage balance, and falls squarely in the observational type theory tradition.

This axiom is a key piece of the—*yet unfinished*—task of stating a **directed structure identity principle** in the present framework. A directed SIP, or perhaps we could say *structure morphism principle*, asserts that the directed paths in a type of "structures" correspond to the classical *homomorphisms*, the structure-preserving functions. A "type of structures" means, roughly, a Σ-type of one or more sets (elements of the universe of sets—see below), plus elements of those sets and functions on those sets. The above axiom allows us to say that the hom-types of "structures" are given by homs on each of their components; all that remains to get to a structure morphism principle is showing that this yields the classical notion of homomorphism. Such a principle has recently been achieved in the simplicial style of directed type theory [GWB24], but there are still some obstacles preventing us from stating it here. We do get that directed paths in the universe of sets are indeed functions, and can get a directed SIP for pointed sets

[14] Represented in pseudoAgda as a **postulate**.

[15] Both h and pair−to−Hom(Hom−to−pair h) are elements of $\mathsf{Hom}(S, S')$, which is always neutral in (1,1)-directed type theory and hence has identity types. But we don't know *a priori* that $\mathsf{ObsHom}(S, S')$ is neutral—to assert the other direction propositionally, we would need to stipulate that neutral types are closed under Σ (which is probably a reasonable requirement anyways), or adopt one of the strategies discussed in Remark 3.1.11 to have identity types for non-neutral types. But we leave this question unexplored.

too—see below—but we aren't yet able to characterize the hom-types of *function* types, ruling out all but the most basic kinds of "structure". We hope to rectify this in the future.

3.2.3 Universes

So far, we have developed the directed analogues of all the key constructs of Martin-Löf Type Theory, except one: *universes*. As discussed in section 1.2, a universe is a type of *codes* for types, and care must be taken to avoid the paradoxes that arise from a universe containing (a code for) itself. Therefore, we restrict our attention in the present work to a universe of *sets*—interpreted in the category model as the category of sets and functions. We suspect that a proper development of (2,1)-directed type theory and its semantics in 2-categories would permit the introduction of a universe of (1-)categories, but this is beyond our scope here.

Definition 3.2.4. For some $\Gamma : \mathsf{Con}$, write

- $\mathsf{Sets}\ \Gamma$ for the set of **sets** in Γ (i.e. the subset of $\mathsf{NeutTy}\ \Gamma$ consisting of types whose hom-types are propositions);
- $\mathsf{Props}\ \Gamma$ for the set of **propositions** in Γ (i.e. the subset of $\mathsf{NeutTy}\ \Gamma$ consisting of types all of whose terms are identical).

Axiom (Set Universe). There is a type $\mathsf{Set} : \mathsf{Ty}\ \Gamma$ (with $\mathsf{Set}[\sigma] = \mathsf{Set}$) equipped with a bijection (which is appropriately stable under substitution):

$$\mathsf{El} : \mathsf{Tm}(\Gamma, \mathsf{Set}) \xrightarrow{\sim} \mathsf{Sets}\ \Gamma.$$

Axiom (Prop Universe). There is a type $\mathsf{Prop} : \mathsf{Ty}\ \Gamma$ (with $\mathsf{Prop}[\sigma] = \mathsf{Prop}$) equipped with a bijection (which is appropriately stable under substitution):

$$\mathsf{El} : \mathsf{Tm}(\Gamma, \mathsf{Prop}) \xrightarrow{\sim} \mathsf{Props}\ \Gamma.$$

In (1,1)-directed type theory, all hom-types are indeed hom-*sets*, reflecting our 1-categorical model theory into the syntax. Accordingly, we can regard the hom-type former as going into $\mathsf{Sets}\ \Gamma$ rather than $\mathsf{Ty}\ \Gamma$, and therefore hom-types are encoded in the universe of sets. We adopt the following notation.

Definition 3.2.5. For any $A : \mathsf{Ty}\ \Gamma$, $t : \mathsf{Tm}(\Gamma, A^-)$, and $t' : \mathsf{Tm}(\Gamma, A)$, write

$$\mathsf{HomSet}_A(t, t') : \mathsf{Tm}(\Gamma, \mathsf{Set})$$

for the set such that

$$\mathsf{El}(\mathsf{HomSet}_A(t, t')) = \mathsf{Hom}_A(t, t').$$

This gives us a source of sets, meaning we can write functions into Set. Now, remember our category-theoretic understanding of this directed type theory: types A are categories, A^- is the opposite category of A, and Set is the category of sets. In

the next chapter, we'll also demonstrate that the functions in this theory are synthetic *functors*—any term of type $A \to B$ automatically comes equipped with a "morphism part", functorially sending directed paths in A to directed paths in B. Therefore, we can regard terms of type $A^- \to \mathsf{Set}$ as *synthetic presheaves on* A, contravariant functors taking values in the category of sets. Our lack of a suitable notion of synthetic natural transformation[16] prevents us from giving a thorough development of the category of presheaves, we are able to make the following key definition.

Definition 3.2.6. Given a type $A : \mathsf{Ty}\ \Gamma$ for $\Gamma : \mathsf{NeutCon}$, define the **Yoneda embedding**

$$\mathsf{y} : \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to A[\mathsf{e}]^- \to \mathsf{Set})$$

by

$$\mathsf{y} := \mathsf{lam}(\mathsf{lam}((\mathsf{HomSet}(\mathsf{v}_0, \mathsf{v}_1))[\mathsf{ee}\ \{A, A^-\}])).$$

Now we turn our attention to the directed extensionality principle for Set, the fact that it's supposed to represent the category of sets and *functions*. Accordingly, we need to characterize the hom-types of Set as being given by functions. This will be the principle of **directed univalence** for this (directed) universe.

Axiom (Directed Univalence). The function Hom–to–func sending

$$h : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-X, Y)) \quad \mapsto \quad \mathsf{tr}\ h\ \mathsf{I}_{\mathsf{El}\ X} : \mathsf{Tm}(\Gamma, (\mathsf{El}\ X)[\mathsf{e}] \to \mathsf{El}\ Y)$$

is a bijection.

Following [HS95, Section 5.4], we remark that this principle of *directed universe extensionality* is inconsistent with the "uniqueness of homs" principle mentioned at the beginning of this chapter (the directed analogue of UIP), assuming Set contains some nontrivial set. For instance, suppose we encode the boolean type $\mathbb{2}$ (which is a set) by a term $2 : \mathsf{Tm}(\Gamma, \mathsf{Set})$, i.e. so $\mathsf{El}(2) = \mathbb{2}$. Then we can define two distinct terms of type $\mathbb{2}[\mathsf{e}] \to \mathbb{2}$, e.g. $K_{\mathsf{tt}} := \mathsf{lam}^+(\mathsf{tt}[\mathsf{p}])$ and $K_{\mathsf{ff}} := \mathsf{lam}^+(\mathsf{ff}[\mathsf{p}])$. Thus, obtain two distinct terms—call them k_{tt} and k_{ff}, respectively—of type $\mathsf{Hom}(2, 2)$ since func–to–Hom is injective. If UHP were stated internally, i.e. we had some term $\alpha : \mathsf{Tm}(\Gamma, \mathsf{Id}(k_{\mathsf{tt}}, k_{\mathsf{ff}}))$, then we can construct an identity $\mathsf{Id}(\mathsf{tt}, \mathsf{ff})$ as the composite of the following identities (let x be an arbitrary term of type $\mathbb{2}$, say, tt).

- We have a definitional equality between $K_{\mathsf{tt}}\ \$^+\ x$ and tt, and likewise for $K_{\mathsf{ff}}\ \$^+\ x$

[16] We do have natural transformations in our theory: the terms of type $A \to B$ are functors from the category A to the category B, and the type $A \to B$, like any type, comes equipped with hom-types. So if $F : \mathsf{Tm}(\Gamma, (A \to B)^-)$ and $G : \mathsf{Tm}(\Gamma, A \to B)$, then $\mathsf{Hom}(F, G)$ is the type of natural transformations between F and G, and we can talk about the identity natural transformation (refl) and (vertical) composition of natural transformations, etc. But when we say that we "lack a suitable notion of synthetic natural transformation", we mean that it is not possible in the present theory to manipulate these natural transformations *component-wise*: given α of $\mathsf{Hom}(F, G)$ and some object t of A, we don't have syntax for obtaining the component α_t of type $\mathsf{Hom}_B(F\ \$^+\ t, G\ \$^+\ t)$, nor do we have a mechanism for defining such a natural transformation component-wise. Such syntax would require a way to have variables appearing negatively and positively, which isn't possible in our current framework.

and ff; so the respective refls are of type

$$\mathsf{Id}(\mathsf{tt},\ K_{\mathsf{tt}}\ \$^{+}\ x) \qquad \text{and} \qquad \mathsf{Id}(K_{\mathsf{ff}}\ \$^{+}\ x,\ \mathsf{ff}).$$

- By Proposition 3.2.7 below, obtain identities

$$\mathsf{Id}(K_{\mathsf{tt}}\ \$^{+}\ x,\ \mathsf{tr}\ k_{\mathsf{tt}}\ x) \qquad \text{and} \qquad \mathsf{Id}(\mathsf{tr}\ k_{\mathsf{ff}}\ x,\ K_{\mathsf{ff}}\ \$^{+}\ x).$$

- And the key step: by transporting[17] $\mathsf{refl}_{\mathsf{tr}\ k_{\mathsf{tt}}\ x}$ along α, obtain an identity

$$\mathsf{Id}(\mathsf{tr}\ k_{\mathsf{tt}}\ x,\ \mathsf{tr}\ k_{\mathsf{ff}}\ (x[\mathsf{p}])).$$

So UHP plus Directed Univalence Axiom implies that $\mathsf{Id}(\mathsf{tt},\mathsf{ff})$. We could also employ Set as the prototypical non-neutral type in Theorem 3.1.20 (in place of the walking arrow $\vec{2}$) to argue that symmetry is independent of directed type theory because the interpretation of the universe Set in the category model, i.e. the actual category of sets, is not a groupoid. That is: in the syntax of directed type theory plus a universe Set containing, say, codes 0 and 1 for the sets $\emptyset$ and $\mathbb{1}$, respectively, we cannot prove symmetry of homs: if we could, then the interpretation of this syntax into the category model would imply the existence of a morphism in the category of sets from a singleton set (the interpretation of the term 1 in the category model) to the empty set (the interpretation of 0), of which there are none. Similarly, we could carry out the above boolean-based argument at the metatheoretic level to prove that the syntax of directed type theory plus a universe Set containing a code 2 for $\mathbb{2}$ cannot possibly prove UHP, because the category model's interpretation of Set has four distinct terms of type $\mathsf{Hom}(2,2)$.

Finally, let's put all this together and take the first step towards a *directed structure identity principle* in this theory. Consider the type of *pointed sets*, which we can define as

$$\mathsf{Set}_{\bullet} := \Sigma\ \mathsf{Set}\ (\mathsf{El}\ \mathsf{v}_0).$$

Now, given $X, Y : \mathsf{Tm}(\Gamma, \mathsf{Set})$ and $x_0 : \mathsf{Tm}(\Gamma, \mathsf{El}\ X)$ and $y_0 : \mathsf{Tm}(\Gamma, \mathsf{El}\ Y)$, it is clear what a hom of type

$$\mathsf{Hom}_{\mathsf{Set}_{\bullet}}(\mathsf{Pair}^{-}(-X, x_0), \mathsf{Pair}(Y, y_0))$$

ought to be: it should be a function F from the set X to the set Y and an identity $\mathsf{Id}(F\ \$^{+}\ x_0, y_0)$. Applying Sigma Observation Axiom, we see that the hom-type above is equivalent to $\mathsf{ObsHom}(\mathsf{Pair}^{-}(-X, x_0), \mathsf{Pair}(Y, y_0))$, i.e.

$$\Sigma\quad(\mathsf{Hom}_{\mathsf{Set}}(-X, Y))\quad(\mathsf{Id}(\mathsf{tr}\ \mathsf{v}_0\ (x_0[\mathsf{p}]), y_0)).$$

The second component becomes an identity type because it's a hom in the neutral type $\mathsf{El}(Y)$. Now, transform the first component by way of Directed Univalence Axiom: the hom-type above now becomes

$$\Sigma\quad((\mathsf{El}\ X)[\mathsf{e}] \to \mathsf{El}\ Y)\quad(\mathsf{Id}(\mathsf{tr}\ (\mathsf{func{-}to{-}Hom}\ \mathsf{v}_0)\ (x_0[\mathsf{p}]), y_0)).$$

[17] In the context Γ, with motive

$$\mathsf{Id}((\mathsf{tr}\ k_{\mathsf{tt}}\ x)[\mathsf{p}], \mathsf{tr}\ \mathsf{v}_0\ x) \quad : \mathsf{Ty}(\Gamma \rhd^{+} \mathsf{Hom}(-2, 2)).$$

Here, we rely on the neutrality of $\mathsf{El}\ X)[\mathsf{e}] \to \mathsf{El}\ Y$ in order for the transport and application of func–to–Hom in the extended context to be legitimate. Technically, we didn't add this as an axiom, but we could.[18] Now we just need to prove that $\mathsf{tr}\ (\text{func–to–Hom}\ \mathsf{v}_0)\ x_0$ is the same thing as $\mathsf{v}_0\ \$^+\ x_0$.

Proposition 3.2.7. *For every* $\Gamma : \mathsf{NeutCon}$, $X, Y : \mathsf{Tm}(\Gamma, \mathsf{Set})$, $x_0 : \mathsf{Tm}(\Gamma, \mathsf{El}\ X)$ *and* $F : \mathsf{Tm}(\Gamma, (\mathsf{El}\ X)[\mathsf{e}] \to \mathsf{El}\ Y)$, *there is an identity*

$$\mathsf{Id}(F\ \$^+\ x_0, \mathsf{tr}\ (\text{func–to–Hom}\ F)\ x_0).$$

Proof. By Directed Univalence Axiom, write F as $\text{Hom–to–func}(\varphi)$ for some $\varphi : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-X, Y))$. We prove the corresponding claim,

$$\mathsf{Id}((\text{Hom–to–func}\ \varphi)\ \$^+\ x_0, \mathsf{tr}\ \varphi\ x_0)$$

for all φ by induction: if $\varphi = \mathsf{refl}_{-X}$, then this just amounts to a witness of the identity

$$\mathsf{Id}(\mathsf{I}_{\mathsf{El}X}\ \$^+\ x_0, x_0)$$

so we use refl_{x_0}. □

In this case, we're actually applying Proposition 3.2.7 with the context Γ being $\Gamma \rhd^+ (\mathsf{El}\ X)[\mathsf{e}] \to \mathsf{El}\ Y$, the set X being $X[\mathsf{p}]$, F being v_0, and so on. So, finally, we have our structure morphism principle:

Principle (Structure Morphism Principle—Pointed Sets).

$$\begin{aligned}&\mathsf{Tm}(\Gamma, \mathsf{Hom}(\mathsf{Pair}^-(-X, x_0), \mathsf{Pair}(Y, y_0)))\\ &\cong \mathsf{Tm}(\Gamma, \Sigma\ ((\mathsf{El}\ X)[\mathsf{e}] \to \mathsf{El}\ Y)\ (\mathsf{Id}(\mathsf{v}_0\ \$^+\ x_0, y_0))).\end{aligned}$$

Of course, this is the simplest possible case for a directed SIP: we don't yet have a characterization of the hom-types of function types (a directed function extensionality), preventing us from doing the same analysis for, say, $\mathfrak{N}$-algebras, or monoids, or categories, or CwFs. But the development so far provides a proof-of-concept, pending further development.

This concludes our discussion of observational/extensionality laws in directed type theory. We have taken the fundamental type formers from Martin-Löf Type Theory—Σ-types, Π-types, identity types, and universes—and considered their analogue in directed type theory. For all of these besides Π-types, we were able to characterize their hom-types:

- the hom-types *of hom-types* provide syntax for synthetic higher category theory; in $(m, 1)$-directed type theory, we stipulate that hom-types are neutral, thus homs-between-homs are rather *identities* between homs—matching the 1-category

[18] A more principled development—which we don't undertake here—would be to stipulate that Set comes equipped with its own internal Π-types and Σ-types. In particular, for every $X, Y : \mathsf{Tm}(\Gamma, \mathsf{Set})$, there's some $Y^X : \mathsf{Tm}(\Gamma, \mathsf{Set})$. Then we could say that $\mathsf{El}\,(Y^X) = (\mathsf{El}\ X)[\mathsf{e}] \to (\mathsf{El}Y)$, and thereby conclude that the right-hand side is a set (since the left-hand side is), and, in particular, neutral.

theoretic practice of considering equations between the elements of hom-sets;
- the hom-types of Σ-types are given observationally, as a Σ-type of hom-types between the respective components;
- the hom-type between two elements of the universe Set are given by *functions* on the sets they denote—a statement of *univalence* for the (1,1)-truncated setting.

The latter of two were asserted as inverses to operations internally definable by transport (i.e. by directed path induction), as is the practice in homotopy type theory. As mentioned, we do not give a characterization for Π-types—a *directed function extensionality*—in the present work; there are more difficult polarity problems (involving *dinaturality*) which must be solved to make such a treatment possible.

directedTT

```
module SigmaObserve {Γ : NeutCon}{A : Ty Γ}{B : Ty (Γ ▷⁺ A)}
            (S : Tm(Γ, Σ(A,B)⁻))
 where

  ObsHom(S,v₀) : Ty (Γ ▷⁺ Σ(A,B))
  ObsHom(S,v₀) :=
    Σ (Hom( (pr₁⁻ S)[ p ] , pr₁(v₀) )) (Hom( trpr₂S , pr₂(v₀)[ p ]))
      where
        trpr₂S : Tm(Γ ▷⁺ Σ(A,B) ▷⁺ Hom( (pr₁⁻ S)[ p ] , pr₁(v₀) ),
                    B[ p ∘ p ,₊ pr₁ v₁ ]⁻)
        trpr₂S := (J⁺ pr₂⁻(S))[ p ∘ p ,₊ (pr₁ v₁) ,₊ v₀ ]

  obsreflS : Tm(Γ, ObsHom(S,v₀)[ id ,₊ -S ])
  obsreflS := Pair(refl,refl)

  variable
    S' : Tm(Γ, Σ(A,B))

  Hom-to-pair : Tm(Γ, Hom(S,S')) ⟶ Tm(Γ, ObsHom(S,v₀) [ id ,₊ S' ])
  Hom-to-pair h := tr_ObsHom(S,v₀) h obsreflS

  postulate
    pair-to-Hom :
      Tm(Γ, ObsHom(S,v₀) [ id ,₊ S' ]) ⟶ Tm(Γ, Hom(S,S'))
    pairHomβ :
      (h : Tm(Γ, Hom(S,S'))) ⟶ pair-to-Hom(Hom-to-pair h) = h
    pairHomη :
      (O : Tm(Γ, ObsHom(S,v₀) [ id ,₊ S' ])) ⟶
        Hom-to-pair(pair-to-Hom O) = O
```

Figure 3.11: Statement of Sigma Observation Axiom in directed type theory syntax.

$$
\begin{aligned}
&(\mathsf{J}^+\ (\mathsf{pr}_2^-S))[\mathsf{p}\circ\mathsf{p}\ ,{+}\ (\mathsf{pr}_1\mathsf{v}_1)\ ,{+}\ \mathsf{v}_0][\mathsf{q}(\mathsf{id}_\Gamma\ ,{+}\ S')] \\
&= (\mathsf{J}^+\ (\mathsf{pr}_2^-S))[\mathsf{p}\circ\mathsf{p}\ ,{+}\ (\mathsf{pr}_1\mathsf{v}_1)\ ,{+}\ \mathsf{v}_0][\mathsf{p}\ ,{+}\ S'[\mathsf{p}]\ ,{+}\ \mathsf{v}_0] \\
&= (\mathsf{J}^+\ (\mathsf{pr}_2^-S))[\mathsf{p}\ ,{+}\ (\mathsf{pr}_1\ S')[\mathsf{p}]\ ,{+}\ \mathsf{v}_0] \\
&= (\mathsf{J}^+\ (\mathsf{pr}_2^-S))[\mathsf{p}\circ\mathsf{p}\circ\mathsf{p}\ ,{+}\ \mathsf{v}_1\ ,{+}\ \mathsf{v}_0][\mathsf{id}_{\Gamma'}\ ,{+}\ (\mathsf{pr}_1\ S')[\mathsf{p}]\ ,{+}\ \mathsf{v}_0] \\
&= (\mathsf{J}^+\ (\mathsf{pr}_2^-S))[\mathsf{q}(\mathsf{p})][\mathsf{id}_{\Gamma'}\ ,{+}\ (\mathsf{pr}_1\ S')[\mathsf{p}]\ ,{+}\ \mathsf{v}_0] \\
&= (\mathsf{J}^+\ ((\mathsf{pr}_2^-S)[\mathsf{p}]))[\mathsf{id}_{\Gamma'}\ ,{+}\ (\mathsf{pr}_1\ S')[\mathsf{p}]\ ,{+}\ \mathsf{v}_0] && \mathsf{J}^+[] \\
&= \mathsf{tr}\ \mathsf{v}_0\ ((\mathsf{pr}_2^-S)[\mathsf{p}]) && \text{(Definition 3.1.7)}
\end{aligned}
$$

Figure 3.12: Calculation of the domain of the second component in $\mathsf{ObsHom}(S, S')$. Here, $\Gamma' := \Gamma \rhd^+ \mathsf{Hom}(\mathsf{pr}_1^-S, \mathsf{pr}_1 S')$.

PSEUDOAGDA

```
Set : Ty Γ
Set γ := Set

El : {Γ : Con} ⟶ Tm(Γ,Set) ⟶ NeutTy Γ
|El(X) γ| := X γ
(El(X) γ) [ x₀ , x₁ ] := (x₀ = x₁)
```

Figure 3.13: Construction of the universe Set in the category model

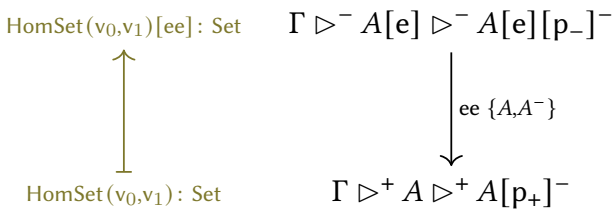


Figure 3.14: The substitution by ee performed in Definition 3.2.6.

directedTT

```
module UniverseObserve {Γ : NeutCon}(X : Tm(Γ, Set))
 where

  ObsHom(–X,v₀) : Ty (Γ ▷⁺ Set)
  ObsHom(–X,v₀) := (El X)[ e ] ⟶ El v₀

  obsreflX : Tm(Γ, ObsHom(–X,v₀)[ id ,₊ X ])
  obsreflX := I_El X  -- Definition 2.3.11

  variable
    Y : Tm(Γ, Set)

  Hom–to–func(v₀) : Tm(Γ ▷⁺ Set ▷⁺ Hom(–X,v₀), ObsHom(–X,v₀)[ p ])
  Hom–to–func(v₀) := J⁺ obsreflX

  Hom–to–func : Tm(Γ, Hom(–X,Y)) ⟶
        Tm(Γ, ObsHom(–X,v₀) [ id ,₊ Y ])
  Hom–to–func h :=  Hom–to–func(v₀)[ id ,₊ Y ,₊ h ]
          --    = tr h obsreflX

  postulate
    func–to–Hom :
      Tm(Γ, ObsHom(–X,v₀) [ id ,₊ Y ]) ⟶ Tm(Γ, Hom(–X,Y))
    funcHomβ :
      (h : Tm(Γ, Hom(–X,Y))) ⟶ func–to–Hom(Hom–to–func h) = h
    funcHomη :
      (F : Tm(Γ, ObsHom(–X,v₀) [ id ,₊ Y ])) ⟶
        Hom–to–func(func–to–Hom F) = F
```

Figure 3.15: Statement of Directed Univalence Axiom in directed type theory syntax.

Chapter 4

Synthetic-Inductive Category Theory

"*Come with me and you'll be*
In a world of pure imagination
...
We'll begin with a spin
Traveling in the world of my creation
What we'll see will defy
Explanation"

Willy Wonka, *Willy Wonka & the Chocolate Factory*

4.1 An Introduction to Informal (1,1)-Directed Type Theory

Our investigation into the category model culminates with a demonstration of its "auto-synthetic" ability: in the category model, and indeed in *any* model of (1,1)-directed type theory, the types are endowed with the structure of *synthetic 1-categories* by virtue of their hom-types. We have already seen the starting-point of this *synthetic category theory* in the foregoing definition of *composition*; our purpose now is to fill out more of the theory. Of course, we cannot hope to be exhaustive—far, far too much has been said about 1-categories in the past eighty years to replicate here—but hopefully we can lay out enough to get a better idea of our theory's overall shape and character, and to guide future work.

Though the present work is not a work of homotopy type theory *per se*, we are certainly working in the HoTT tradition. Of all the great ideas incorporated into the "HoTT Book" [Uni13], perhaps the most genius[1] is its pioneering of *informal type theory*.[2] The concepts and proofs of the HoTT Book are not presented as manipulations

[1] In this author's opinion.

[2] We are told that Peter Aczel was a particularly strong proponent of writing the HoTT Book in this style.

in a formal deductive calculus, but written in an informal, human-readable fashion. As the book notes [Uni13, Introduction], purely-formal presentations of such a theory constitute a significant barrier to entry and hinder the theory's practical utility to working mathematicians—any formal system which aspires to being widely understood and used *must* admit an informal style,[3] which is still *rigorous* (in the sense of being *formalizable*). We seek to apply this lesson to our (1,1)-directed type theory. Therefore, in the present section, we will conduct synthetic 1-category theory in *informal* directed type theory, but assure ourselves of its rigor by formalizing it into our formal syntax. We begin by recounting the content of the previous two chapters in this informal style.

4.1.1 Neutral-Polarized Type Theory

As demonstrated in the preceding chapters, the suitable environment for directed type theory to occur is a *neutral context*. Accordingly, our informal reasoning will take place in some unspecified neutral context. We will never refer to this context itself: it is the ambient environment in which we operate, but we take it for granted. We shall therefore say "let A be a type" and "let $t : A$" to mean $A : \mathsf{Ty}\ \Gamma$ and $t : \mathsf{Tm}(\Gamma, A)$, respectively. As is our privilege when working with neutral Γ, we suppress the distinction between Γ and Γ^- and between $\triangleright^+$ and $\triangleright^-$. For the informal style, we also make the e and ee isomorphisms implicit, so for instance the endo-function type is again written $A \to A$.[4]

Perhaps for some purposes it might be useful to have our ambient, unspecified context consist of some (neutral-typed) variables to which we could actually refer, but in the present work Γ might as well be the empty context. Indeed, we'll adopt the term **closed** to mean that the term or type in question is formed in the ambient context, *not* depending on further free variables. By contrast, an **open** term/type will be one that *does* depend on such variables; for instance, we'll write

$$x : A^-, y : A \vdash M(x, y)\ \mathrm{type}$$

to indicate that $M : \mathsf{Ty}(\Gamma \triangleright^+ A^- \triangleright^+ A[\mathsf{p}])$. The judgment "$\mathrm{neut.\ type}$" will be used to specify that M is a *neutral* type. We'll also use the phrase "type-in-telescope" (and "term-in-telescope") to refer to types like M, which depend on a (nonempty) telescope of free variables. The lighter color of x and y serves as a visual reminder that x and y (and any term containing them) are *not* closed terms.

This distinction between closed and open will be critical for properly handling matters of polarity: when dealing with open terms, we must carefully attend to the polarity in order to respect variance. We will still keep track of variance by annotating the types as A^- or A (with $(A^-)^-$ still equal to A). Our closed terms exist in a neutral

[3] The classical example of this is Zermelo-Fraenkel set theory. Most mathematicians, if pressed, would cite this theory as the ultimate foundation of their work. But only a tiny portion of these mathematicians work with the formal deductive calculus of ZF(C); they're able to claim that they work in this theory because it has an informal style—we can reason about sets, elements, subsets, and so on, using our natural language. For the vast majority of contemporary mathematics supposedly conducted in ZFC, the task of actually formalizing it into the formal deductive calculus is left undone—but, if the mathematics is rigorous enough, we're convinced it *could* be done.

[4] Though in formalizing it, we're still obliged to write $A[\mathsf{e}] \to A$ if $A : \mathsf{Ty}\ \Gamma$ or $A \to A[e^-]$ if $A : \mathsf{Ty}\ \Gamma^-$.

context, and can thus be freely converted between polarities:

$$\frac{t : A^-}{-t : A} \qquad \frac{t : A^-}{-(-t) = t} \tag{Term Negation}$$

As in the formal case, we are only able to negate *open* terms if they meet stringent neutrality requirements. We encapsulate this as the following principle.

Principle (Var-Neg). An open term

$$x_1 : A_1, \ldots, x_n : A_n \vdash t(x_1, \ldots, x_n) : A(x_1, \ldots, x_n)$$

can only be negated (forming $-t : A^-$ in the same telescope) if every variable occurring in t (and in A) is of a neutral type.

The formal reasoning behind this is that the term-negation operation is only defined for neutral contexts, in this case, either Γ or some neutral extension of Γ. If "every variable occurring in" $t : A$ is neutral, this means that the judgment $t : A$ can be made in a neutral context—the one consisting of Γ extended by just those variables actually appearing in $t : A$—and then weakened into whatever other context is needed. For instance, consider the term-in-telescope

$$x_1 : A_1, x_2 : A_2, x_3 : A_3(x_1, x_2) \vdash x_2 : A_2$$

where A_1 and A_3 are *not* neutral, but A_2 *is*, and moreover A_2 doesn't contain x_1. Formally, this means

$$A_1 : \mathsf{Ty}\ \Gamma \quad \text{and} \quad A_2 : \mathsf{NeutTy}\ \Gamma \quad \text{and} \quad A_3 : \mathsf{Ty}(\Gamma \rhd^+ A_1 \rhd^+ A_2[\mathsf{p}]).$$

Now, there's still a way to negate the term

$$\mathsf{v}_1 : \mathsf{Tm}(\Gamma \rhd^+ A_1 \rhd^+ A_2[\mathsf{p}] \rhd^+ A_3, A_2[\mathsf{p} \circ \mathsf{p} \circ \mathsf{p}])$$

to get a term of type $A_2[\mathsf{p} \circ \mathsf{p} \circ \mathsf{p}]^-$. Namely, we perform the negation in the neutral context $\Gamma \rhd^+ A_2$:

$$-\mathsf{v}_0 : \mathsf{Tm}(\Gamma \rhd^+ A_2, A_2[\mathsf{p}]^-)$$

and then weaken it to the desired context

$$(-\mathsf{v}_0)[\mathsf{q}(\mathsf{p}_{A_1})][\mathsf{p}_{A_3}] : \mathsf{Tm}(\Gamma \rhd^+ A_1 \rhd^+ A_2[\mathsf{p}] \rhd^+ A_3, A_2[\mathsf{p} \circ \mathsf{p} \circ \mathsf{p}]^-).$$

Var-Neg articulates exactly when this kind of trick is possible: if A_2 "contained" the variable $x_1 : A_1$, that is, A_2 was only a type in context $\Gamma \rhd^+ A_1$ and not a type in context Γ weakened into $A_2[\mathsf{p}] : \mathsf{Ty}(\Gamma \rhd^+ A_1)$, then the above trick wouldn't be possible—the expression $\Gamma \rhd^+ A_2$ wouldn't denote a well-formed context. And likewise if we wanted to negate some term

$$x_1 : A_1, x_2 : A_2, x_3 : A_3 \vdash s_2(x_1, x_2, x_3) : A_2$$

that contained the variables x_1 and x_3, i.e. could not be expressed as

$$x_2 : A_2 \vdash s_2(x_2) : A_2.$$

Remark 4.1.1. Suppose we have a term-in-telescope

$$x_1 : A_1, x_2 : A_2, x_3 : A_3(x_1, x_2) \vdash t(x_2) : T(x_2)$$

which, pursuant to Var-Neg, *can* be negated by to obtain $-t(x_2) : T(x_2)^-$. Formally, $t : \mathsf{Tm}(\Gamma \rhd^+ A_2, T)$ and the term-in-telescope is actually

$$t[\mathsf{q}(\mathsf{p}_{A_1})][\mathsf{p}_{A_3}] : \mathsf{Tm}(\Gamma \rhd^+ A_1 \rhd^+ A_2[\mathsf{p}] \rhd^+ A_3, T[\mathsf{q}(\mathsf{p}_{A_1})][\mathsf{p}_{A_3}])$$

Then, given closed terms $s_1 : A_1$, $s_2 : A_2$, and $s_3 : A_3(s_1, s_2)$, we can substitute these in for x_1, x_2, x_3:

$$-t(s_2) : T(s_2)^-.$$

Now, in formalizing this into the syntax of (1,1)-directed type theory, there are two, potentially different, terms to which $-t(s_2)$ could refer.

- We could be *eager* in performing the negation, that is, negate the term by the above-mentioned procedure first and then substitute in the terms. By this logic, the term $-t(s_2)$ formally translates to

  $$\big((-t)[\mathsf{q}(\mathsf{p}_{A_1})][\mathsf{p}_{A_3}]\big)[\mathsf{id} ,_+ s_1 ,_+ s_2 ,_+ s_3].$$

  The negation operation here, we emphasize once more, is happening in the neutral context $\Gamma \rhd^+ A_2$.
- Alternatively, we could be *lazy* in performing the negation, i.e. treat any negations happening outside of Γ itself as purely formal symbolism, and only perform negations once enough closed terms (i.e. terms in Γ) have been substituted to bring us back to Γ, where we're definitely allowed to negate. If we adopt this strategy, then $-t(s_2)$ denotes

  $$\begin{aligned} &- \big(t[\mathsf{q}(\mathsf{p}_{A_1})][\mathsf{p}_{A_3}][\mathsf{id} ,_+ s_1 ,_+ s_2 ,_+ s_3]\big) \\ &= -\big(t[\mathsf{id} ,_+ s_2]\big). \end{aligned}$$

Of course, there's no ambiguity. These are the same term:

$$\begin{aligned} &(-t)[\mathsf{q}(\mathsf{p}_{A_1})][\mathsf{p}_{A_3}][\mathsf{id} ,_+ s_1 ,_+ s_2 ,_+ s_3] \\ &= (-t)[\mathsf{q}(\mathsf{p}_{A_1})][\mathsf{id} ,_+ s_1 ,_+ s_2] \\ &= (-t)[\mathsf{p} \circ \mathsf{p} ,_+ \mathsf{v}_0][\mathsf{id} ,_+ s_1 ,_+ s_2] \\ &= (-t)[\mathsf{p} \circ \mathsf{p} \circ (\mathsf{id} ,_+ s_1 ,_+ s_2) ,_+ \mathsf{v}_0[\mathsf{id} ,_+ s_1 ,_+ s_2]] \\ &= (-t)[\mathsf{id} ,_+ s_2] \\ &= -(t[\mathsf{id} ,_+ s_2]) \end{aligned}$$

The last step is an application of Theorem 2.3.30: the substitution

$$(\mathsf{id} ,_+ s_2) \quad : \mathsf{Sub}\ \Gamma\ (\Gamma \rhd^+ A_2)$$

is a substitution between neutral contexts, and negation is stable under such substitutions.

The "eager" approach to negation of open terms requires strict adherence to Var-Neg: as noted above, this becomes nonsense if the types A_2 or T or the term t depend on the non-neutral variables at all. However, the "lazy" approach, if fully developed, seems like it would permit us to transgress Var-Neg. Consider the putative term-in-telescope

$$x_1 : A_1, x_2 : A_2, x_3 : A_3(x_1, x_2) \vdash -x_1 : A_1^-.$$

Var-Neg would forbid this, and under the eager approach it would fail: $\Gamma \rhd^+ A_1$ is not neutral, so we cannot negate the de Bruijn index 0 in that context. But, if the negation were adopted lazily, i.e. only performed once the telescope were fulfilled with closed terms, then there's no issue: $-x_1$, substituted with s_1, s_2, s_3 as above, would just become $-s_1$, a perfectly-valid closed term of type A_1^-.

In the present work, we follow the eager approach and steadfastly uphold Var-Neg. This is because it's unclear how to reconcile the lazy approach with the necessary restrictions on directed path induction: recall that the inability to negate non-neutral variables was the crucial mechanism preventing symmetry from being provable by directed path induction (but permitting it for neutral types). If we want to adopt the lazy method and run roughshod over Var-Neg, then careful work is needed to explain why this newfound liberty wouldn't permit us to prove symmetry.[a] We don't yet know what the lazy approach, properly developed, would look like, but this idea has the potential to alleviate the more difficult restrictions of our polarity calculus.

[a] For instance, perhaps we might require the motive for directed path induction to be eagerly expressible.

4.1.2 Directed Type Theory

With that established, we can do directed type theory. As discussed in the beginning of chapter 3, the Hom-formation rule can be stated in great generality: for instance, we could state for an arbitrary telescope Δ as follows.

$$\frac{\Delta \vdash x : A^- \quad \Delta \vdash y : A}{\Delta \vdash \mathsf{Hom}(x, y) \text{ neut. type}} \qquad \text{(Hom Formation)}$$

But often times we won't need this level of generality; instead, it might suffice to say

$$\frac{}{x : A^-, y : A \vdash \mathsf{Hom}(x, y) \text{ neut. type}}$$

or even just the version for closed terms:

$$\frac{t : A^- \quad t' : A}{\mathsf{Hom}(t, t') \text{ neut. type.}}$$

Note that we assert $\mathsf{Hom}(t, t')$ as a neutral type; this tells us we're working in $(m, 1)$-directed type theory. As before, we'll write Id for the hom-types of neutral types, and assume UIP for hom-types (so we're indeed working in (1,1)-directed type theory). Also, we'll continue to affirm judgmentally that A^- is the *opposite category* of A:

$$\mathsf{Hom}_{A^-}(x', x) = \mathsf{Hom}_A(x, x'). \qquad \text{(Op-Observ.)}$$

This latter statement of Hom-formation is the one which is relevant for the case of hom-*introduction*: since we need t to occur both positively and negatively in the type of refl_t, we can only introduce refl for closed terms:

$$\frac{t : A^-}{\mathsf{refl}_t : \mathsf{Hom}(t, -t).} \qquad \text{(Hom Introduction)}$$

As before, we have two principles of path induction: J^+ allows us to induct *forwards*, i.e. prove a statement generically over *coslices* merely by proving it for refl; and J^- allows us to induct *backwards*, proving a statement generically over *slices* by proving it for refl. These satisfy their respective β laws, Coslice β and Slice β.

Principle (Coslice Path Induction). [Formal]

Parameters

$t : A^-$

Universal Data

$\mathsf{refl}_t : \mathsf{Hom}(t, -t)$

Eliminator

$$\frac{\begin{array}{c} t : A^- \\ x' : A, u : \mathsf{Hom}(t, x') \vdash M(x', u)\ \mathrm{type} \\ m : M(-t, \mathsf{refl}_t) \end{array}}{x' : A, u : \mathsf{Hom}(t, x') \vdash \mathsf{J}^+\ m : M(x', u)}$$

β Law

$$\mathsf{J}^+\ m\ (-t, \mathsf{refl}_t) = m \qquad \text{(Coslice } \beta\text{)}$$

Principle (Slice Path Induction). [Formal]

Parameters

$t' : A$

Universal Data

$\mathsf{refl}_{-t'} : \mathsf{Hom}(-t', t')$

Eliminator

$$\frac{\begin{array}{c} x : A^-, u : \mathsf{Hom}(x, t') \vdash M(x, u)\ \mathrm{type} \\ m : M(-t', \mathsf{refl}_{-t'}) \end{array}}{x : A^-, u : \mathsf{Hom}(x, t') \vdash \mathsf{J}^-\ m : M(x, u)}$$

β Law

$$\mathsf{J}^- \; m \; (-t', \mathsf{refl}_{-t'}) = m \qquad \text{(Slice } \beta\text{)}$$

Our first task will be to construct the synthetic category structure of types by defining composition of homs. As before, this is effortless with the help of directed path induction.

Construction 4.1.2 (Composition). [Formal]
Given $f : \mathsf{Hom}(t, t')$ and $g : \mathsf{Hom}(-t', t'')$, define $f \cdot g : \mathsf{Hom}(t, t'')$ by either:

- Coslice path induction: define $f \cdot \mathsf{refl}_{t'} := f$
- Slice path induction: define $\mathsf{refl}_{-t'} \cdot g := g$

Again, it will be quite convenient to assert that these definitions coincide judgmentally.

Principle (Composition Coincidence). [Formal]
The two definitions of $f \cdot g$ given in Composition are equal.

Thus, both $f \cdot \mathsf{refl} = f$ and $\mathsf{refl} \cdot g = g$ hold as judgmental equalities, and can be proved as propositional identities, $\mathsf{Id}(f \cdot \mathsf{refl}, f)$ and $\mathsf{Id}(\mathsf{refl} \cdot g, g)$ by refl rather than by another directed path induction. Associativity, however, does require path induction.

Construction 4.1.3 (Associativity of Composition). [Formal]
For hom-terms f, g, h such that $f \cdot (g \cdot h)$ is well-typed, the associativity witness

$$\mathsf{assoc}\ f\ g\ h \quad : \mathsf{Id}(f \cdot (g \cdot h),\ (f \cdot g) \cdot h)$$

is defined by coslice path induction on h:

$$\mathsf{assoc}\ f\ g\ \mathsf{refl} := \mathsf{refl}_{f \cdot g} \quad : \mathsf{Id}(f \cdot (g \cdot \mathsf{refl}),\ (f \cdot g) \cdot \mathsf{refl})$$

Let us also repeat the observation that neutral types have symmetric hom-types, i.e. identity types.

Construction 4.1.4 (Symmetry of Identity). [Formal]
Suppose A is a neutral type and $t : A^-$. Then the term-in-telescope

$$x' : A, u : \mathsf{Id}(t, x') \vdash u^{-1} : \mathsf{Id}(-x', -t)$$

is given by $\mathsf{J}^+\mathsf{refl}_t$, i.e.

$$\mathsf{refl}_t^{-1} := \mathsf{refl}_t.$$

As alluded to in Remark 4.1.1, here we observe the most critical function of Var-

Neg: the negation of the free variable x' in the type-in-telescope $\mathsf{Id}(-x', -t)$ is only permitted because A is neutral. This constitutes our explanation for why symmetry is not provable in general, and hence why our directed type theory is truly *directed*: the hom-formation rule and Var-Neg each impose polarity disciplines, and a putative proof of symmetry for arbitrary (not necessarily neutral) types cannot satisfy both. Of course, the constraints of polarity and neutrality are often cumbersome, and we might look for ways to weaken the polarity calculus to permit various constructions. But the unprovability of symmetry serves as our fundamental benchmark: any weakening of this system which permits the above proof to go forward for arbitrary types is too weak.

4.1.3 Type Formers

For the purposes of synthetic category theory, we won't have much use for dependent functions—our focus will instead be on non-dependent functions, which will serve as *synthetic functors*. This is convenient, as our only available utility for working with multivariable function types in polarized type theory—the ee isomorphisms of section 2.3—is only useful for non-dependent functions[5] (recall that the ee isomorphisms were specified for *flat telescopes*). Multivariable *dependent* functions could probably be treated in this informal style, but that would require us to handle negative context extension, which we avoid doing here. Instead, we'll just content ourselves with the following rules for defining non-dependent functions.

$$\frac{A \text{ type} \quad B \text{ type}}{A \to B \text{ type}} \qquad (\to\text{Formation})$$

$$\frac{A_1, \dots, A_n, B \text{ type} \quad F : A_1 \to \cdots \to A_n \to B}{x_1 : A_1, \dots, x_n : A_n \vdash F(x_1, \dots, x_n) : B} \qquad (\to\text{Application})$$

$$\frac{A_1, \dots, A_n, B \text{ type} \quad x_1 : A_1, \dots, x_n : A_n \vdash h(x_1, \dots, x_n) : B}{\lambda x_1 \ \dots \ x_n.h(x_1, \dots, x_n) : A_1 \to \cdots \to A_n \to B} \qquad (\to\text{Abstraction})$$

The latter two rules are mutually inverse—these are the appropriate β and η laws. Both encode the use of the ee isomorphisms to avoid working with negative context extention: the →Application law represents the operation of applying app, n times to get a term of type $B[\mathsf{p} \circ \cdots \circ \mathsf{p}]$ in context $\Gamma \rhd^- A_1[\mathsf{e}] \rhd^- \cdots \rhd^- A_n[\cdots]$, and then substituting by the n-ary ee^{-1} to turn all the $\rhd^-$s into $\rhd^+$s. Likewise, →Abstraction represents substitution by the n-ary ee, followed by n-many applications of lam. The $n = 1$ version of these procedures is encapsulated in the app^+ and lam^+ operators of Definition 2.3.10, while the $n = 2$ version of lambda abstraction is utilized in Proposition 2.3.34 and Definition 3.2.6.

Let us also call to mind Proposition 2.2.18 and its law Proposition 2.3.21:

$$\frac{F : (A \to B)^-}{F^- : A^- \to B^-} \qquad \frac{}{F^{--} = F} \qquad \frac{t : A^- \quad F : A \to B}{-((-F)^- \ t) = F(-t)} \tag{4.1.5}$$

[5] And single-variable dependent functions.

We leave the antecedent as $F : (A \to B)^-$ to match how it was phrased before, but, since we're talking about closed terms, it's no trouble to coerce between $(A \to B)^-$ and $A \to B$, as we do in the latter law. As we said when initially introducing this law, it reflects the fact that any functor $A \to B$ can equivalently be viewed as a functor on the opposite categories $A^- \to B^-$.

As we've seen, Σ-types are simpler to incorporate into polarized type theory, as their inherently positive nature means we can operate solely with positive context extension. This extends into the informal setting: we're able to state their laws in relative generality, and, unlike with Π-types, it is no extra trouble to work with the dependent type constructs.

$$\frac{\Delta \vdash A \text{ type} \quad \Delta, x : A \vdash B(x) \text{ type}}{\sum_{x:A} B(x) \text{ type}} \qquad (\Sigma\text{-Formation})$$

$$\frac{\Delta \vdash a : A \quad \Delta \vdash b : B(a)}{\Delta \vdash (a, b) : \sum_{x:A} B(x)} \qquad (\Sigma\text{-Introduction})$$

$$\frac{\Delta, z : \sum_{x:A} B(x) \vdash M(z) \text{ type} \quad \Delta, x : A, y : B(x) \vdash m : M(x, y)}{\Delta, z : \sum_{x:A} B(x) \vdash \Sigma\text{-elim } m\ z : M(z)} \qquad (\Sigma\text{-Elimination})$$

All these can be stated in an arbitrary telescope Δ, corresponding to the fact that neutral contexts weren't necessary to state the introduction and elimination principles for Σ-types in their convenient form. We only need to assume the terms/types are closed in order to coerce between $\sum_{x:A} B(x)$ and its opposite.

$$\frac{a : A^- \quad b : B(-a)^-}{(a, b) : (\sum_{x:A} B(x))^-} \qquad \frac{}{-(a, b) = (-a, -b)} \qquad (\Sigma^-\text{-Introduction})$$

So we're overloading the $(\_, \_)$ operation to stand for both Pair and Pair$^-$.

Finally, we have the universe of sets,

$$\frac{}{\Delta \vdash \mathsf{Set} \text{ type}} \qquad \frac{\Delta \vdash X : \mathsf{Set}}{\Delta \vdash \mathsf{El}\ X \text{ neut. type}} \qquad \frac{\Delta \vdash s', t' : \mathsf{El}\ X \quad \Delta \vdash p, q : \mathsf{Id}(s', t')}{\Delta \vdash \mathsf{UIP}(p, q) : \mathsf{Id}(p, q)}$$

which contains all hom-types

$$\frac{\Delta \vdash t : A^- \quad \Delta \vdash t' : A}{\Delta \vdash \mathsf{HomSet}(t, t') : \mathsf{Set}} \qquad \frac{\Delta \vdash t : A^- \quad \Delta \vdash t' : A}{\Delta \vdash \mathsf{El}\ (\mathsf{HomSet}(t, t')) = \mathsf{Hom}(t, t').}$$

Using this, we can recreate Definition 3.2.6 in the informal setting:

Construction 4.1.6 (Yoneda Embedding). [Formal]
For any type A, define the **Yoneda embedding** $\mathsf{y} : A \to A^- \to \mathsf{Set}$ by

$$\mathsf{y} := \lambda x'\ x.\mathsf{HomSet}(x, x')$$

We reason more about the properties of Yoneda in section 4.4.

4.1.4 Inductive Category Theory

Finally, a word on methodology. The category theory conducted here differs from the usual category-theoretic practice: there is a difference in *ontology*—our theory is synthetic (categories are our basic objects, and we never need to demonstrate that something *is a category*) while standard category theory is analytic (categories are defined objects with respect to an ambient theory, e.g. set theory); but there is also a difference in *methodology*: for us, category-theoretic notions (e.g. 'product', 'pushout', 'right adjoint') are to be characterized by *principles of induction* rather than the traditional *universal mapping properties*. Capturing the essential features of a new construct with an induction principle is how type-theoretic reasoning is best done, and we'll find that the key constructs of basic category theory are quite amenable to this style.

Throughout this chapter, we'll be introducing several more principles of induction; we'll continue to use the format seen above, which makes explicit the **parameters** of the induction; the **universal data** that's being asserted, i.e. the objects and morphisms whose universal mapping property is being expressed in the principle; the **eliminator** rule asserting the operation of the universal property; and the appropriate computation rule or β **law** governing how the eliminator works. This makes explicit the usual logical structure of universal mapping properties in category theory. For instance, we can articulate the universal mapping property of the product as

> "given objects X, Y (the **parameters**), a *product* of X and Y consists of a span $X \leftarrow P \rightarrow Y$ (the **universal data**) such that for every other span $X \leftarrow Z \rightarrow Y$, obtain a unique morphism $Z \rightarrow P$ (the **elimination** rule) such that the triangles commute (the β **law**)."

Below, we'll have a slightly different framing of the universal mapping property for products as an induction principle, but it will follow this same template.

However, it is incumbent upon us to demonstrate that we really are getting at the same notion, e.g. that our use of the word 'product' coincides with that of standard category theory. The demonstration will consist of two steps:

- **Synthesis**: First, we show that the universal mapping property (stated in synthetic category theory) arises as a consequence of our principle of induction. In other words, the inductive characterization is at least as strong as the universal mapping property characterization. For each notion, we'll do this demonstration informally first, but we'll also do so formally, i.e. by explicitly constructing terms in an arbitrary neutral context of the syntax model of (1,1)-directed type theory.
- **Analysis**: The other step in proving the adequacy of our inductive categorical notion is showing that it's not *too strong*. Placing limits on our theory's power is a job for the semantics: what we will do is show that our inductive notion corresponds to the standard one in the empty context of the category model (we'll use the term "ECCM analysis", since it is based in the **e**mpty **c**ontext of the **c**ategory **m**odel). For instance, we'll demonstrate that our inductive notion of 'products' is not too strong by showing that any category with products (in the standard sense) will have them (in our sense) when that category is regarded as a type in the empty context of the category model.

With these two steps accomplished, we'll have shown that our inductive notion is a faithful rendering of the standard notion, but one better suited to the synthetic category theory setting provided to us by (1,1)-directed type theory.

Without further ado, we can begin our synthetic category theory in earnest.

4.2 Binary (Co)Products

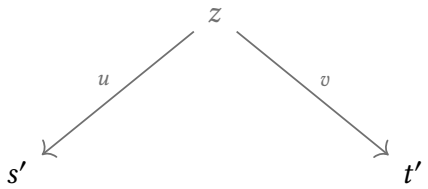
4.2.1 Binary Products—Informal

One of the key notions in elementary category theory are **products**. Binary products (and, more generally, n-ary products) take the idea of *cartesian products of sets* and abstract their essential properties to purely category-theoretic *universal mapping property*. The category-theoretic notion of product encompasses the various notions of 'product' possessed by set-based structures (e.g. the product topology, the direct product of groups) as well as various constructions less commonly considered 'products' (such as the binary meets in partially-ordered sets).

In our synthetic category theory, the universal mapping property (UMP) of products will be a principle of induction on *binary cones*. Fix some arbitrary type A and terms $s', t' : A$, and consider type families of the form

$$z : A^-, u : \mathsf{Hom}(z, s'), v : \mathsf{Hom}(z, t') \vdash M(z, u, v) \text{ type}.$$

The telescope z, u, v is an abstract, binary-cone-shaped diagram over the objects s', t' in A.



In order to inhabit $M(z, u, v)$ abstractly over z, u, v, *by induction*, we need s', t' to come equipped with a *universal cone*. This is our notion of 'product'.

Principle (Binary Cone Induction). [Formal]

Parameters

$s'\ t' : A$

Universal Data

$P : A$
$\pi_1 : \mathsf{Hom}(-P, s')$
$\pi_2 : \mathsf{Hom}(-P, t')$

Eliminator

$$\frac{\begin{array}{c} z : A^-, u : \mathsf{Hom}(z, s'), v : \mathsf{Hom}(z, t') \vdash M(z, u, v) \text{ type} \\ m : M(-P, \pi_1, \pi_2) \end{array}}{z : A^-, u : \mathsf{Hom}(z, s'), v : \mathsf{Hom}(z, t') \vdash \mathsf{elim}\ m\ (z, u, v) : M(z, u, v)}$$

β Law

$$\mathsf{elim}\ m\ (-P, \pi_1, \pi_2) = m \qquad \text{(Binary Cone } \beta\text{)}$$

In words: a *product* of s' and t' is a binary cone (i.e. a span) $-P, \pi_1, \pi_2$ which is logically generic over binary cones, in the sense that any property (M) which can be proved (m) for P, π_1, π_2 can be proved $(\mathsf{elim}\ m\ (z, u, v))$ for an arbitrary cone z, u, v. Note that our choice of polarities for the closed terms s', t', and P are somewhat arbitrary (e.g. we could've specified $P : A^-$ instead) since these are closed terms and can be freely coerced. But we do have to be careful about the open terms—z must be annotated negative, since its role is to be the *domain* of the morphisms consisting of the cone.

From this inductive characterization, we can recover the standard category-theoretic notion of 'product' (this is the "synthesis" step mentioned above). First, we have that there is a 'pairing' operation combining the legs of a binary cone into a single map into the product object.

Proposition 4.2.1. *Suppose P, π_1, π_2 is a product of s' and t'. Then there is a term-in-telescope*

$$z : A^-, u : \mathsf{Hom}(z, s'), v : \mathsf{Hom}(z, t') \vdash \langle u, v\rangle : \mathsf{Hom}(z, P)$$

along with identities

$$z : A^-, u : \mathsf{Hom}(z, s'), v : \mathsf{Hom}(z, t') \vdash \mathsf{tri}_1 : \mathsf{Id}(\langle u, v\rangle \cdot \pi_1, u)$$
$$z : A^-, u : \mathsf{Hom}(z, s'), v : \mathsf{Hom}(z, t') \vdash \mathsf{tri}_2 : \mathsf{Id}(\langle u, v\rangle \cdot \pi_2, v).$$

Proof. [Formal]
Consider

$$\mathsf{refl}_{-P} : \mathsf{Hom}(-P, P).$$

By the Principle of Binary Cone Induction, it suffices to define $\langle \pi_1, \pi_2\rangle$ in order to construct $\langle u, v\rangle$ for abstract z, u, v, and $\langle \pi_1, \pi_2\rangle := \mathsf{refl}_{-P}$ works. That is,

$$\langle u, v\rangle := \mathsf{elim}\ \mathsf{refl}_{-P}\ (z, u, v).$$

The required identities also follow easily: by Binary Cone β, we know

$$\langle \pi_1, \pi_2\rangle = \mathsf{refl}_{-P},$$

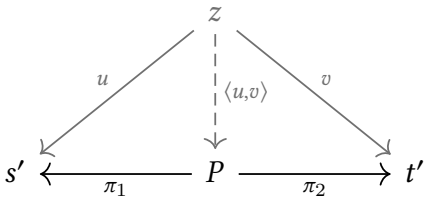
and thus the right unit laws for composition

$$\mathsf{runit}(\pi_1) : \mathsf{Id}(\mathsf{refl}_{-P} \cdot \pi_1, \pi_1) \qquad \mathsf{runit}(\pi_2) : \mathsf{Id}(\mathsf{refl}_{-P} \cdot \pi_2, \pi_2)$$

prove that $\mathsf{Id}(\langle \pi_1, \pi_2\rangle \cdot \pi_1, \pi_1)$ and likewise for π_2. By the Principle of Binary Cone Induction, we have the required identities for abstract z, u, v. □

Incorporating the interpretations of directed equality from section 3.1, we might read the situation like this: a product P of s' and t' is supposed to consist of an amalgamation

of s' and t'; this is witnessed by the directed paths π_1 and π_2, which *extract* or *project* out one component, i.e. are processes by which P can *become* s' and t', respectively. That P consists *only* of a combination of s' and t'—nothing more—is demonstrated by the universality of P, π_1, π_2 among binary cones, in particular the result we just proved, Proposition 4.2.1: for any hypothetical state z with processes u and v by which z could become s' and t', there must be a process $\langle u, v\rangle$ by which z becomes P, and the processes u and v must just consist of $\langle u, v\rangle$ followed by π_1 and π_2, respectively. The usual depiction of this situation is the following commutative diagram:



The two identities proved above are the statement that the two triangles depicted are *commutative triangles*.

The other key component of the UMP of products is that the induced morphism $\langle u, v\rangle$ is the *unique* one making these triangles commute. For simplicity, we only prove this claim for *closed terms*,[6] in the following way.

Proposition 4.2.2. *There is an identity*

$$z : A^-, w : \mathsf{Hom}(z, P) \vdash \eta_0(w) : \mathsf{Id}(w, \langle w \cdot \pi_1, w \cdot \pi_2\rangle).$$

Proof. [Formal]
By the Principle of Slice Path Induction, it suffices to construct an identity

$$\eta_0(\mathsf{refl}_{-P}) : \mathsf{Id}(\mathsf{refl}_{-P}, \langle \mathsf{refl}_{-P} \cdot \pi_1, \mathsf{refl}_{-P} \cdot \pi_2\rangle).$$

By the right unit laws of composition, we know $\mathsf{refl}_{-P} \cdot \pi_1 = \pi_1$ and $\mathsf{refl}_{-P} \cdot \pi_2 = \pi_2$, so all we need is an identity $\mathsf{Id}(\mathsf{refl}_{-P}, \langle \pi_1, \pi_2\rangle)$. By Binary Cone β, $\mathsf{refl}_{\mathsf{refl}_{-P}}$ suffices. □

Corollary 4.2.3. *For any $k : A^-$, morphisms $g_1 : \mathsf{Hom}(k, s')$, $g_2 : \mathsf{Hom}(k, t')$, and $h : \mathsf{Hom}(k, P)$, with identities*

$$\tau_1 : \mathsf{Id}(h \cdot \pi_1, g_1) \qquad \textit{and} \qquad \tau_2 : \mathsf{Id}(h \cdot \pi_2, g_2)$$

we have an identity

$$\eta(h, \tau_1, \tau_2) : \mathsf{Id}(h, \langle g_1, g_2\rangle).$$

[6] The version of this proof for open terms involves performing transports in the telescope $z : A^-, u : \mathsf{Hom}(z, s'), v : \mathsf{Hom}(z, t')$, which we're not *officially* able to do by the letter of the Principle of Coslice Path Induction, but might be justifiable by logic similar to Var-Neg—the ferms we're transporting are all of identity types, which are neutral.

Proof. [Formal]
Transport the identity

$$\eta_0(h) : \mathsf{Id}(h, \langle h \cdot \pi_1, h \cdot \pi_2 \rangle)$$

along τ_1 and τ_2. □

4.2.2 Binary Products—Formal

Now let's make the notion of 'product' (and the accompanying constructions) fully formal by expressing them in the syntax of (1,1)-directed type theory. In what follows, let Γ be an arbitrary neutral context, $A : \mathsf{Ty}\ \Gamma$, and $s', t' : \mathsf{Tm}(\Gamma, A)$.

Definition 4.2.4 (Binary Products—Formal)**.** A **product** of s' and t' consists of:

- a term $P : \mathsf{Tm}(\Gamma, A)$, with
- terms $\pi_1 : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-P, s'))$ and $\pi_2 : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-P, t'))$

satisfying the **principle of binary cone induction**:

$$\frac{\begin{array}{c} M : \mathsf{Ty}(\Gamma \rhd^+ A^- \rhd^+ \mathsf{Hom}(\mathsf{v}_0, s'[\mathsf{p}]) \rhd^+ \mathsf{Hom}(\mathsf{v}_1, t'[\mathsf{p} \circ \mathsf{p}])) \\ m : \mathsf{Tm}(\Gamma, M[\mathsf{id} ,_+ -P ,_+ \pi_1 ,_+ \pi_2]) \end{array}}{\mathsf{elim}_M\ m : \mathsf{Tm}(\Gamma \rhd^+ A^- \rhd^+ \mathsf{Hom}(\mathsf{v}_0, s'[\mathsf{p}]) \rhd^+ \mathsf{Hom}(\mathsf{v}_1, t'[\mathsf{p} \circ \mathsf{p}]),\ \ M)} \quad \text{(Binary Cone Induction)}$$

such that the following β**-law** is satisfied.

$$\mathsf{elim}_M\ m\ [\mathsf{id} ,_+ -P ,_+ \pi_1 ,_+ \pi_2] = m \qquad \text{(Binary Cone } \beta\text{)}$$

For the purposes of formalizing the constructions involving products, it will be helpful to remind ourselves of some of the abbreviations used in the previous chapter.

Definition 4.2.5. For some context Γ and type $A : \mathsf{Ty}\ \Gamma$, write

- $\mathsf{Csl}\{t\}$ for the type
$$\mathsf{Hom}(t[\mathsf{p}], \mathsf{v}_0) : \mathsf{Ty}(\Gamma \rhd^+ A)$$
for any $t : \mathsf{Tm}(\Gamma, A^-)$;
- $\mathsf{Sl}\{t'\}$ for the type
$$\mathsf{Hom}(\mathsf{v}_0, t'[\mathsf{p}]) : \mathsf{Ty}(\Gamma \rhd^+ A^-);$$
for any $t' : \mathsf{Tm}(\Gamma, A)$;
- $f \cdot \mathsf{v}_0$ for the term-in-telescope
$$\mathsf{J}^+_{-\mathsf{t}', \mathsf{Csl}\{t\}}\ f \quad : \mathsf{Tm}(\Gamma \rhd^+ A \rhd^+ \mathsf{Csl}\{-\mathsf{t}'\},\ \mathsf{Hom}(t[\mathsf{p} \circ \mathsf{p}], \mathsf{v}_1))$$
for any $f : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, t'))$;
- $\mathsf{v}_0 \cdot g$ for the term-in-telescope
$$\mathsf{J}^-_{\mathsf{t}', \mathsf{Sl}\{t''\}}\ g \quad : \mathsf{Tm}(\Gamma \rhd^+ A^- \rhd^+ \mathsf{Sl}\{\mathsf{t}'\},\ \mathsf{Hom}(\mathsf{v}_1, t''[\mathsf{p} \circ \mathsf{p}]))$$
for any $g : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-t', t''))$.

Recall that $f \cdot g$ is equal to $(f \cdot \mathsf{v}_0)[\mathsf{id} \mathbin{,_+} t'' \mathbin{,_+} g]$ *and* to $(\mathsf{v}_0 \cdot g)[\mathsf{id} \mathbin{,_+} t \mathbin{,_+} f]$—this was the content of Composition Coincidence Axiom.

With this, we can formalize all the constructions given above; this is done in Figure 4.1. We adopt the notation $\langle \mathsf{v}_1, \mathsf{v}_0 \rangle$ for the paired term *in-telescope*, which becomes $\langle f, g \rangle$ when closed terms (i.e. terms in Γ) f and g of the appropriate type are substituted in. Since an open term can be post-composed by a closed term, we're able to discuss the commutative triangles and prove the uniqueness part of the universal property.

So we've satisfied the 'synthesis' portion of our demonstration: the notion of 'product' defined in Definition 4.2.4 generates the usual universal mapping property of products in the synthetic category theory. Now the other way around: analyzing this concept in the empty context of the category model.

Proposition 4.2.6 (Binary Products—ECCM Analysis)**.** *Let $\mathcal{C}$ be a category and fix $I, J : |\mathcal{C}|$. Then suppose $P : |\mathcal{C}|$, $p_I : \mathcal{C}\ [P, I]$ and $p_J : \mathcal{C}\ [P, J]$ has the universal mapping property of the product of I and J.*[a] *Then, considering $\mathcal{C}$ as a type in the empty context of the category model, I, J, P as terms of $\mathcal{C}$, and p_I, p_J terms of the appropriate hom-types, we have that P, p_I, p_J is a product of I, J in the sense of Definition 4.2.4.*

[a] For every $K : |\mathcal{C}|$ and maps $i : \mathcal{C}\ [K, I]$ and $j : \mathcal{C}\ [K, J]$, there exists a unique map $\langle i, j \rangle : \mathcal{C}\ [K, P]$ such that $p_I \circ \langle i, j \rangle = i$ and $p_J \circ \langle i, j \rangle = j$.

Proof. Let $\mathcal{C}, I, J, P, p_I, p_J$ be as given. Then a motive M for binary cone induction is a functor with object part

$$M_{\#1} : (X : |\mathcal{C}|) \to \mathcal{C}\ [X, I] \to \mathcal{C}\ [X, J] \to \mathrm{Cat}$$

and morphism part

$$M_{\#2} : (x_{10} : \mathcal{C}[X_1, X_0]) \to M_{\#1}(X_0, i_0, j_0) \Rightarrow M_{\#1}(X_1, i_0 \circ x_{10}, j_0 \circ x_{10}).$$

The supplied *method* m is just an object of the category $M(P, p_I, p_2)$.

The desired term $\mathsf{elim}(m)$ should have shape

$$\begin{aligned}
&\mathsf{elim}(m)_{\#1} : (X : |\mathcal{C}|) \to (i : \mathcal{C}\ [X, I]) \to (j : \mathcal{C}\ [X, J]) \to |M(X, i, j)| \\
&\mathsf{elim}(m)_{\#2} : (x_{10} : \mathcal{C}[X_1, X_0]) \to \\
&\qquad M(X_1, i_0 \circ x_{10}, j_0 \circ x_{10})\ [\\
&\qquad\quad M\ x_{10}\ (\mathsf{elim}(m)_{\#1}\ (X_0, i_0, j_0)), \\
&\qquad\quad \mathsf{elim}(m)_{\#1}(X_1, i_0 \circ x_{10}, j_0 \circ x_{10}) \\
&\qquad]
\end{aligned}$$

For the object part, it suffices to put

$$(\mathsf{elim}\ m)\ (X, i, j) := M\ (\langle i, j \rangle)\ m \qquad : M(X, p_I \circ \langle i, j \rangle, p_J \circ \langle i, j \rangle).$$

Note that this has the right type because $\langle i, j \rangle$ makes the triangles commute. For the

morphism part, observe:

$$\begin{aligned}
&M\ x_{10}\ ((\mathsf{elim}\ m)\ (X_0, i_0, j_0))) \\
&= M\ x_{10}\ (M\ \langle i_0, j_0\rangle\ m) \\
&= M\ (\langle i_0, j_0\rangle \circ x_{10})\ m && \text{(Functoriality of } M\text{)} \\
&= M\ \langle i_0 \circ x_{10}, j_0 \circ x_{10}\rangle\ m && \text{(UMP of products)} \\
&= (\mathsf{elim}\ m)(X_1, i_0 \circ x_{10}, j_0 \circ x_{10}).
\end{aligned}$$

And so the natural choice for $\mathsf{elim}\ \ m\ \ x_{10}$ is just the identity morphism. Since $\langle p_I, p_J\rangle = \mathsf{id}_P$ and M is functorial, we get that $(\mathsf{elim}\ m)[\mathsf{id}_\bullet ,_+ P ,_+ p_I ,_+ p_J]$ must be m, satisfying the β law.

□

Remark 4.2.7. Proposition 4.2.6 is a source of examples of products: for any category $\mathcal{C}$ can be regarded as a closed type in directed type theory and if $\mathcal{C}$ has binary products (or at least a product for some specific pair of objects), then those become products in the sense of Definition 4.2.4. In particular, the universe Set has all binary products, because its interpretation does.

At the moment, we're unable to reason *internally* to (1,1)-directed type theory that functor categories $A \to B$ have products when their codomain category B does (in particular, presheaf categories $A^- \to \mathsf{Set}$ always have binary products), as this requires us to be able to reason about natural transformations componentwise. This is one of the first issues we hope to address once a suitable theory of natural transformations in (1,1)-directed type theory is given.

directedTT

```
ConeTel : Con
ConeTel := Γ ▷⁺ A⁻ ▷⁺ Hom(v₀, s'[ p ]) ▷⁺ Hom(v₁, t'[ p ∘ p ])

-- Pairing
⟨v₁,v₀⟩ : Tm(ConeTel, Hom(v₂, P [ p ∘ p ∘ p ]))
⟨v₁,v₀⟩ := elim refl_P

⟨_,_⟩ : Tm(Γ, Hom(k,s')) → Tm(Γ, Hom(k,t')) → Tm(Γ, Hom(k,P))
⟨ g₁ , g₂ ⟩ := ⟨v₁,v₀⟩ [ id ,+ k ,+ g₁ ,+ g₂ ]
β {m = refl_P} : ⟨ π₁ , π₂ ⟩ = refl_P

-- Triangles commute
tri₁ : Tm(ConeTel, Id((v₀· π₁)[ p ∘ p ,+ ⟨v₁,v₀⟩ ],  v₁ )
tri₁ := elim refl_π1

tri₂ : Tm(ConeTel, Id((v₀· π₂)[ p ∘ p ,+ ⟨v₁,v₀⟩ ],  v₀ )
tri₂ := elim refl_π2

-- Pairing is the only morphism making the triangles commute

η₀ : Tm(Γ ▷⁺ A⁻ ▷⁺ Sl{P}, Id(v₀, ⟨v₁,v₀⟩[ p ,+ v₀· π₁ ,+ v₀· π₂ ]))
η₀ := J⁻ refl_refl_P

η : {k : Tm(Γ,A⁻)}{g₁ : Tm(Γ,Hom(k,s'))}{g₂ : Tm(Γ,Hom(k,t'))} →
      (h : Tm(Γ, Hom(k, P))) →
      Tm(Γ, Id( h · π₁ , g₁ )) →
      Tm(Γ, Id( h · π₂ , g₂ )) →
      Tm(Γ, Id( h , ⟨g₁,g₂⟩ ))
η h τ₁ τ₂ := tr τ₁ (tr τ₂ (η₀[ id ,+ k ,+ h ]))
```

Figure 4.1: Universal mapping property of binary products, where $s', t' : \mathrm{Tm}(\Gamma, A)$ and P, π_1, π_2 are assumed to form a product of s' and t' (witnessed by elim).

4.2.3 Binary Coproducts

Now, we can play the standard 'duality' trick of category theory: flip all the arrows around, turn the minuses into pluses and *vice-versa*, use *slice* path induction instead of *coslice*, pre-compose instead of post-compose, etc. We thereby arrive at our notion of **coproduct**.

Principle (Binary Cocone Induction). [Formal]

Parameters

$s\ t : A^-$

Universal Data

$Q : A^-$
$\iota_1 : \mathsf{Hom}(s, -Q)$
$\iota_2 : \mathsf{Hom}(t, -Q)$

Eliminator

$$\frac{\begin{array}{c} z' : A, u : \mathsf{Hom}(s, z'), v : \mathsf{Hom}(t, z') \vdash M(z', u, v)\ \mathrm{type} \\ m : M(-Q, \iota_1, \iota_2) \end{array}}{z' : A, u : \mathsf{Hom}(s, z'), v : \mathsf{Hom}(t, z') \vdash \mathsf{elim}\ m\ (z', u, v) : M(z', u, v)}$$

β Law

$$\mathsf{elim}\ m\ (-Q, \iota_1, \iota_2) = m \qquad \text{(Binary Cocone } \beta\text{)}$$

Remark 4.2.8. Observe that the Principle of Binary Cocone Induction is just the Principle of Binary Cone Induction for the opposite category A^-, rewritten with Op-Observ.. So the claim, "coproducts are just products in the opposite category," (and *vice-versa*) is essentially a tautology in our theory. Of course, the same observation will apply to subsequent dual pairs, like pullbacks and pushouts, left and right adjoints, etc.

We can then perform the same synthetic derivation of the standard universal mapping property, *mutatis mutandis*.

Proposition 4.2.9. *Suppose Q, ι_1, ι_2 is a coproduct of s and t. Then there is a term-in-telescope*

$$z' : A, u : \mathsf{Hom}(s, z'), v : \mathsf{Hom}(t, z') \vdash [\![u, v]\!] : \mathsf{Hom}(Q, z')$$

along with identities

$$z' : A, u : \mathsf{Hom}(s, z'), v : \mathsf{Hom}(t, z') \vdash \mathsf{tri}_1 : \mathsf{Id}(\iota_1 \cdot [\![u, v]\!], u)$$
$$z' : A, u : \mathsf{Hom}(s, z'), v : \mathsf{Hom}(t, z') \vdash \mathsf{tri}_2 : \mathsf{Id}(\iota_2 \cdot [\![u, v]\!], v).$$

Proof. [Formal]
Consider

$$\mathsf{refl}_Q : \mathsf{Hom}(Q, -Q).$$

By the Principle of Binary Cocone Induction, it suffices to define $[\![\iota_1, \iota_2]\!]$ in order to construct $[\![u, v]\!]$ for abstract z', u, v, and $[\![\iota_1, \iota_2]\!] := \mathsf{refl}_Q$ works. That is,

$$[\![u, v]\!] := \mathsf{elim}\ \mathsf{refl}_Q\ (z', u, v).$$

The required identities also follow easily: by Binary Cocone β, we know

$$[\![\iota_1, \iota_2]\!] = \mathsf{refl}_Q,$$

and thus the left unit laws for composition

$$\mathsf{lunit}(\iota_1) : \mathsf{Id}(\iota_1 \cdot \mathsf{refl}_Q, \iota_1) \qquad \mathsf{lunit}(\iota_2) : \mathsf{Id}(\iota_2 \cdot \mathsf{refl}_Q, \iota_2)$$

prove that $\mathsf{Id}(\iota_1 [\![\iota_1, \iota_2]\!] \cdot, \iota_1)$ and likewise for ι_2. By the Principle of Binary Cocone Induction, we have the required identities for abstract z', u, v. □

Proposition 4.2.10. *There is an identity*

$$z' : A, w : \mathsf{Hom}(Q, z') \vdash \eta_0(w) : \mathsf{Id}(w, [\![\iota_1 \cdot w, \iota_2 \cdot w]\!]).$$

Proof. [Formal]
By the Principle of Coslice Path Induction, it suffices to construct an identity

$$\eta_0(\mathsf{refl}_Q) : \mathsf{Id}(\mathsf{refl}_Q, [\![\iota_1 \cdot \mathsf{refl}_Q, \iota_2 \cdot \mathsf{refl}_Q]\!]).$$

By the left unit laws of composition, we know $\iota_1 \cdot \mathsf{refl}_Q = \iota_1$ and $\iota_2 \cdot \mathsf{refl}_Q = \iota_2$, so all we need is an identity $\mathsf{Id}(\mathsf{refl}_Q, [\![\iota_1, \iota_2]\!])$. By Binary Cocone β, $\mathsf{refl}_{\mathsf{refl}_Q}$ suffices. □

Corollary 4.2.11. *For any $k' : A$, morphisms $f_1 : \mathsf{Hom}(s, k')$, $f_2 : \mathsf{Hom}(t, k')$, and $h : \mathsf{Hom}(Q, k')$, with identities*

$$\tau_1 : \mathsf{Id}(\iota_1 \cdot h, f_1) \qquad \textit{and} \qquad \tau_2 : \mathsf{Id}(\iota_2 \cdot h, f_2)$$

we have an identity

$$\eta(h, \tau_1, \tau_2) : \mathsf{Id}(h, [\![f_1, f_2]\!]).$$

Proof. [Formal]
Transport the identity

$$\eta_0(h) : \mathsf{Id}(h, [\![\iota_1 \cdot h, \iota_2 \cdot h]\!])$$

along τ_1 and τ_2. □

And likewise for the corresponding formalization and ECCM analysis.

Definition 4.2.12 (Binary Coproducts—Formal)**.** A **coproduct** of s and t consists of:

- a term $Q : \mathsf{Tm}(\Gamma, A^-)$, with
- terms $\iota_1 : \mathsf{Tm}(\Gamma, \mathsf{Hom}(s, -Q))$ and $\iota_2 : \mathsf{Tm}(\Gamma, \mathsf{Hom}(t, -Q))$

satisfying the **principle of binary cocone induction**:

$$\frac{\begin{array}{c} M : \mathsf{Ty}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(s[\mathsf{p}], \mathsf{v}_0) \rhd^+ \mathsf{Hom}(t[\mathsf{p} \circ \mathsf{p}], \mathsf{v}_1)) \\ m : \mathsf{Tm}(\Gamma, M[\mathsf{id} \mathbin{,_+} -Q \mathbin{,_+} \iota_1 \mathbin{,_+} \iota_2]) \end{array}}{\mathsf{elim}_M\ m : \mathsf{Tm}(\Gamma \rhd^+ A \rhd^+ \mathsf{Hom}(s[\mathsf{p}], \mathsf{v}_0) \rhd^+ \mathsf{Hom}(t[\mathsf{p} \circ \mathsf{p}], \mathsf{v}_1),\ \ M)} \quad \text{(Binary Cocone Induction)}$$

such that the following β**-law** is satisfied.

$$\mathsf{elim}_M\ m\ [\mathsf{id} \mathbin{,_+} -Q \mathbin{,_+} \iota_1 \mathbin{,_+} \iota_2] = m \qquad \text{(Binary Cocone } \beta\text{)}$$

Proposition 4.2.13 (Binary Coproducts—ECCM Analysis)**.** *Let $\mathcal{C}$ be a category and fix $I, J : |\mathcal{C}|$. Then suppose $Q : |\mathcal{C}|$, $q_I : \mathcal{C}\ [I, Q]$ and $q_J : \mathcal{C}\ [J, Q]$ has the universal mapping property of the coproduct of I and J.*[a] *Then, considering $\mathcal{C}$ as a type in the empty context of the category model, I, J, Q as terms of $\mathcal{C}$, and q_I, q_J terms of the appropriate hom-types, we have that Q, q_I, q_J is a coproduct of I, J in the sense of Definition 4.2.12.*

[a] For every $K : |\mathcal{C}|$ and maps $i : \mathcal{C}\ [I, K]$ and $j : \mathcal{C}\ [J, K]$, there exists a unique map $[\![i, j]\!] : \mathcal{C}\ [Q, K]$ such that $[\![i, j]\!] \circ q_I = i$ and $[\![i, j]\!] \circ q_J = j$.

The proof is almost identical to that of Proposition 4.2.6, but with slightly-different variance.

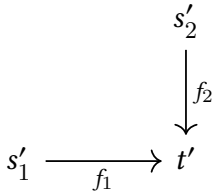
DIRECTEDTT

```
CoconeTel : Con
CoconeTel := Γ ▷⁺ A ▷⁺ Hom(s[ p ], v₀) ▷⁺ Hom(t[ p ∘ p ], v₁)

-- Copairing
[[v₁,v₀]] : Tm(CoconeTel, Hom(Q[ p ∘ p ∘ p ], v₂))
[[v₁,v₀]] := elim refl_Q

[[_,_]] : Tm(Γ, Hom(s,k')) → Tm(Γ, Hom(t,k')) → Tm(Γ, Hom(Q,k'))
[[ f₁ , f₂ ]] := ⟨v₁,v₀⟩ [ id ,+ k' ,+ f₁ ,+ f₂ ]
+β {m = refl_Q} : [[ ι₁ , ι₂ ]] = refl_Q

-- Triangles commute
tri₁ : Tm(CoconeTel, Id((ι₁·v₀)[ p ∘ p ,+ [[v₁,v₀]] ],  v₁ )
tri₁ := elim refl_ι1

tri₂ : Tm(CoconeTel, Id((ι₂·v₀)[ p ∘ p ,+ [[v₁,v₀]] ],  v₀ )
tri₂ := elim refl_ι2

-- Copairing is the only morphism making the triangles commute

η₀ : Tm(Γ ▷⁺ A ▷⁺ Csl{-Q}, Id( v₀ , [[v₁,v₀]][ p ,+ ι₁ ·v₀ ,+ ι₂ ·v₀ ]))
η₀ w := (J⁺ refl_refl_Q)

η : {k' : Tm(Γ, A)}{f₁ : Tm(Γ, Hom(s,k'))}{f₂ : Tm(Γ, Hom(t,k'))} →
      (h : Tm(Γ, Hom(Q, k'))) →
      Tm(Γ, Id( ι₁ · h , f₁ )) →
      Tm(Γ, Id( ι₂ · h , f₂ )) →
      Tm(Γ, Id( h , [[f₁,f₂]] ))
η h τ₁ τ₂ := tr τ₁ (tr τ₂ (η₀[ id ,+ k' ,+ h ]))
```

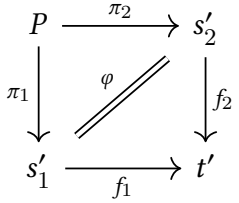
Figure 4.2: Universal mapping property of binary coproducts

4.3 Pullbacks and Pushouts

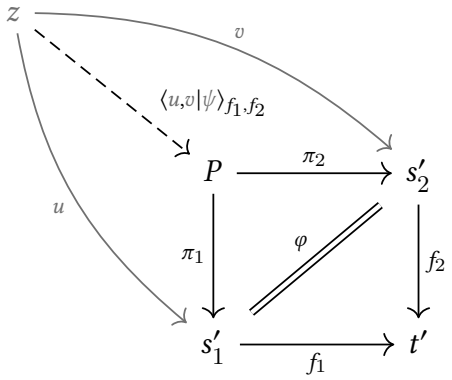
The development of pullbacks and pushouts in our synthetic category theory is, naturally, largely identical to that of products and coproducts. The only additional detail to attend to is the handling of the identities witnessing the commutativity of the square. Consider the pullback of f_1, f_2 as follows.



A *pullback* of this cospan consists not just of an object P and morphisms π_1, π_2, but also a witness φ to the commutativity of the square.



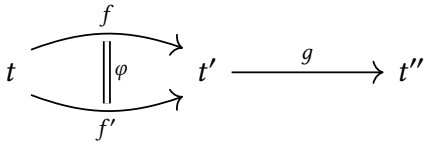
And then the universal mapping property supplies a morphism z to P whenever we supply morphisms $u : \mathsf{Hom}(z, s_1')$ and $v : \mathsf{Hom}(z, s_2')$ *and* a witness ψ that $u \cdot f_1$ equals $v \cdot f_2$.



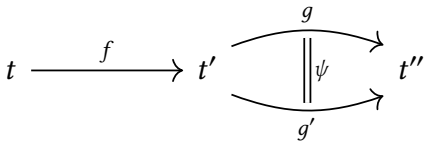
The key property of this induced map $\langle u, v \mid \psi \rangle_{f_1,f_2}$ is, as usual, that the triangles commute:

$$\mathsf{Id}(\langle u, v \mid \psi \rangle_{f_1,f_2} \cdot \pi_1, u) \qquad \text{and} \qquad \mathsf{Id}(\langle u, v \mid \psi \rangle_{f_1,f_2} \cdot \pi_2, v).$$

But, our demonstration that $\langle u, v \mid \psi \rangle_{f_1,f_2}$ is the *unique* morphism having this property will rely on us being able to "whisker" identities by morphisms. That is, given morphisms f, f', g and $\varphi : \mathsf{Id}(f, f')$



we need to be able to obtain $\varphi \cdot g : \mathsf{Id}(f \cdot g, f' \cdot g)$ and, dually, given $\psi : \mathsf{Id}(g, g')$



obtain $f \cdot \psi : \mathsf{Id}(f \cdot g, f \cdot g')$. This is relevant to our universal mapping property synthesis because our approaching to proving the uniqueness of $\langle u, v \mid \psi \rangle$ will follow the same template as the previous proofs: we'll show that for *any* morphism $w : \mathsf{Hom}(z, P)$, we obtain an identity

$$\eta_0 : \mathsf{Id}(w, \langle w \cdot \pi_1, w \cdot \pi_2 \mid w \cdot \varphi \rangle).$$

Note the use of $w \cdot \varphi$. We therefore define whiskering.

Construction 4.3.1 (Whiskering). [Formal]

- Given $f, f' : \mathsf{Hom}(t, t')$ and $\varphi : \mathsf{Id}(f, f')$, define

$$z' : A, u : \mathsf{Hom}(-t', z') \vdash \varphi \cdot u : \mathsf{Id}(f \cdot u, f' \cdot u)$$

  by $\varphi \cdot u := \mathsf{J}^+\ \varphi\ (z', u)$.
- Given $g, g' : \mathsf{Hom}(-t', t'')$ and $\psi : \mathsf{Id}(g, g')$, define

$$z : A, u : \mathsf{Hom}(z, -t) \vdash u \cdot \psi : \mathsf{Id}(u \cdot g, u \cdot g)$$

  by $u \cdot \psi := \mathsf{J}^-\ \psi\ (z, u)$.

With this established, we can proceed with our notion.

Principle (Span Induction). [Formal]

Parameters

$s_1'\ s_2'\ t' : A$
$f_1 : \mathsf{Hom}(-s_1', t')$
$f_2 : \mathsf{Hom}(-s_2', t')$

Universal Data

$P : A$
$\pi_1 : \mathsf{Hom}(-P, s_1')$
$\pi_2 : \mathsf{Hom}(-P, s_2')$
$\varphi : \mathsf{Id}(\pi_1 \cdot f_1, \pi_2 \cdot f_2)$

Eliminator

$$\frac{z : A^-, u : \mathsf{Hom}(z, s_1'), v : \mathsf{Hom}(z, s_2'), \psi : \mathsf{Id}(u \cdot f_1, v \cdot f_2) \vdash M(z, u, v, \psi)\ \mathrm{type} \qquad m : M(-P, \pi_1, \pi_2, \varphi)}{z, u, v, \psi \vdash \mathsf{elim}\ m\ (z, u, v, \psi) : M(z, u, v, \psi)}$$

β Law

$$\mathsf{elim}\ m\ (-P, \pi_1, \pi_2, \varphi) = m \qquad (\mathrm{Span}\ \beta)$$

The construction of the pairing map is almost identical to Proposition 4.2.1: we define

$$z : A^-, u : \mathsf{Hom}(z, s_1'), v : \mathsf{Hom}(z, s_2'), \psi : \mathsf{Id}(u \cdot f_1, v \cdot f_2) \vdash \langle u, v \mid \psi \rangle_{f_1, f_2} : \mathsf{Hom}(z, P)$$

by induction: $\langle \pi_1, \pi_2 \mid \varphi \rangle := \mathsf{refl}_{-P}$. And, similarly, induction allows us to conclude that

$$\mathsf{Id}(\langle u, v \mid \psi \rangle_{f_1, f_2} \cdot \pi_1, u)$$

from the observation that $\mathsf{refl}_{-P} \cdot \pi_1 = \pi_1$, and likewise for π_2. Also essentially the same is the construction of

$$z : A^-, w : \mathsf{Hom}(z, P) \vdash \eta_0(w) : \mathsf{Id}(w, \langle w \cdot \pi_1, w \cdot \pi_2 \mid w \cdot \varphi \rangle_{f_1, f_2})$$

by the Principle of Slice Path Induction: $\eta_0(\mathsf{refl}_{-P}) := \mathsf{refl}_{\mathsf{refl}_{-P}}$ works because $\mathsf{refl}_{-P} \cdot \pi_1$, $\mathsf{refl}_{-P} \cdot \pi_2$, and $\mathsf{refl}_{-P} \cdot \varphi$ are judgmentally equal to π_1, π_2, and φ, respectively (all by Slice β) and $\langle \pi_1, \pi_2 \mid \varphi \rangle_{f_1, f_2} = \mathsf{refl}_{-P}$ by Cospan β. We proceed a bit more carefully when defining η witnessing the uniqueness of this pairing morphism for closed terms.

Proposition 4.3.2. *Fix any* $k : A^-$, $g_1 : \mathsf{Hom}(k, s_1')$, $g_2 : \mathsf{Hom}(k, s_2')$, *and* $\psi : \mathsf{Id}(g_1 \cdot f_1, g_2 \cdot f_2)$. *Then, for any* $h : \mathsf{Hom}(k, P)$, *with identities*

$$\tau_1 : \mathsf{Id}(h \cdot \pi_1, g_1) \qquad \textit{and} \qquad \tau_2 : \mathsf{Id}(h \cdot \pi_2, g_2)$$

we have an identity

$$\eta(h, \tau_1, \tau_2) : \mathsf{Id}(h, \langle g_1, g_2 | \psi \rangle).$$

Proof. [Formal]
Consider the identity

$$\eta_0(h) : \mathsf{Id}(h, \langle h \cdot \pi_1, h \cdot \pi_2 | h \cdot \varphi \rangle).$$

Now, we want to transport this along the identities τ_1 and τ_2. To do so, first consider

$$u_1 : \mathsf{Hom}(k, s_1'), \theta : \mathsf{Id}(h \cdot \pi_1, u_1) \vdash \mathsf{tr}\ \theta\ (h \cdot \varphi) : \mathsf{Id}(u_1 \cdot f_1, (h \cdot \pi_2) \cdot f_2),$$

and recall that $\mathsf{tr}\ \mathsf{refl}\ (h \cdot \varphi) = h \cdot \varphi$. This will serve as the third argument to $\langle \_, \_ \mid \_ \rangle$

as we transport the first argument along τ_1: we have the motive

$$u_1 : \mathsf{Hom}(k, s_1'), \theta : \mathsf{Id}(h \cdot \pi_1, u_1) \vdash \mathsf{Id}(h, \langle u_1, h \cdot \pi_2 \mid \mathsf{tr}\ \theta\ (h \cdot \varphi)\rangle\ \mathrm{type}.$$

and $\eta_0(h) : \mathsf{Id}(h, \langle h \cdot \pi_1, h \cdot \pi_2 \mid \mathsf{tr}\ \mathsf{refl}_{h\cdot\pi_1}\ (h \cdot \varphi)\rangle)$, so

$$\mathsf{tr}\ \tau_1\ \eta_0(h) \quad : \mathsf{Id}(h, \langle g_1, h \cdot \pi_2 \mid \mathsf{tr}\ \tau_1\ (h \cdot \varphi)\rangle).$$

Likewise with τ_2:

$$\mathsf{tr}\ \tau_2\ (\mathsf{tr}\ \tau_1\ \eta_0(h)) \quad : \mathsf{Id}(h, \langle g_1, g_2 \mid \mathsf{tr}\ \tau_2\ (\mathsf{tr}\ \tau_1\ (h \cdot \varphi))\rangle).$$

This is almost right, but we need the third argument to be ψ. But because both ψ and $\mathsf{tr}\ \tau_2\ (\mathsf{tr}\ \tau_1\ (h \cdot \varphi))$ are identities between homs, i.e. terms of type $\mathsf{Id}(g_1 \cdot f_1, g_2 \cdot f_2)$, they must be equal by UIP. So we can transport $\mathsf{tr}\ \tau_2\ (\mathsf{tr}\ \tau_1\ \eta_0(h))$ along

$$\mathsf{UIP}(\mathsf{tr}\ \tau_2\ (\mathsf{tr}\ \tau_1\ (h \cdot \varphi)), \psi) : \mathsf{Id}(\mathsf{tr}\ \tau_2\ (\mathsf{tr}\ \tau_1\ (h \cdot \varphi)), \psi)$$

and be done. □

The handling of pushouts is simply dual to this; we include the principle of *cospan induction* for completeness.

Principle (Cospan Induction). [Formal]

Parameters

$s_1\ s_2\ t : A^-$
$g_1 : \mathsf{Hom}(t, -s_1)$
$g_2 : \mathsf{Hom}(t, -s_2)$

Universal Data

$Q : A^-$
$\iota_1 : \mathsf{Hom}(s_1, -Q)$
$\iota_2 : \mathsf{Hom}(s_2, -Q)$
$\psi : \mathsf{Id}(g_1 \cdot \iota_1, g_2 \cdot \iota_2)$

Eliminator

$$\frac{z' : A, u : \mathsf{Hom}(s_1, z'), v : \mathsf{Hom}(s_2, z'), \psi : \mathsf{Id}(g_1 \cdot u, g_2 \cdot v) \vdash M(z', u, v, \psi)\ \mathrm{type} \qquad m : M(-Q, \iota_1, \iota_2, \psi)}{z', u, v, \psi \vdash \mathsf{elim}\ m\ (z', u, v, \psi) : M(z', u, v, \psi)}$$

β Law

$$\text{elim } m\ (-Q, \iota_1, \iota_2, \psi) = m \qquad \text{(Cospan } \beta\text{)}$$

A formalization of the above in the syntax of (1,1)-directed type theory is given in Figure 4.3, Figure 4.4, and Figure 4.5.

directedTT

$$\begin{aligned}
&_\cdot v_0 : \{f\ f' : \mathsf{Tm}(\Gamma,\mathsf{Hom}(t,t'))\} \longrightarrow \mathsf{Tm}(\Gamma, \mathsf{Id}(f,f')) \\
&\quad \longrightarrow \mathsf{Tm}(\Gamma \triangleright^+ A \triangleright^+ \mathsf{Csl}\{-t'\}, \mathsf{Id}(f \cdot v_0, f' \cdot v_0)) \\
&\varphi \cdot v_0 := J^+_{-t',\mathsf{Id}(f\cdot v_0, f'\cdot v_0)}\ \varphi \\
&\beta : (\varphi \cdot v_0)[\ \mathsf{id} ,_+ t' ,_+ \mathsf{refl}_{t'}\ ] = \varphi \\
\\
&v_0 \cdot _ : \{g\ g' : \mathsf{Tm}(\Gamma,\mathsf{Hom}(-t',t''))\} \longrightarrow \mathsf{Tm}(\Gamma, \mathsf{Id}(g,g')) \\
&\quad \longrightarrow \mathsf{Tm}(\Gamma \triangleright^+ A^- \triangleright^+ \mathsf{Sl}\{t'\}, \mathsf{Id}(v_0 \cdot g, v_0 \cdot g')) \\
&v_0 \cdot \psi := J^-_{t',\mathsf{Id}(v_0\cdot g, v_0\cdot g')}\ \psi \\
&\beta : (v_0 \cdot \psi)[\ \mathsf{id} ,_+ -t' ,_+ \mathsf{refl}_{t'}\ ] = \psi
\end{aligned}$$

Figure 4.3: Whiskering of a hom with an identity

directedTT

```
SpanTel : Con
SpanTel :=
      Γ
      ▷⁺ A⁻                                              -- Apex
      ▷⁺ Hom(v₀, s'₁[ p ]) ▷⁺ Hom(v₁, s'₂[ p ∘ p ])      -- Legs
      ▷⁺ Id ( (v₀· f₁)[ p ] , (v₀· f₂)[ p ∘ p ,₊ v₀ ] )  -- Square commutes

-- Pairing
⟨v₂,v₁|v₀⟩_{f₁,f₂} : Tm(SpanTel, Hom(v₃, P [ p ∘ p ∘ p ∘ p ]))
⟨v₂,v₁|v₀⟩_{f₁,f₂} := elim refl_P

⟨_,_|_⟩_{f₁,f₂} : {k : Tm(Γ,A⁻)}
      → (g₁ : Tm(Γ, Hom(k,s'₁)))
      → (g₂ : Tm(Γ, Hom(k,s'₂)))
      → Tm(Γ, Id( g₁ · f₁ , g₂ · f₂ ))
      → Tm(Γ, Hom(k,P))
⟨ g₁ , g₂ | ψ ⟩_{f₁,f₂} := ⟨v₂,v₁|v₀⟩_{f₁,f₂} [ id ,₊ k ,₊ g₁ ,₊ g₂ ,₊ ψ ]
β {m = refl_P} : ⟨ π₁ , π₂ | φ ⟩_{f₁,f₂} = refl_P

-- Triangles commute
tri₁ : Tm(SpanTel, Id((v₀· π₁)[ p ∘ p ∘ p ,₊ ⟨v₂,v₁|v₀⟩_{f₁,f₂} ],  v₂ )
tri₁ := elim refl_{π₁}

tri₂ : Tm(SpanTel, Id((v₀· π₂)[ p ∘ p ∘ p ,₊ ⟨v₂,v₁|v₀⟩_{f₁,f₂} ],  v₁ )
tri₂ := elim refl_{π₂}

-- Pairing is the only morphism making the triangles commute

η₀ : Tm(Γ ▷⁺ A⁻ ▷⁺ Sl{P}, Id(v₀, ⟨v₂,v₁|v₀⟩[ p ,₊ v₀· π₁ ,₊ v₀· π₂ ,₊ v₀· φ ]))
η₀ := J⁻ refl_{refl_P}

η : {k : Tm(Γ,A⁻)}{g₁ : Tm(Γ,Hom(k,s'₁))}{g₂ : Tm(Γ,Hom(k,s'₂))} →
   {ψ : Tm(Γ, Id(g₁ · f₁, g₂ · f₂))} →
      (h : Tm(Γ, Hom(k, P))) →
      Tm(Γ, Id( h · π₁ , g₁ )) →
      Tm(Γ, Id( h · π₂ , g₂ )) →
      Tm(Γ, Id( h , ⟨g₁,g₂|ψ⟩_{f₁,f₂} ))
η h τ₁ τ₂ := tr eq (tr τ₂ (tr τ₁ (η₀[ id ,₊ k ,₊ h ])))
   where
      eq : Tm(Γ, Id( tr τ₂ (tr τ₁ (h · φ)), ψ)
      eq := UIP(tr τ₂ (tr τ₁ (h · φ)), ψ)
```

Figure 4.4: Universal mapping property of pullbacks

directedTT

```
CospanTel : Con
CospanTel :=
    Γ
    ▷⁺ A                                          -- Apex
    ▷⁺ Hom(s₁[ p ], v₀) ▷⁺ Hom(s₂[ p ∘ p ], v₁)    -- Legs
    ▷⁺ Id ( (g₁ ·v₀)[ p ] , (g₂ ·v₀)[ p ∘ p ,+ v₀ ] )   -- Square commutes

-- Copairing
⟦v₂,v₁|v₀⟧g1,g2 : Tm(CospanTel, Hom(Q[ p ∘ p ∘ p ∘ p ], v₃))
⟦v₂,v₁|v₀⟧g1,g2 := elim reflQ

⟦_,_|_⟧g1,g2 : {k' : Tm(Γ,A)}
    → (f₁ : Tm(Γ, Hom(s₁,k')))
    → (f₂ : Tm(Γ, Hom(s₂,k')))
    → Tm(Γ, Id( g₁ · f₁ , g₂ · f₂ ))
    → Tm(Γ, Hom(Q,k'))
⟦ f₁ , f₂ | φ ⟧g1,g2 := ⟦v₂,v₁|v₀⟧g1,g2 [ id ,+ k' ,+ f₁ ,+ f₂ ,+ φ ]
β {m = reflQ} : ⟦ ι₁ , ι₂ | ψ ⟧g1,g2 = reflQ

-- Triangles commute
tri₁ : Tm(CospanTel, Id((ι₁· v₀)[ p ∘ p ∘ p ,+ ⟦v₂,v₁|v₀⟧g1,g2 ],  v₂ )
tri₁ := elim refl_ι1

tri₂ : Tm(CopanTel, Id((ι₂· v₀)[ p ∘ p ∘ p ,+ ⟦v₂,v₁|v₀⟧g1,g2 ],  v₁ )
tri₂ := elim refl_ι2

-- Copairing is the only morphism making the triangles commute

η₀ : Tm(Γ ▷⁺ A ▷⁺ Csl{Q}, Id(v₀, ⟦v₂,v₁|v₀⟧g1,g2[ p ,+ v₀· π₁ ,+ v₀· π₂ ,+ v₀· ψ
    ]))
η₀ := J⁺ refl_reflQ

η : {k' : Tm(Γ,A)}{f₁ : Tm(Γ,Hom(s₁,k'))}{f₂ : Tm(Γ,Hom(s₂,k'))} →
  {φ : Tm(Γ, Id(g₁ · f₁, g₂ · f₂))} →
    (h : Tm(Γ, Hom(Q,k'))) →
    Tm(Γ, Id( ι₁ · h , f₁ )) →
    Tm(Γ, Id( ι₂ · h , f₂ )) →
    Tm(Γ, Id( h , ⟦f₁,f₂|φ⟧g1,g2))
η h τ₁ τ₂ := tr eq (tr τ₂ (tr τ₁ (η₀[ id ,+ k' ,+ h ])))
  where
    eq : Tm(Γ, Id( tr τ₂ (tr τ₁ (ψ · h)), φ)
    eq := UIP(tr τ₂ (tr τ₁ (ψ · h)), φ)
```

Figure 4.5: Universal mapping property of pushouts

4.4 Functors and Adjoints

> "The concept of an adjoint functor is in fact one of the main things that the reader should take away from the study of this book. It is a strictly category-theoretical notion that has turned out to be a conceptual tool of the first magnitude—on par with the idea of a continuous function.
>
> In fact, just as the idea of a topological space arose in connection with continuous functions, so also the notion of category arose in order to define that of a functor, at least according to one of the inventors. The notion of functor arose—so the story goes on—in order to define natural transformations. One might as well continue that natural transformations serve to define adjoints"
>
> Steve Awodey, *Category Theory* [Awo10, p. 2]

4.4.1 Functors and Natural Transformations—Informal

Functors

Let us recall again the idea we earlier dubbed, "the central dogma of category theory": that every notion of "structure" must come equipped with a corresponding notion of "structure-preserving morphism". This dictum applies equally well to categories themselves, of course: we have the notion of **functor**, a structure-preserving map of categories. We've indicated several times previously that a function term $F : A \to B$ in (1,1)-directed type theory will play the role of a **synthetic functor** between the synthetic categories A and B. We've already seen that such an F transforms the "objects": from $t' : A$, obtain $F(t') : B$. But the "object part" of a functor is, well, only part of the functor: it must also have a "morphism part" transforming the A-morphisms in $\mathsf{Hom}_A(t, t')$ to B-morphisms in $\mathsf{Hom}_B(-(F(-t)), F(t'))$, in a way that respects identities and composition. In undirected type theory, this is the *action on paths* principle, ap, witnessing the singular-valued property of functions: if $\mathsf{Id}(t, t')$, then certainly $\mathsf{Id}(F(t), F(t'))$ for any F. Given this, and the functional programming convention of calling the morphism part of a functor "fmap", or just "map" (e.g. for the List endofunctor), it seems appropriate that we should call the directed principle, "map".

Construction 4.4.1 (Functoriality). [Formal]
Given $F : A \to B$ and $t : A^-$, define

$$z' : A, u : \mathsf{Hom}(t, z') \vdash \mathsf{map}\ F\ u : \mathsf{Hom}(-(F(-t)), F(z'))$$

by $\mathsf{map}\ F\ u := \mathsf{J}^+\ \mathsf{refl}_{-(F(-t))}$. For closed $f : \mathsf{Hom}(t, t')$, we can more compactly write

$$\mathsf{map}\ F\ f := \mathsf{tr}\ f\ \mathsf{refl}_{-(F(-t))} \quad : \mathsf{Hom}(-(F(-t)), F(t')).$$

The Coslice β law tells us that $\mathsf{map}\ F\ \mathsf{refl}_t = \mathsf{refl}_{-(F(-t))}$. Functoriality—that map distributes over composition—is proven by (coslice) path induction as well: fixing closed, $f : \mathsf{Hom}(t, t')$

$$z'' : A, v : \mathsf{Hom}(-t', z'') \vdash \mathsf{J}^+\ \mathsf{refl}_{\mathsf{map}\ F\ f} : \mathsf{Id}(\mathsf{map}\ F\ (f \cdot v), (\mathsf{map}\ F\ f) \cdot (\mathsf{map}\ F\ v)).$$

Our only really concrete example of a functor is the Yoneda embedding. Unfortunately, our theory is a bit too limited to prove internally what we know must be the case: that its morphism parts are given by composition. To see the issue, pick some $t', t'' : A$ and observe

$$x : A^-, v : \mathsf{Hom}(x, t') \vdash \mathsf{map}\ (\mathsf{y}\ t'')\ v : \mathsf{Hom}(-\mathsf{HomSet}(-t', t''), \mathsf{HomSet}(x, t'')). \tag{4.4.2}$$

This is a morphism in Set, i.e. a function:

$$x : A^-, v : \mathsf{Hom}(x, t') \vdash \mathsf{Hom{-}to{-}func}(\mathsf{map}\ (\mathsf{y}\ t'')\ v) : \mathsf{Hom}(-t', t'') \to \mathsf{Hom}(x, t'')). \tag{4.4.3}$$

There's no doubt what this function must be: it must be precomposition by v. Unfortunately, we can't write this down so abstractly: the expression $\lambda h.v \cdot h$ is not well-formed, because we don't know how to compose two variable morphisms (one must be closed, so we can do path induction on the other)—perhaps a *bidirectional* induction principle could remedy this situation. Even if we could write down such a function, we would need a principle of function extensionality to prove the two functions equal by proving them pointwise-equal. None of these issues seem insurmountable, but we leave them for future work.

Natural Transformations

In the present work, the highest universe we consider is Set. We don't undertake a consideration of a *universe of categories*, that is, a type Cat whose objects encode 1-categories (i.e. the El operator produces an arbitrary directed 1-type, not a set) and whose hom-types correspond to arbitrary synthetic functors. Such a development ought to be done—not in (1,1)-directed type theory, but in *(2,1)*-directed type theory. First, there is a concern of paradox: if Cat is a type encoding all categories, but all the types in (1,1)-directed type theory are categories; therefore, Cat must contain a code for *itself*, which is known to be a problematic situation in type theory. Moreover, if we tried to introduce Cat in (1,1)-directed type theory, then Cat could not be appropriately (directed) univalent: given $\mathcal{C}, \mathcal{D} :$ Cat, directed univalence would insist that $\mathsf{Hom}(-\mathcal{C}, \mathcal{D})$ corresponds to the function type $\mathsf{El}(\mathcal{C}) \to \mathsf{El}(\mathcal{D})$. The former must be a set, because all hom-types in (1,1)-directed type theory are sets. But we are missing a lot of information if we insist that functor categories $\mathsf{El}(\mathcal{C}) \to \mathsf{El}(\mathcal{D})$ are all sets. In standard category theory, functor categories have rich categorical structure given by natural transformations; we would be "truncating" all this structure away to fit Cat into (1,1)-directed type theory. So, in the present work, there is no type Cat whose objects encode all 1-categories, and we let functor categories $A \to B$ be genuine 1-categories (with natural transformations as their hom-sets—see below), not the hom-sets of Cat.

Let's consider natural transformations. There are two perspectives one can take on natural transformations. First is the kind of "birds-eye" view we've been advancing: a natural transformation is a morphism of functors (this is how we'll be able to talk about natural transformations in our theory). However, natural transformations are not just some abstract arrow between functors: they have to consist of some "stuff". In standard category theory, saying "α is a natural transformation $F \to G$" (for functors $F, G : A \Rightarrow B$) means that α consists of a family of B-morphisms $\mathsf{Hom}_B(F(a), G(a))$,

indexed by objects $a : |A|$, satisfying the appropriate naturality property. Though natural transformations are often constructed abstractly, i.e. without having to explicitly define them out component-wise, sometimes this is necessary—both perspectives on natural transformations are important.

In standard, analytic category, the low-level component-wise aspect comes first: natural transformations are *defined* as natural families of morphisms, and only later can we define 'functor categories', thereby seeing natural transformations as *morphisms between functors*. This is the "analytic" approach to natural transformations. We're doing the opposite, a "synthetic" approach: for us, we just needed to introduce function types into directed type theory, and then automatically (by path induction) we get that (a) the terms of type $A \to B$ are functors with respect to the synthetic category structure of A and B (Functoriality); and (b) the type $A \to B$ is itself a synthetic category, i.e. has hom-types: given functors $F : (A \to B)^-$ and $G : A \to B$, a term $\alpha : \mathsf{Hom}(F, G)$ is a morphism in this functor category. Semantically, such an α will indeed turn out to a natural transformation from F to G (indexed by the context); but in order to really *deserve* the name "natural transformation", there really should be a way to break α into its components. That is, we should have a rule of the form

$$\frac{\begin{array}{cc} F : (A \to B)^- & G : A \to B \\ \alpha : \mathsf{Hom}(F, G) & t' : A \end{array}}{\alpha \mathbin{@} t' \ : \mathsf{Hom}_B(-((-F)\ t'), G(t'))} \tag{4.4.4}$$

along with a witness of naturality:

$$\frac{\begin{array}{ccc} F : (A \to B)^- & G : A \to B & \alpha : \mathsf{Hom}(F, G) \\ t : A^- & t' : A & f : \mathsf{Hom}(t, t') \end{array}}{\mathsf{Nat}(\alpha; f) \ : \mathsf{Id}((\alpha \mathbin{@} (-t)) \cdot (\mathsf{map}\ G\ f),\ (\mathsf{map}\ (-F)\ f) \cdot (\alpha \mathbin{@} t'))}$$

We can *almost* do this with the present tools. The $\_ \mathbin{@} \_$ operator is possible, by path induction on α. Consider the motive

$$Z : A \to B \vdash \mathsf{Hom}(-((-F)\ t'), Z(t'))\ \mathrm{type}. \tag{4.4.5}$$

Now, this is, strictly speaking, not permitted by →Application: that rule only tells us how to apply closed terms. However, by working through the gritty details (done in subsection 4.4.2 below), we can find that this is doable in (1,1)-directed type theory. So then, by the Principle of Coslice Path Induction,

$$Z : A \to B, \alpha : \mathsf{Hom}(F, Z) \vdash \mathsf{J}^+\ \mathsf{refl}_{-((-F)\ t')}\ (Z, \alpha) : \mathsf{Hom}(-((-F)\ t'), Z(t')).$$

So we write $\alpha \mathbin{@} t' := \mathsf{tr}\ \alpha\ \mathsf{refl}_{-((-F)\ t')}$ for closed terms.

Notice, though, how crucial it is that t' is a *closed* term: if t' were a variable of type A, then Var-Neg would forbid $-((-F)\ t')$; if it were instead a variable of type A^-, then we'd need to negate it in order to apply Z to it to get a term of type B—also forbidden. Fundamentally, t' is appears both positively and negatively in $\mathsf{Hom}(F(t'), G(t'))$, so the only way to incorporate this into our polarity calculus[7] is by positing t' in a neutral context.

[7] In its current form.

This turns out to be the issue preventing an internal proof of naturality. Given $f : \mathsf{Hom}(t, t')$, and $\alpha : \mathsf{Hom}(F, G)$, we want to prove the commutativity of the square

$$\begin{array}{ccc} (-F)\ -\ t & \xrightarrow{\alpha\,@\,(-t)} & G(-t) \\ {\scriptstyle \mathsf{map}\ (-F)\ f}\downarrow & & \downarrow{\scriptstyle \mathsf{map}\ G\ f} \\ (-F)\ t' & \xrightarrow[\alpha\,@\,t']{} & G(t'). \end{array}$$

But what do we induct on, α or f? Suppose we say f; then G and α are closed and fixed, but the bottom of the square needs to be open:

$$\begin{array}{ccc} (-F)\ -\ t & \xrightarrow{\alpha\,@\,(-t)} & G(-t) \\ {\scriptstyle \mathsf{map}\ (-F)\ u}\downarrow & & \downarrow{\scriptstyle \mathsf{map}\ G\ u} \\ (-F)\ z' & \xrightarrow[\alpha\,@\,z']{} & G(z'). \end{array}$$

But, alas, $\alpha\,@\,z'$ doesn't make sense: we can't perform the path induction needed to define $@$ if there's an open variable $z' : A$. Likewise if we induct on α: if t' and f are now closed terms, but α has a free endpoint Z,

$$\begin{array}{ccc} (-F)\ -\ t & \xrightarrow{\alpha\,@\,(-t)} & Z(-t) \\ {\scriptstyle \mathsf{map}\ (-F)\ f}\downarrow & & \downarrow{\scriptstyle \mathsf{map}\ Z\ f} \\ (-F)\ t' & \xrightarrow[\alpha\,@\,t']{} & Z(t') \end{array}$$

then map $Z\ f$ is not well-defined. So we're stuck. It appears we don't have the tools to prove this internally with directed path induction. In Proposition 4.4.8 below, we show that naturality for closed terms is admissible in (1,1)-directed type theory, indeed, validated by the category model. For the time being, that'll have to do.

Finally, let us observe that we also lack the ability to *write* new natural transformations in terms of their components. It doesn't make sense in our current polarity calculus to have a λ-rule producing natural transformations, e.g. of the form

$$\frac{x : A^- \vdash h(x) : \mathsf{Hom}(F^-(x), G(-x))}{\lambda x.h(x) \quad : \mathsf{Hom}(F, G)} \tag{4.4.6}$$

because we violate Var-Neg by negating x. This is the flipside of the observation above that t' had to be closed in order for $\alpha\,@\,t'$ to be definable: our polarity calculus presently does not have a mechanism for divariant variables.

So, in conclusion: the synthetic category theory practiced here has a lot of potential, but our flavor of (1,1)-directed type theory needs further development in order to be able to competently handle standard category theoretic constructions.

4.4.2 Functors and Natural Transformations—Formal

Functors

The construction of the morphism part of synthetic functors (Functoriality) is formalized in Figure 4.6. Let us briefly note that map $G\ f$ is interpreted in the category model, as we might expect, as the morphism part of functors:

$$\begin{aligned}
(\mathsf{map}\ G\ f)\ \gamma &:= ((\mathsf{J}^+\ \mathsf{refl}_{-(G\ \$^+\ (-t))})[\mathsf{id} \mathbin{,_+} t' \mathbin{,_+} f])\ \gamma \\
&= ((\mathsf{J}^+\ \mathsf{refl}_{-(G\ \$^+\ (-t))})\ (\gamma, t'(\gamma), f(\gamma)) \\
&= (\mathsf{Hom}((-(G\ \$^+\ (-t)))[\mathsf{p}\circ\mathsf{p}], (\mathsf{app}^+G)[\mathsf{p}]))\ (\mathsf{id}_\gamma, f(\gamma))\ (\mathsf{id}_{G\ \gamma\ (t\ \gamma)}) \\
&\qquad\qquad \text{(Figure 3.2)} \\
&= (\mathsf{app}^+G)(\mathsf{id}_\gamma, f(\gamma)) \circ B\ \mathsf{id}_\gamma\ \mathsf{id}_{G\ \gamma\ (t\ \gamma)} \circ (-(G\ \$^+\ (-t)))\ \mathsf{id}_\gamma\ \mathsf{id}_{G\ \gamma\ (t\ \gamma)} \\
&\qquad\qquad \text{(Definition 3.1.1)} \\
&= (\mathsf{app}^+G)(\mathsf{id}_\gamma, f(\gamma)) \\
&= (\mathsf{app}\ G)[\mathsf{ee}^{-1}]\ (\mathsf{id}_\gamma, f(\gamma)) \\
&= (\mathsf{app}\ G)(\mathsf{id}_\gamma^{-1}, f(\gamma)) \\
&= (G\ \mathsf{id}_\gamma(t'\ \gamma)) \circ B\ \mathsf{id}_\gamma\ (G\ \gamma\ (f\ \gamma)) \qquad \text{(Figure 2.7)} \\
&= G\ \gamma\ (f(\gamma))
\end{aligned}$$

that is, the functor $G(\gamma) : A(\gamma) \Rightarrow B(\gamma)$ applied to the $A(\gamma)$-morphism $f(\gamma)$ from $t(\gamma)$ to $t'(\gamma)$.

As for the above reasoning about the Yoneda embedding's morphism part: we started by instantiating the $\mathsf{map}(\_, \mathsf{v}_0)$ of Figure 4.6 with $\mathsf{y}\ \$^+\ t''$; Equation 4.4.2 rendered formally says

$$\begin{aligned}
&\mathsf{map}(\mathsf{y}\ \$^+\ t'', \mathsf{v}_0) : \mathsf{Tm}(\Gamma \rhd^+ A^- \rhd^+ \mathsf{Hom}(\mathsf{v}_0, t'[\mathsf{p}]), \\
&\quad \mathsf{Hom}(-\mathsf{HomSet}(-t'[\mathsf{p}\circ\mathsf{p}], t''[\mathsf{p}\circ\mathsf{p}]), \mathsf{HomSet}(\mathsf{v}_1, t''[\mathsf{p}\circ\mathsf{p}]))).
\end{aligned}$$

This matches the form of $\mathsf{map}(\_, \mathsf{v}_0)$ because $\mathsf{y}\ \$^+\ t''$ is contravariant: note that we've written $\mathsf{Hom}_A(\mathsf{v}_0, t'[\mathsf{p}])$ in the context instead of $\mathsf{Hom}_{A^-}(t'[\mathsf{p}], \mathsf{v}_0)$.

Subsequently, we apply Hom–to–func to this hom-term (Equation 4.4.3). Formally, this is a substitution of the in-telescope version of Hom–to–func

$$\begin{aligned}
&\mathsf{Hom\text{–}to\text{–}func}(\mathsf{v}_0) : \mathsf{Tm}(\Gamma \rhd^+ \mathsf{Set} \rhd^+ \mathsf{Hom}(-\mathsf{HomSet}(-t'[\mathsf{p}], t''[\mathsf{p}]), \mathsf{v}_0), \\
&\quad \mathsf{HomSet}(-t'[\mathsf{p}], t''[\mathsf{p}])[\mathsf{e}] \to \mathsf{El}\ \mathsf{v}_1)
\end{aligned}$$

which we can substitute with

$$\begin{array}{ccc}
\Gamma \rhd^+ A^- \rhd^+ \mathsf{Hom}(\mathsf{v}_0, t'[\mathsf{p}]) & & \\
\Big\downarrow{\scriptstyle \mathsf{p}\circ\mathsf{p} \mathbin{,_+} \mathsf{HomSet}(\mathsf{v}_1, t''[\mathsf{p}\circ\mathsf{p}]) \mathbin{,_+} \mathsf{map}(\mathsf{y}\ \$^+\ t'', \mathsf{v}_0)} & & \\
\Gamma \rhd^+ \mathsf{Set} \rhd^+ \mathsf{Hom}(-\mathsf{HomSet}(-t'[\mathsf{p}], t''[\mathsf{p}]), \mathsf{v}_0). & & \\
\Big\uparrow{\scriptstyle \mathsf{Hom\text{–}to\text{–}func}(\mathsf{map}(\mathsf{y}\ \$^+\ t'', \mathsf{v}_0))} & & \\
\mathsf{Hom\text{–}to\text{–}func}(\mathsf{v}_0) & &
\end{array}$$

directedTT

```
map(_,v₀) : {t}(F : Tm(Γ, A[e] ⟶ B)) ⟶
  Tm(Γ ▷⁺ A ▷⁺ Hom(t[p], v₀), Hom((–(F $⁺ (–t)))[ p ∘ p ], (app⁺ F)[ p ])
map(F,v₀) := J⁺ refl_{–(F $⁺ (–t))}

map : {t}{t'}(F : Tm(Γ, A[e] ⟶ B)) ⟶ Tm(Γ, Hom(t,t')) ⟶
  Tm(Γ, Hom(–(F $⁺ (–t)), F $⁺ t'))
map F f := (map(F,v₀))[ id ,+ t' ,+ f ]   --    = tr f refl

funct : (F : Tm(Γ, A[e] ⟶ B)) ⟶
  (f : Tm(Γ, Hom(t,t'))) ⟶ (g : Tm(Γ, Hom(–t',t''))) ⟶
  Tm(Γ, Id(map F (f · g), (map F f) · (map F g)))
funct F f g := (J⁺ refl_{map F f})[ id ,+ t'' ,+ g ]
```

Figure 4.6: Morphism part of functions

Natural Transformations

First, let's substantiate the claim (made in passing) that a term $\alpha : \mathsf{Tm}(\Gamma, \mathsf{Hom}(F, G))$ for $F : \mathsf{Tm}(\Gamma, (A[\mathsf{e}] \to B)^-)$ and $G : \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to B)$ is semantically interpreted (in the category model) as a natural transformation between F and G.

Recall Figure 2.9:[8] the object parts of F and G send $\gamma : |\Gamma|$ to functors $A[\mathsf{e}](\gamma) \Rightarrow B(\gamma)$ (and also remember that e does nothing to objects, so $A[\mathsf{e}](\gamma) = A(\gamma)$). But they have opposite morphism parts: for $\gamma_{01} : \Gamma\ [\gamma_0, \gamma_1]$ and $a_1 : |A\ \gamma_1|$,

$$\begin{aligned} F\ \gamma_{01}\ a_1 &: (B\ \gamma_1)\ [F\ \gamma_1\ a_1,\ B\ \gamma_{01}\ (F\ \gamma_0\ (A\ \gamma_{01}^{-1}\ a_1))] \\ G\ \gamma_{01}\ a_1 &: (B\ \gamma_1)\ [B\ \gamma_{01}\ (G\ \gamma_0\ (A\ \gamma_{01}^{-1}\ a_1)),\ G\ \gamma_1\ a_1]. \end{aligned}$$

Then, applying Definition 3.1.1, we get that the object part of α is a family of natural transformations between the object parts of F and G: for each $\gamma : |\Gamma|$, $\alpha(\gamma)$ is a natural transformation from $F(\gamma)$ to $G(\gamma)$.

$$\alpha\ \gamma : (a : |A\ \gamma|) \to (B\gamma)[F\ \gamma\ a,\ G\ \gamma\ a]$$

$$\begin{array}{ccc} F\ \gamma\ a_0 & \xrightarrow{\alpha\ \gamma\ a_0} & G\ \gamma\ a_0 \\ \Big\downarrow{\scriptstyle F\ \gamma\ a_{01}} & & \Big\downarrow{\scriptstyle G\ \gamma\ a_{01}} \\ F\ \gamma\ a_1 & \xrightarrow[\alpha\ \gamma\ a_1]{} & G\ \gamma\ a_1 \end{array}$$

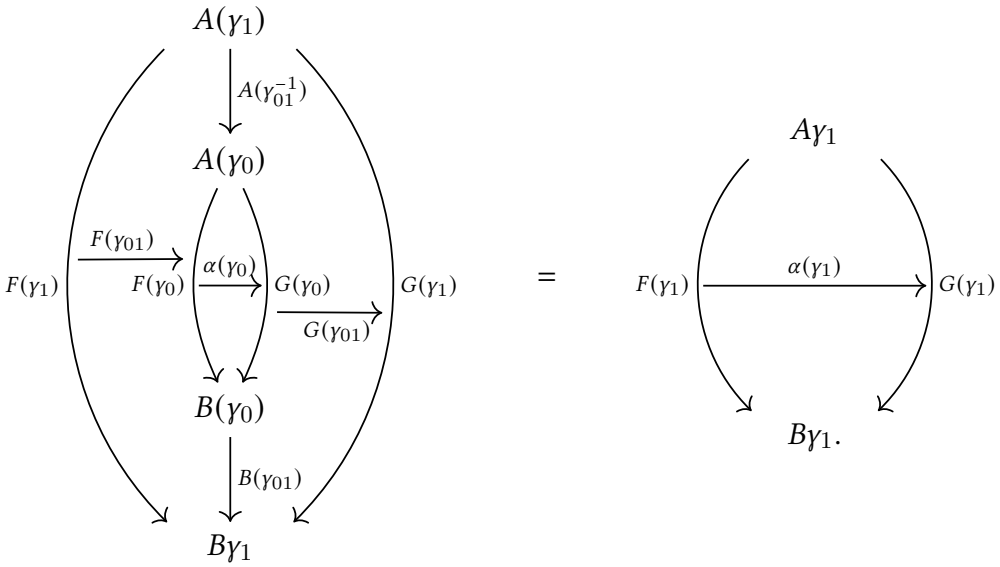
As with any hom-term in the category model, the "morphism part", $\alpha(\gamma_{01})$ is an equation[9] between $\alpha(\gamma_1)$ and $\alpha(\gamma_0)$ transposed with F and G. That is, for every $a_1 : |A\ \gamma_1|$,

[8] NB: In the figures from chapter 2, we typically took $A : \mathsf{Ty}\ \Gamma^-$, so, since our A is in $\mathsf{Ty}\ \Gamma$, we'll throughout we'll be using $A[\mathsf{e}]$ instead of A.

[9] Of natural transformations $F(\gamma_1) \to G(\gamma_1)$

$$(G\ \gamma_{01}\ a_1)\ \circ\ (B\ \gamma_{01}(\alpha\ \gamma_0\ (A\ \gamma_{01}^{-1}\ a_1)))\ \circ\ (F\ \gamma_{01}\ a_1)\quad =\quad \alpha\ \gamma_1\ a_1. \tag{4.4.7}$$

Diagrammatically:



So α is not just a single natural transformation, but a family of natural transformations, functorial in Γ.

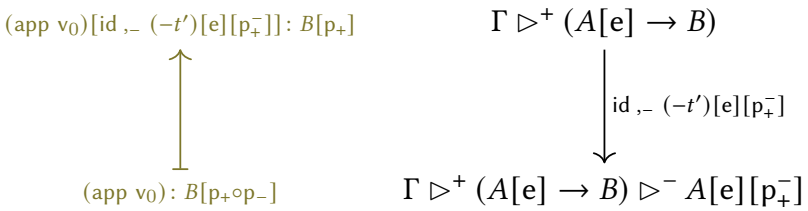
The transport motive depicted in Equation 4.4.5 is indeed possible in (1,1)-directed type theory, but requires a fair amount of details involving negative context extension. The domain of the desired hom-type, denoted informally as $-((-F)\ t')$, is easy: we have $t' : \mathsf{Tm}(\Gamma, A)$ and $-F : \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to B)$, so obtain $-((-F)\ \$^+\ t') : \mathsf{Tm}(\Gamma, B^-)$ and

$$(-((-F)\ \$^+\ t'))[\mathsf{p}] \quad : \mathsf{Tm}(\Gamma \rhd^+ (A[\mathsf{e}] \to B), B[\mathsf{p}]^-).$$

But the codomain—applying the zeroth de Bruijn index to t'—is more difficult. Since $\Gamma \rhd^+ (A[\mathsf{e}] \to B)$ is not neutral, we cannot use our convenient $\$^+$, and must deal with the original app operator, defined in terms of negative context extension. We have the following:

$$\frac{\mathsf{v}_0 : \mathsf{Tm}(\Gamma \rhd^+ (A[\mathsf{e}] \to B), (A[\mathsf{e}] \to B)[\mathsf{p}_+])}{\mathsf{app}\ \mathsf{v}_0 : \mathsf{Tm}(\Gamma \rhd^+ (A[\mathsf{e}] \to B) \rhd^- A[\mathsf{e}][\mathsf{p}_+^-],\ B[\mathsf{p}_+ \circ \mathsf{p}_-])}$$

Now we need to (negatively) substitute in t' as the term of type $A[\mathsf{e}][\mathsf{p}_+^-]$. The appropriate form is $t'[\mathsf{e}][\mathsf{p}_+^-]$:



$$
\begin{aligned}
&(\mathsf{app}\ \mathsf{v}_0)[\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}][\mathsf{p}^-_+]][\mathsf{id} \mathbin{,_+} H] \\
&= (\mathsf{app}\ \mathsf{v}_0)[\mathsf{id} \circ (\mathsf{id} \mathbin{,_+} H) \mathbin{,_-} (-t')[\mathsf{e}][\mathsf{p}^-_+][(\mathsf{id} \mathbin{,_+} H)^-]] && \text{(Naturality of } \_ \mathbin{,_-} \_\text{)} \\
&= (\mathsf{app}\ \mathsf{v}_0)[\mathsf{id} \mathbin{,_+} H \mathbin{,_-} (-t')[\mathsf{e}][\mathsf{p}^-_+][\mathsf{id}^- \mathbin{,_-} H]] \\
&= (\mathsf{app}\ \mathsf{v}_0)[\mathsf{id} \mathbin{,_+} H \mathbin{,_-} (-t')[\mathsf{e}]] \\
&= (\mathsf{app}\ \mathsf{v}_0)[(\mathsf{id} \mathbin{,_+} H) \circ \mathsf{p}^-_+ \circ (\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}]) \quad \mathbin{,_-} \mathsf{v}_0[\mathsf{id} \mathbin{,_+} (-t')[\mathsf{e}]]] \\
&= (\mathsf{app}\ \mathsf{v}_0)[(\mathsf{id} \mathbin{,_+} H) \circ \mathsf{p}^-_+ \circ (\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}]) \quad \mathbin{,_-} \mathsf{v}_0[(\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}])^-]] \\
&= (\mathsf{app}\ \mathsf{v}_0)[(\mathsf{id} \mathbin{,_+} H) \circ \mathsf{p}^-_+ \quad \mathbin{,_-} \mathsf{v}_0][\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}]] \\
&= ((\mathsf{app}\ \mathsf{v}_0)[\mathsf{q}((\mathsf{id} \mathbin{,_+} H)^-)^-])[\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}]] \\
&= (\mathsf{app}(\mathsf{v}_0[\mathsf{id} \mathbin{,_+} H]))[\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}]] && (\mathsf{app}[]) \\
&= (\mathsf{app}\ H)[\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}]] \\
&= (\mathsf{app}\ H)[\mathsf{e}\mathsf{e}^{-1}][\mathsf{id} \mathbin{,_+} t'] && \text{(Corollary 2.3.28)} \\
&= H \mathbin{\$^+} t'
\end{aligned}
$$

Figure 4.7: Calculation of the codomain term in $M[\mathsf{id} \mathbin{,_+} H]$ for some arbitrary term $H : \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to B)$. The $H = -F$ case tells us that $\mathsf{refl}_{-((-F)\mathbin{\$^+}t')}$ has the appropriate codomain for the transport in Figure 4.8, and the $H = G$ case tells us that the transport produces the type we want.

So the informal $Z(t')$ in Equation 4.4.5 is formally expressed as

$$(\mathsf{app}\ \mathsf{v}_0)[\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}][\mathsf{p}^-_+]] \quad : \mathsf{Tm}(\Gamma \rhd^+ (A[\mathsf{e}] \to B), B[\mathsf{p}_+]).$$

Now, write M for $\mathsf{Hom}((-((-F)\mathbin{\$^+}t'))[\mathsf{p}], (\mathsf{app}\ \mathsf{v}_0)[\mathsf{id} \mathbin{,_-} (-t')[\mathsf{e}][\mathsf{p}^-_+]])$. We want to transport in M along some $\alpha : \mathsf{Tm}(\Gamma, \mathsf{Hom}(F, G))$. So we need to supply something of type $M[\mathsf{id} \mathbin{,_+} -F]$. Through an ugly calculation (Figure 4.7 with $H = -F$), we can verify that

$$M[\mathsf{id} \mathbin{,_+} -F] \quad = \quad \mathsf{Hom}(-((-F)\mathbin{\$^+}t'), (-F)\mathbin{\$^+}t'),$$

and so $\mathsf{refl}_{-((-F)\mathbin{\$^+}t')}$ works. So, to conclude, we obtain $\mathsf{tr}\ \mathsf{refl}_{-((-F)\mathbin{\$^+}t')}\ \alpha$ of type $M[\mathsf{id} \mathbin{,_+} G]$, so again by Figure 4.7 (with $H = G$),

$$\mathsf{tr}\ \mathsf{refl}_{-((-F)\mathbin{\$^+}t')}\ \alpha \quad : \mathsf{Tm}(\Gamma, \mathsf{Hom}(-((-F)\mathbin{\$^+}t'), G\mathbin{\$^+}t'))$$

as desired. This is the term we christen $\alpha \mathbin{@} t'$ in Figure 4.8.

As discussed, there does not appear to be a way to prove the naturality of natural transformations internally. But it is true, at least in the category model.

Proposition 4.4.8. *In the syntax of (1,1)-directed type theory, a pair of rules*

$$\frac{\begin{array}{cc}\Gamma : \mathsf{NeutCon} & A, B : \mathsf{Ty}\ \Gamma \\ F : \mathsf{Tm}(\Gamma, (A[\mathsf{e}] \to B)^-) & G : \mathsf{Tm}(\Gamma, A[\mathsf{e}] \to B) \\ \alpha : \mathsf{Tm}(\Gamma, \mathsf{Hom}(F, G)) & t' : \mathsf{Tm}(\Gamma, A)\end{array}}{\alpha \mathbin{@} t' \ : \mathsf{Tm}(\Gamma, \mathsf{Hom}_B(-((-F)\mathbin{\$^+}t'), G\mathbin{\$^+}t'))}$$

directedTT

```
v₀@_ : {Γ}{A}{B}{F : Tm(Γ,(A[e] ⟶ B)⁻)} ⟶ (t' : Tm(Γ, A)) ⟶
  Tm(Γ ▷⁺ (A[e] ⟶ B) ▷⁺ Hom(F,v₀),
    Hom(–((–F) $⁺ t') , (app v₀)[ id ,₊ (–t')[e][p₊⁻] ] ))
v₀@ t' := J⁺ refl_{–((–F)$⁺t')}   -- See Figure 4.7

_@_ : {Γ}{A}{B}{F : Tm(Γ,(A[e] ⟶ B)⁻)}{G : Tm(Γ,A[e] ⟶ B)} ⟶
  Tm(Γ, Hom(F,G)) ⟶ (t' : Tm(Γ, A)) ⟶
  Tm(Γ, Hom(–((–F) $⁺ t') , G $⁺ t' ))
α @ t' :=  (v₀@ t')[ id ,₊ G ,₊ α ]   --     = tr refl α
```

Figure 4.8: Components of a natural transformation

$$\frac{\begin{array}{ccccc} F\colon \mathsf{Tm}(\Gamma,(A[\mathrm{e}] \to B)^-) & G\colon \mathsf{Tm}(\Gamma, A[\mathrm{e}] \to B) & \alpha\colon \mathsf{Tm}(\Gamma,\mathsf{Hom}(F,G)) \\ t\colon \mathsf{Tm}(\Gamma, A^-) & t'\colon \mathsf{Tm}(\Gamma, A) & f\colon \mathsf{Tm}(\Gamma,\mathsf{Hom}(t,t')) \end{array}}{\mathsf{Nat}(\alpha; f)\ :\ \mathsf{Tm}(\Gamma, \mathsf{Id}((\alpha\mathbin{@}(-t))\cdot(\mathsf{map}\ G\ f),\ (\mathsf{map}\ (-F)\ f)\cdot(\alpha\mathbin{@}t')))}$$

is admissible (indeed, validated by the category model).

Proof. If we inspect the terms

$$F^-\ \$^+\ (-t')\colon \mathsf{Tm}(\Gamma, B^-) \qquad \text{and} \qquad G\ \$^+\ t'\colon \mathsf{Tm}(\Gamma, B),$$

we find that their object parts send γ to $F\ \gamma\ (t'\ \gamma)$ and to $G\ \gamma\ (t'\ \gamma)$, respectively. Their morphism parts are defined as follows.

$$\begin{aligned} (F^-\ \$^+\ (-t'))\ \gamma_{01} &: (B\ \gamma_1)\ [F\ \gamma_1\ (t'\ \gamma_1),\ B\ \gamma_{01}\ (F\ \gamma_0\ (t'\ \gamma_0))] \\ (F^-\ \$^+\ (-t'))\ \gamma_{01} &:= B\ \gamma_{01}\ (F\ \gamma_0\ (t'\ \gamma_{01}^{-1}))\circ F\ \gamma_{01}\ (t'\ \gamma_1) \\ (G\ \$^+\ t')\ \gamma_{01} &: (B\ \gamma_1)\ [B\ \gamma_{01}\ (G\ \gamma_0\ (t'\ \gamma_0)),\ G\ \gamma_1\ (t'\ \gamma_1)] \\ (G\ \$^+\ t')\ \gamma_{01} &:= G\ \gamma_{01}\ (t'\ \gamma_1)\circ B\ \gamma_{01}\ (G\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_{01}))). \end{aligned}$$

With all that ready, let's define $\alpha\mathbin{@}t'$. The object part of a Hom_B-term sends γ to a $B(\gamma)$-morphism between the object parts of the domain and codomain term:

$$(\alpha\mathbin{@}t')\ \gamma \quad : (B\ \gamma)\ [F\ \gamma\ (t'\ \gamma), G\ \gamma\ (t'\ \gamma)].$$

This is easy: just put $(\alpha\mathbin{@}t')\ \gamma := \alpha\ \gamma\ (t'\ \gamma)$. The less-obvious part, as usual, is the morphism part: for each γ_{01}, we need to prove that the following square (in $B(\gamma_1)$) commutes.

$$\begin{array}{ccc} B\ \gamma_{01}\ (F\ \gamma_0\ (t'\ \gamma_0)) & \xrightarrow{B\ \gamma_{01}\ ((\alpha\mathbin{@}t')\ \gamma_0)} & B\ \gamma_{01}\ (G\ \gamma_0\ (t'\ \gamma_0)) \\ \uparrow{\scriptstyle F\ \gamma_{01}\ (t'\ \gamma_1)} & & \downarrow{\scriptstyle G\ \gamma_{01}\ (t'\ \gamma_1)} \\ F\ \gamma_1\ (t'\ \gamma_1) & \xrightarrow[(\alpha\mathbin{@}t')\ \gamma_1]{} & G\ \gamma_1\ (t'\ \gamma_1) \end{array}$$

That is: we need to show that $\alpha\ \gamma_1\ (t'\ \gamma_1)$ is equal to

$$\begin{aligned}&(G\ \gamma_{01}\ (t'\ \gamma_1))\\ &\circ B\ \gamma_{01}\ (G\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_{01})))\\ &\circ B\ \gamma_{01}\ (\alpha\ \gamma_0\ (t'\ \gamma_0))\\ &\circ B\ \gamma_{01}\ (F\ \gamma_0\ (t'\ \gamma_{01}^{-1}))\\ &\circ (F\ \gamma_{01}\ (t'\ \gamma_1)).\end{aligned}$$

We'll transform the composite of the three expressions appearing next to $B\ \gamma_{01}$. To start, observe that the following maps are mutual inverses, by functoriality of t'.

$$t'\ \gamma_0 \underset{t\ \gamma_{01}^{-1}}{\overset{A\ \gamma_{01}\ (t'\ \gamma_{01})}{\rightleftarrows}} A\ \gamma_{01}^{-1}\ (t'\gamma_1)$$

So, then, if we instantiate the naturality square for α with $\gamma = \gamma_0$, $a_0 = t'(\gamma_0)$, $a_1 = A\ \gamma_{01}^{-1}\ (t'\gamma_1)$ and $a_{01} = A\ \gamma_{01}^{-1}\ (t'\ \gamma_{01})$, we have:

$$\begin{array}{ccc} F\ \gamma_0\ (t'\gamma_0) & \xrightarrow{\alpha\ \gamma_0\ (t'\ \gamma_0)} & G\ \gamma_0\ (t'\ \gamma_0) \\ {\scriptstyle F\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\gamma_{01}))}\downarrow\ \uparrow{\scriptstyle F\ \gamma_0\ (t'\gamma_{01}^{-1})} & & \downarrow{\scriptstyle G\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_{01}))} \\ F\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_1)) & \xrightarrow[\alpha\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_1))]{} & G\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_1)) \end{array}$$

Applying $B(\gamma_{01})$, we get:

$$\begin{aligned}B\ \gamma_{01}\ (\alpha\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_1))) = &B\ \gamma_{01}\ (G\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_{01})))\\ &\circ B\ \gamma_{01}\ (\alpha\ \gamma_0\ (t'\ \gamma_0))\\ &\circ B\ \gamma_{01}\ (F\ \gamma_0\ (t'\ \gamma_{01}^{-1}))\end{aligned}$$

so our goal simplifies to

$$\begin{aligned}\alpha\ \gamma_1\ (t'\ \gamma_1) = &(G\ \gamma_{01}\ (t'\ \gamma_1))\\ &\circ B\ \gamma_{01}\ (\alpha\ \gamma_0\ (A\ \gamma_{01}^{-1}\ (t'\ \gamma_1)))\\ &\circ (F\ \gamma_{01}\ (t'\ \gamma_1)).\end{aligned}$$

This is just the morphism part of α (Equation 4.4.7) with $a_1 = t'(\gamma_1)$, so we're done with defining α.

For the naturality witness $\mathsf{Nat}(\alpha, f)$, this amounts to a witness for each $\gamma : |\Gamma|$ of the commutativity of the square (in $B(\gamma)$)

$$\begin{array}{ccc} F\ \gamma(t\ \gamma) & \xrightarrow{\alpha\ \gamma\ (-t\ \gamma)} & G\ \gamma\ (t\ \gamma) \\ {\scriptstyle F\ \gamma\ (f\ \gamma)}\downarrow & & \downarrow{\scriptstyle G\ \gamma\ (f\ \gamma)} \\ F\ \gamma(t'\ \gamma) & \xrightarrow[\alpha\ \gamma\ (t'\ \gamma)]{} & G\ \gamma\ (t'\ \gamma) \end{array}$$

which is just the naturality of the natural transformation $\alpha(\gamma) : F(\gamma) \to G(\gamma)$. □

4.4.3 Adjoints—Informal

Finally, we show how the notion of *adjunction*—one of the most important and most useful ideas in category theory—admits representation in our calculus. For the purposes of defining (left or right) adjoint as a principle of induction, the most suitable formulation of "adjoint" will be the universal property of the (co)unit ([Mac78, Chapter IV.I, Theorem 2], items (i) and (iii)), rather than the *hom-set isomorphism* formulation often used. Here is our definition.

Principle (Right Adjoint Induction).

Parameters

$F : A \to B$

Universal Data

$U : B \to A$
$\epsilon : \mathsf{Hom}(-(F \circ U), I_B)$

Eliminator

$$\frac{\begin{array}{c} s' : B \\ z : A^-, v : \mathsf{Hom}((-F)^- z, s') \vdash M(z, v)\ \mathrm{type} \\ m : M(-U(s'), \epsilon \mathbin{@} s') \end{array}}{z : A^-, v : \mathsf{Hom}((-F)^- z, s') \vdash \mathsf{elim}\ m\ (z, v) : M(z, v)}$$

β Law

$$\mathsf{elim}\ m\ (-U(s'), \epsilon \mathbin{@} s') = m \qquad \text{(Right Adjoint } \beta\text{)}$$

Principle (Left Adjoint Induction).

Parameters

$U : B \to A$

Universal Data

$F : A \to B$
$\eta : \mathsf{Hom}(-I_A, U \circ F)$

Eliminator

$$\frac{\begin{array}{c} t : A^- \\ z' : B, v : \mathsf{Hom}(t, U(z')) \vdash M(z', v)\ \mathrm{type} \\ m : M(F(-t), \eta \mathbin{@} (-t)) \end{array}}{z' : B, v : \mathsf{Hom}(t, U(z')) \vdash \mathsf{elim}\ m\ (z', v) : M(z', v)}$$

β Law

$$\mathsf{elim}\ m\ (F(-t), \eta \mathbin{@} (-t)) = m \qquad \text{(Left Adjoint } \beta\text{)}$$

Let's first verify that our use of the natural transformation is well-typed. Consider the motive m required for the eliminator of a right adjoint. In order for $m : M(-U(s'), \epsilon \mathbin{@} s')$ to make sense, we need

$$\epsilon \mathbin{@} s' : \mathsf{Hom}((-F)^-(-U(s')), s').$$

To see this is the case, we use the rule (Equation 4.1.5) that $(-G)^-\ t = -(G(-t))$. Per Equation 4.4.4, the domain of $\epsilon \mathbin{@} s'$ is

$$-((-(-(F \circ U))\ s') = -((F \circ U)\ s') = -(F(U(s'))) = (-F)^-(-U(s')),$$

as desired; the codomain is just $I_B(s') = s'$.

Now, as before, we can obtain the usual universal mapping property from our principle of induction. We'll focus on the right adjoint, but of course the analogous constructions can be done using the left adjoint induction principle. Suppose $F : A \to B$ has a right adjoint U, ϵ. Then, for any $s' : B$, obtain a term-in-telescope

$$z : A^-, v : \mathsf{Hom}_B((-F)^- z, s') \vdash \widetilde{v} : \mathsf{Hom}_A(z, U(s'))$$

by the Principle of Right Adjoint Induction:

$$\widetilde{\epsilon \mathbin{@} s'} := \mathsf{refl}_{-U(s')} : \mathsf{Hom}_A(-U(s'), U(s')).$$

And *vice versa* by the Principle of Slice Path Induction at $U(s')$: since we can define

$$\overline{\mathsf{refl}_{-U(s')}} := \epsilon \mathbin{@} s' : \mathsf{Hom}_B((-F)^-(-U(s')), s'),$$

obtain

$$z : A^-, u : \mathsf{Hom}(z, -U(s')) \vdash \overline{\mathsf{refl}_{-U(s')}} : \mathsf{Hom}_B((-F)^-(z), s').$$

It's just as easy to prove these operations mutually inverse (pointwise, up to propositional equality of morphisms), and from that we have the standard universal property.

As is well-known, there are countless uses for adjunctions in category theory. For instance, adjunctions provide a framework for discussing the existence of *(co)limits*.

Definition 4.4.9. Given some type J, a type A is said to have **all limits** (resp. **all colimits**) of shape J if the constant functor

$$K := \lambda x\ j.x \quad : A \to (J \to A)$$

has a right (resp. left) adjoint $(J \to A) \to A$.

Example 4.4.10. Say that a type A has **all binary products** if it has all limits of shape $\mathbb{2}$, i.e. there is some

$$\mathsf{prod} : (\mathbb{2} \to A) \to A \qquad \text{and} \qquad \epsilon : \mathsf{Hom}(-(K \circ \mathsf{prod}), I_{\mathbb{2}\to A}).$$

with the universal mapping property of right adjoints, Right Adjoint Induction Principle. Then, for any $s', t' : A$, we can obtain their product by

$$P := \mathsf{prod}(\lambda b.\mathsf{if}\ b\ \mathsf{then}\ s'\ \mathsf{else}\ t') \quad : A.$$

If we take the components of ϵ, we get a natural transformation

$$\pi := \epsilon\ @\ (\lambda b.\mathsf{if}\ b\ \mathsf{then}\ s'\ \mathsf{else}\ t') \quad : \mathsf{Hom}_{\mathbb{2}\to A}(-K(P), \lambda b.\mathsf{if}\ b\ \mathsf{then}\ s'\ \mathsf{else}\ t').$$

The components of *this* transformation provide us the projection morphisms:

$$\begin{aligned} \pi_1 &:= \pi\ @\ \mathsf{tt} \quad : \mathsf{Hom}(-P, s') \\ \pi_2 &:= \pi\ @\ \mathsf{ff} \quad : \mathsf{Hom}(-P, t'). \end{aligned}$$

We can *almost* prove that this satisfies the the Principle of Binary Cone Induction: if M is a type-in-telescope

$$z : A^-, u : \mathsf{Hom}(z, s'), v : \mathsf{Hom}(z, t') \vdash M(z, u, v)\ \mathrm{type}$$

for which we have a term $m : M(-P, \pi_1, \pi_2)$, then observe we can construct the type-in-telescope

$$z : A^-, w : \mathsf{Hom}((-K)^- z, \lambda b.\mathsf{if}\ b\ \mathsf{then}\ s'\ \mathsf{else}\ t') \vdash M'(z, w)\ \mathrm{type}$$

by

$$M'(z, w) := M(z, w\ @\ \mathsf{tt}, w\ @\ \mathsf{ff})$$

and indeed $m : M'(-P, \epsilon\ @\ (\lambda b.\mathsf{if}\ b\ \mathsf{then}\ s'\ \mathsf{else}\ t'))$, so by the the Principle of Right Adjoint Induction, obtain

$$z : A^-, w : \mathsf{Hom}((-K)^- z, \lambda b.\mathsf{if}\ b\ \mathsf{then}\ s'\ \mathsf{else}\ t') \vdash \mathsf{elim}\ m\ (z, w) : M'(z, w).$$

Now, if we wanted to obtain from this an element of $M(z, u, v)$ for our abstract binary cone (z, u, v), we would need to package u, v into an abstract natural transformation $\mathsf{Hom}((-K)^- z, \lambda b.\mathsf{if}\ b\ \mathsf{then}\ s'\ \mathsf{else}\ t')$, which we said we don't have the tools to do. But this at least demonstrates that we can get very close, serving as a proof-of-concept for future work.

Chapter 5

Conclusion

5.1 Research Questions Reprised

To review, let us briefly revisit the questions posed in section 0.2.

Question (Research Question 1). What is an appropriate semantics-driven methodology for developing directed type theory and synthetic category theory?

An essential characteristic of the present work is that our investigation into the **semantics of directed type theory** is *prior to* (or at least *simultaneous with*) the development of its syntax. Unlike many traditional (and even contemporary) formal theories, we do not pronounce the syntax of our formal language and leave the semantics to be developed years or decades later, or never at all. Instead, we start with the semantics—the **category model of directed type theory**—and deduce what the semantics of directed type theory *ought to be*, based on what's possible in the model.

We had an immanently practical reason for proceeding this way. The basic goal of directed type theory is to make a theory which is just like *un*directed type theory in every way possible, *except* that symmetry cannot be proved. In simplicial type theory, this is achieved by adding a singularly *directed* type—the directed interval—to undirected type theory, and defining hom-types in terms of the directed interval. But we wanted to hew closer to Martin-Löf Type Theory, and posit our hom-types judgmentally, that is, as generated inductively by refl and eliminated by a J-rule with judgmental β-law. To do this, we had to surgically weaken the J-rule of MLTT to make it strong enough to still prove composition (and associativity of composition, transport, map, Hom–to–func, and so on), but still too weak to prove symmetry. How could we be confident we had achieved this? The claim, "the J-rule *cannot* prove symmetry" is, of course, not the kind of statement which can be proved in the theory itself: it's a **metatheoretic** claim, the kind that demands a semantics. Any attempt to do what we've done here—define a directed type theory with judgmentally-asserted hom-types—needs to proceed from

semantics, to safeguard the *directed* character of the theory.

The basic structural mechanics of type theory (contexts, substitutions, types, terms, variables, etc.) can be extended in an endless variety of different ways—type theorists have studied far too many different systems of type- and term-formers to effectively survey. In light of such diversity, it becomes desirable to have a modular framework for reasoning about these different theories and their semantic models. In the theory of **generalized algebra** expounded in chapter 1, we have such a framework: the GAT $\mathfrak{CwF}$ of **categories with families** makes the fundamental structural mechanics of type theory fully mathematically explicit, and the semantics of specific type theories can be stated as extensions of this GAT. Critically, every such extension automatically has an **initial algebra**, meaning that we're always justified in speaking of the *syntax* of a type theory given in this way (without having to resolve a difficult "initiality conjecture"). This ideally facilitates semantics-driven development of a language: given an intended model, we can focus our attention on abstractly articulating its salient features in a GAT, and have the syntax (and the possibility of metatheoretic arguments using the interpretation of the syntax in our intended model) available automatically.

This is the approach we have taken to developing a directed type theory. The modularity provided by generalized algebra made it easy to define an array of 'model notions' for directed type theory: polarized CwFs (Definition 2.1.16); zero-ary, unary, and general neutral-polarized CwFs (Definition 2.3.7,Definition 2.3.25,Definition 2.3.44); directed CwFs (Definition 3.1.6); (1,1)-DCwFs, (1,0)-DCwFs, (0,1)-DCwFs, and (0,0)-DCwFs (Definition 3.1.13); and these models augmented with various type-formers and axioms—to capture more and more detail of the category model. We used the fact that the category model is a (1,1)-DCwF with specific non-neutral types (e.g. $\vec{2}$, Set) and therefore admits an interpretation morphism from the *initial* such model, the syntax of (1,1)-directed type theory with these types, to prove symmetry is independent of the theory. And we used the category model's situation relative to the syntax model of (1,1)-directed type theory to perform "ECCM analysis", that is, to compare (and validate) our **synthetic category-theoretic** notions against the *analytic* category theory of the category model's empty context.

Question (Research Question 2). How can the groupoid model's uses of *symmetry* be made explicit in the syntax of type theory?

Our primary task throughout chapter 2 and into chapter 3 was to contend with the fact that Hofmann and Streicher's groupoid interpretation of type theory is indeed a *groupoid* interpretation of type theory—they sometimes make use of the fact that the structures in question are *groupoids* specifically and not arbitrary categories, i.e. they allow themselves to invert morphisms. To adapt these definitions into the category model, we enriched the syntax of type theory with **polarity** annotations making explicit "which way the morphisms had to be pointed" to get the semantics to work out. Recall we had several different varieties of these "polarity problems":

- We had **shallow** polarity problems, where Hofmann and Streicher use that a type A in the groupoid model is given by a family of groupoids, i.e. the categories $A(\gamma)$ have invertible morphisms, but *not* that the context Γ itself is a groupoid. These were addressed by introducing the type A^- (interpreted as the fiberwise

opposite of A), and annotating appropriately, e.g. in hom-introduction.
- We had **deep** polarity problems, where a construction relies on the fact that the *contexts* are groupoids. To remedy such problems, we extended our polarity calculus to operate on contexts, substitutions, and context extension as well. This allowed us, for instance, to capture that the domain type A in $\Pi(A, B)$ appears negatively, i.e. depends contravariantly on the *context*.
- We encountered a number of places where either the deep polarity calculus was too cumbersome to be workable, or where a term needed to appear both positively and negatively. These issues we addressed by reintroducing groupoids to the theory, in the form of **neutral** contexts and types. When working in a neutral context, we had a number of utilities for overcoming the constraints of the polarity calculus, but, critically, not so much that symmetry became provable.

As we'll discuss more shortly, there are polarity problems which are not solved by any of these techniques. In particular, we do not have a mechanism for introducing a variable that can appear both negatively and positively (i.e. perform some kind of *divariant* context extension). An extension of the present theory capable of such a thing would need to have a more sophisticated polarity calculus; the present work serves as a template for how we might go about designing such a calculus.

Question (Research Question 3). What is *directed equality*, and how does it work?

In chapter 3, we advanced several readings of what the presence, absence, and (non-)uniqueness of a terms of type $\mathsf{Hom}(t, t')$ *represents*. Consistent with our ultimate aim—synthetic 1-category theory—and our semantics in the category model, we can understand the types of directed type theory as categories, the terms as objects, and the hom-terms as *morphisms*. In this reading, the type $\mathsf{Hom}(t, t')$ having multiple, distinct inhabitants simply represents the existence of distinct, parallel morphisms—witnessing that the category A is not a *preorder*. This perspective naturally adapts to (m, n)-directed type theory where m and n are not both 1. But we had other ways of reading this data:
- we could think of the types as **directed homotopy spaces**, the terms as points, and hom-terms as *directed paths*;
- we could think of a type as some kind of **rewrite system**, the terms as *expressions*, and the hom-terms as *reductions*;
- we could think of a type as a **state space**, the terms as *states*, and the hom-terms as *processes*.

For each idea and construct(ion) in directed type theory, we can try to apply each of these readings; some things are very natural under every reading (e.g. composition), whereas some notions are easier to connect to one interpretation than others. But what we have done here is establish some basic interpretation for what a **directed notion of equality** means, and how it works.

As part of this exploration, we developed a few directed **extensionality** principles governing the hom-types of structured types. In particular, we characterized the hom-types of Σ-types—a hom between dependent pairs is a dependent pair of homs—and of the universe Set of sets, whose homs are given by functions on the corresponding types. This provides two key ingredients towards a **structure morphism principle**, which would stipulate that the hom-types of structured sets are given by their classical

notion of homomorphism. Work on this point continues.

Question (Research Question 4). What does synthetic category-theoretic reasoning in the directed type theory of the category model look like?

The present work culminates in a brief development synthetic category theory in (1,1)-directed type theory. This development is done using an **informal style** of type theory, rather than the formal syntax—making it more digestible for human readers (including, potentially, someone who is totally unfamiliar with the underlying formal calculus). Rather than simply translating category-theoretic concepts (like 'right adjoint') into the language of type theory (a somewhat dubious procedure, given the tight polarity constraints on our Π-types), we instead modify the practice of category theory to more naturally fit within the directed-type-theoretic framework. This is done by recasting the *universal mapping properties* of category theory as *principles of induction*. Though the theory needs expansion to accommodate more elaborate category-theoretic reasoning (in particular, to work with natural transformations componentwise), the amount developed here serves as a proof-of-concept.

5.2 Some Suggestions for Future Work

Every aspect of the present work could benefit from further development. Though we cannot hope to be exhaustive, here are some (we think) worthwhile avenues to explore.

There's much more to be said about the ONEGAT language. Unary parametricity for ONEGAT amounts to the fact that every displayed algebra over the initial algebra admits a section (for arbitrary GAT), but the corresponding statement of *binary*, i.e *relational* parametricity has not, to our knowledge, been given. Moreover, further study is needed into the following phenomenon: whenever a GAT $\mathfrak{G}$ is extended to another GAT $\mathfrak{H}$, we obtain the forgetful functor $\mathfrak{H}$-Alg $\Rightarrow$ $\mathfrak{G}$-Alg; under what circumstances does this functor have a left or right adjoint? Most of the cases we dealt with seemed to admit appropriate adjoints—free and core groupoids of a category, free PCwF on a CwF (Proposition 2.2.5), etc.—but this claim needs to be worked out in precise detail.

We also alluded several times to *second order generalized algebraic theories*. It has been recently shown [KX24] that SOGATs admit a nice signature language in the same vein ONEGAT, and that SOGAT signatures can be systematically translated to GAT signatures. However, not all GAT signatures arise in this way; in particular, there hasn't (to our knowledge) been a systematic study of capturing *substructural* features—like our deep polarity calculus—in the SOGAT setting. Further work is needed to be able to extend SOGATs, so that notions like *neutral-context operations* are expressible.

With regards to the directed type theory expounded here, there is much more to explore. One possible feature to investigate are adding identity types atop the hom-types of this directed type theory. We indicated two apparent ways to do so: one based on left adjoints (identity types are the symmetric closure of hom-types, so $\mathsf{Hom}(t, t')$ implies $\mathsf{Id}(t, t')$) or on right adjoints (identity types are the core of hom-types, so $\mathsf{Id}(t, t')$ implies $\mathsf{Hom}(t, t')$ and $\mathsf{Hom}(-t', -t)$). And of course, one could explore a system with both kinds of identity: consider, for example, a *synthetic rewriting system* where $\mathsf{Hom}(t, t')$ encodes that t reduces to t'. In this case, the left identity would be

some kind of *extensional equivalence*: t and t' are identical if they reduce to the same thing; the right identity would be some kind of *literal equality*: t and t' are only identical if they reduce to each other. Evaluating systems like this in light of the interpretations advanced in chapter 3 sounds like a fruitful endeavor.

We also mentioned several times that the present system needs to be extended if we want to be able to *write* natural transformations component-wise. As we remarked above, any putative "natural transformation lambda rule" like Equation 4.4.6 violates Var-Neg by making the abstracted variable appear both positively and negatively. So perhaps we need some kind of di-variant context extension, allowing for such variable abstractions. Or maybe some relaxation of Var-Neg (that still upholds the independence of symmetry) is possible. Numerous developments in our synthetic category theory are pending the resolution of this question, such as the proper treatment of presheaf categories (including a statement of the Yoneda Lemma).

Finally, as we suggested with the name "(1,1)-directed type theory," there are numerous possible variations on our syntax and semantics. The most obvious candidates to explore are the adjacent directed type theories: (0,1)-directed type theory (with the preorder model as its paradigm model) and (2,1)-directed type theory (which would require the appropriate definition of the 2-category model). Presumably, the latter would allow for a study of the category of categories within the synthetic setting. We look forward to such developments.

Bibliography

[ABK+21] Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, Christian Sattler, and Filippo Sestini. 'Constructing a universe for the setoid model.' In: *FoSSaCS*. 2021, pp. 1–21 (cit. on pp. 9, 14, 78, 90).

[ACKS24] Thorsten Altenkirch, Yorgo Chamoun, Ambrus Kaposi, and Michael Shulman. 'Internal parametricity, without an interval'. In: *Proceedings of the ACM on Programming Languages* 8.POPL (2024), pp. 2340–2369 (cit. on p. 10).

[AK16] Thorsten Altenkirch and Ambrus Kaposi. 'Type theory in type theory using quotient inductive types'. In: *ACM SIGPLAN Notices* 51.1 (2016), pp. 18–29 (cit. on p. 10).

[AKS22] Thorsten Altenkirch, Ambrus Kaposi, and Michael Shulman. *Towards Higher Observational Type Theory*. 28th International Conference on Types for Proofs and Programs (TYPES 2022). 2022 (cit. on p. 10).

[AL19] Benedikt Ahrens and Peter LeFanu Lumsdaine. 'Displayed categories'. In: *Logical Methods in Computer Science* 15 (2019) (cit. on pp. 10, 14, 68, 74, 97).

[ALN25] Benedikt Ahrens, Peter LeFanu Lumsdaine, and Paige Randall North. 'Comparing Semantic Frameworks for Dependently-Sorted Algebraic Theories'. In: *Programming Languages and Systems*. Ed. by Oleg Kiselyov. Singapore: Springer Nature Singapore, 2025, pp. 3–22. ISBN: 978-981-97-8943-6 (cit. on pp. 9, 36).

[Alt21] Thorsten Altenkirch. 'Martin Hofmann's contributions to type theory: Groupoids and univalence'. In: *Mathematical Structures in Computer Science* 31.9 (2021), pp. 953–957 (cit. on p. 10).

[Alt99] Thorsten Altenkirch. 'Extensional equality in intensional type theory'. In: *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*. IEEE. 1999, pp. 412–420 (cit. on pp. 9–10, 78, 149).

[AMS07] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. 'Observational equality, now!' In: *Proceedings of the 2007 workshop on Programming languages meets program verification*. 2007, pp. 57–68 (cit. on p. 10).

[AN22] Benedikt Ahrens and Paige Randall North. 'Univalent foundations and the equivalence principle'. In: *arXiv preprint arXiv:2202.01892* (2022) (cit. on p. 142).

[ANvdW23] Benedikt Ahrens, Paige Randall North, and Niels van der Weide. 'Bicategorical type theory: semantics and syntax'. In: *Mathematical Structures in Computer Science* 33.10 (2023). ISSN: 1469-8072 (cit. on pp. 4, 11).

[Awo10] Steve Awodey. *Category theory*. 2nd ed. Oxford university press, 2010 (cit. on p. 193).

[Awo18] Steve Awodey. 'Natural models of homotopy type theory'. In: *Mathematical Structures in Computer Science* 28.2 (2018), pp. 241–286 (cit. on pp. 9, 36).

[BCDE20] Marc Bezem, Thierry Coquand, Peter Dybjer, and Martín Escardó. 'A Note on Generalized Algebraic Theories and Categories with Families'. In: *arXiv preprint arXiv:2012.08370* (2020) (cit. on p. 9).

[BCH14] Marc Bezem, Thierry Coquand, and Simon Huber. 'A Model of Type Theory in Cubical Sets'. In: *19th International Conference on Types for Proofs and Programs (TYPES 2013)*. Ed. by Ralph Matthes and Aleksy Schubert. Vol. 26. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014, pp. 107–128. ISBN: 978-3-939897-72-9. DOI: 10.4230/LIPIcs.TYPES.2013.107. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2013.107 (cit. on pp. 9–10, 43).

[BD08] Alexandre Buisse and Peter Dybjer. 'Towards formalizing categorical models of type theory in type theory'. In: *Electronic Notes in Theoretical Computer Science* 196 (2008), pp. 137–151 (cit. on p. 9).

[Bou54] N Bourbaki. 'LA TRIBU No 34. Congrès super-œcuménique du frigidaire et des revêtements troués, Murols (17–31 août 1954)'. In: *Archives de l'Association des Collaborateurs de Nicolas Bourbaki* (1954) (cit. on p. 1).

[Car78] John Cartmell. 'Generalised algebraic theories and contextual categories'. PhD thesis. Oxford, 1978 (cit. on p. 9).

[Car86] John Cartmell.
'Generalised algebraic theories and contextual categories'.
In: *Annals of pure and applied logic* 32 (1986), pp. 209–243
(cit. on pp. 9, 28).

[CCD21] Simon Castellan, Pierre Clairambault, and Peter Dybjer. 'Categories with families: Unityped, simply typed, and dependently typed'.
In: *Joachim Lambek: The Interplay of Mathematics, Logic, and Linguistics* (2021), pp. 135–180 (cit. on p. 73).

[CCHM18] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg.
'Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom'.
In: *21st International Conference on Types for Proofs and Programs*. 2018
(cit. on pp. 9–10, 43).

[CMN24] Fernando Chu, Éléonore Mangel, and Paige Randall North.
'A directed type theory for 1-categories'. In: *30th International Conference on Types for Proofs and Programs TYPES 2024–Abstracts*. 2024, p. 205 (cit. on p. 13).

[Cor03] Leo Corry. *Modern algebra and the rise of mathematical structures*.
Springer Science & Business Media, 2003 (cit. on pp. 1–2).

[Ded63] Richard Dedekind. 'The Nature and Meaning of Numbers'.
In: *Essays on the theory of numbers*. Trans. by Wooster Woodruff Beman.
Dover Books on Mathematics.
New York, New York, USA: Dover Publications, 1963, pp. 31–105
(cit. on p. 22).
Trans. of *Was sind und was sollen die Zahlen?* German. Braunschweig: Vieweg, 1888. DOI: 10.24355/dbbs.084-200902200100-1.

[Dij17] Gabe Dijkstra. 'Quotient inductive-inductive definitions.'
PhD thesis. University of Nottingham, 2017 (cit. on p. 10).

[Dyb95] Peter Dybjer. 'Internal type theory'.
In: *International Workshop on Types for Proofs and Programs*.
Springer. 1995, pp. 120–134 (cit. on p. 9).

[EM45] Samuel Eilenberg and Saunders MacLane.
'General theory of natural equivalences'. In: *Transactions of the American Mathematical Society* 58.2 (1945), pp. 231–294
(cit. on pp. 2, 86).

[GKNB20] Daniel Gratzer, GA Kavvos, Andreas Nuyts, and Lars Birkedal.
'Multimodal dependent type theory'. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. 2020, pp. 492–506
(cit. on p. 11).

[GWB24] Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. *Directed univalence in simplicial homotopy type theory*. 2024. arXiv: 2407.09146 [cs.LO]. URL: https://arxiv.org/abs/2407.09146 (cit. on pp. 11, 150, 156).

[GWB25] Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. *The Yoneda embedding in simplicial type theory*. 2025. arXiv: 2501.13229 [cs.LO]. URL: https://arxiv.org/abs/2501.13229 (cit. on p. 11).

[Har92] Godfrey Harold Hardy. *A mathematician's apology*. Cambridge University Press, 1992 (cit. on p. 96).

[Hof95a] Martin Hofmann. 'A simple model for quotient types'. In: *International Conference on Typed Lambda Calculi and Applications*. Springer. 1995, pp. 216–234 (cit. on pp. 9–10, 78).

[Hof95b] Martin Hofmann. 'Extensional concepts in intensional type theory'. PhD thesis. University of Edinburgh, 1995 (cit. on pp. 10, 149).

[Hof97] Martin Hofmann. 'Syntax and semantics of dependent types'. In: *Extensional Constructs in Intensional Type Theory*. Springer, 1997, pp. 13–54 (cit. on pp. 9, 43, 100).

[Hof99] Martin Hofmann. 'Semantical analysis of higher-order abstract syntax'. In: *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*. IEEE. 1999, pp. 204–213 (cit. on p. 10).

[HS95] Martin Hofmann and Thomas Streicher. 'The groupoid interpretation of type theory'. In: *Twenty-five years of constructive type theory (Venice, 1995)* 36 (1995), pp. 83–111 (cit. on pp. 3, 9–10, 80, 101, 103, 138, 149, 158).

[HS99] Martin Hofmann and Thomas Streicher. 'Lifting grothendieck universes'. In: *Unpublished note* 199 (1999), p. 3 (cit. on p. 9).

[Jac93] Bart Jacobs. 'Comprehension categories and the semantics of type dependency'. In: *Theoretical Computer Science* 107.2 (1993), pp. 169–207 (cit. on p. 9).

[Kap17] Ambrus Kaposi. 'Type theory in a type theory with quotient inductive types'. PhD thesis. University of Nottingham, 2017 (cit. on p. 10).

[KKA19] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. 'Constructing quotient inductive-inductive types'. In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019). DOI: 10.1145/3290315. URL: https://doi.org/10.1145/3290315 (cit. on pp. 9–10, 14, 25, 34–35, 56, 60, 63, 65, 70–71).

[KRW23] Nikolai Kudasov, Emily Riehl, and Jonathan Weinberger. *Formalizing the ∞-Categorical Yoneda Lemma*. 2023. arXiv: 2309.08340 [math.CT] (cit. on p. 11).

[KX24] Ambrus Kaposi and Szumi Xie. 'Second-Order Generalised Algebraic Theories: Signatures and First-Order Semantics'. In: *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2024 (cit. on pp. 10, 209).

[Le 16] Lieven Le Bruyn. *le lemme de la Gare du Nord*. Nov. 2016. URL: http://www.neverendingbooks.org/le-lemme-de-la-gare-du-nord (cit. on p. 1).

[Le 22] Lieven Le Bruyn. *Le Guide Bourbaki: Murol(s)*. July 2022. URL: http://www.neverendingbooks.org/le-guide-bourbaki-murols (cit. on p. 1).

[LH11] Daniel R Licata and Robert Harper. '2-Dimensional Directed Dependent Type Theory'. In: (2011) (cit. on pp. 4, 6, 11, 15, 93, 103).

[LLV24] Andrea Laretto, Fosco Loregian, and Niccolò Veltri. 'Directed equality with dinaturality'. In: *arXiv preprint arXiv:2409.10237* (2024) (cit. on p. 12).

[Lor23] Fosco Loregian. *Coend calculus*. 2023. arXiv: 1501.02503 [math.CT] (cit. on p. 12).

[Mac78] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer New York, 1978. DOI: 10.1007/978-1-4757-4721-8. URL: https://doi.org/10.1007/978-1-4757-4721-8 (cit. on p. 203).

[Man24] Éléonore Mangel. *A directed homotopy type theory for 1-categories*. unpublished. 2024 (cit. on p. 13).

[Mar75] Per Martin-Löf. 'An Intuitionistic Theory of Types: Predicative Part'. In: *Logic Colloquium '73*. Ed. by H.E. Rose and J.C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. Elsevier, 1975, pp. 73–118 (cit. on p. 2).

[Mar82] Per Martin-Löf. 'Constructive Mathematics and Computer Programming'. In: *Logic, Methodology and Philosophy of Science VI*. Ed. by L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski. Vol. 104. Studies in Logic and the Foundations of Mathematics. Elsevier, 1982, pp. 153–175 (cit. on p. 2).

[Mog91] Eugenio Moggi. 'A category-theoretic account of program modules'. In: *Mathematical Structures in Computer Science* 1.1 (1991), pp. 103–139 (cit. on p. 9).

[NA24] Jacob Neumann and Thorsten Altenkirch. 'Synthetic 1-Categories in Directed Type Theory'. In: *arXiv preprint arXiv:2410.19520* (2024) (cit. on pp. 13, 120, 130).

[NA25] Jacob Neumann and Thorsten Altenkirch.
'Synthetic 1-Categories in Directed Type Theory'. In: *30th International Conference on Types for Proofs and Programs (TYPES 2024)*.
Ed. by Rasmus Ejlers Møgelberg and Benno van den Berg.
Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025 (cit. on p. 13).

[Nas24] Hayato Nasu. 'An Internal Logic of Virtual Double Categories'.
In: *arXiv preprint arXiv:2410.06792* (2024) (cit. on p. 12).

[Nas25] Hayato Nasu. 'Logical Aspects of Virtual Double Categories'.
In: *arXiv preprint arXiv:2501.17869* (2025) (cit. on p. 12).

[NL23] Max S New and Daniel R Licata.
'A Formal Logic for Formal Category Theory'. In: *Foundations of Software Science and Computation Structures LNCS 13992* (2023), p. 113 (cit. on p. 12).

[Nor19] Paige Randall North. 'Towards a directed homotopy type theory'. In: *Electronic Notes in Theoretical Computer Science* 347 (2019), pp. 223–239 (cit. on pp. 4, 6, 12, 15, 85, 92, 117, 134, 136).

[Nuy15] Andreas Nuyts.
'Towards a directed homotopy type theory based on 4 kinds of variance'.
In: *Mém. de mast. Katholieke Universiteit Leuven* (2015) (cit. on pp. 4, 11).

[Rey83] John C Reynolds. 'Types, abstraction and parametric polymorphism'.
In: *Information Processing 83, Proceedings of the IFIP 9th World Computer Congres*. 1983, pp. 513–523 (cit. on pp. 35, 72).

[Rij22] Egbert Rijke. *Introduction to Homotopy Type Theory*. 2022.
arXiv: 2212.11082 [math.LO].
URL: https://arxiv.org/abs/2212.11082
(cit. on pp. iii, 7, 154).

[RS17] E. Riehl and M. Shulman. 'A type theory for synthetic ∞-categories'.
In: *Higher Structures* 1.1 (2017), pp. 116–193.
DOI: 10.2140/hs.2017.1.116. eprint: arXiv:1705.07442
(cit. on pp. 4, 11).

[Shu22] Michael Shulman. *Towards a Third-Generation HOTT*.
Carnegie Mellon University HoTT Seminar. 2022.
URL: https://www.cmu.edu/dietrich/philosophy/hott/seminars/index.html (cit. on p. 10).

[Str93] Thomas Streicher. 'Investigations into intensional type theory'.
In: *Habilitiation Thesis, Ludwig Maximilian Universität* (1993), p. 57 (cit. on p. 3).

[Uem21] Taichi Uemura. 'Abstract and concrete type theories'.
PhD thesis. University of Amsterdam, 2021 (cit. on pp. 10, 29).

[Uem23] Taichi Uemura. 'A general framework for the semantics of type theory'. In: *Mathematical Structures in Computer Science* 33.3 (Mar. 2023). ISSN: 1469-8072. DOI: 10.1017/s0960129523000208. URL: http://dx.doi.org/10.1017/S0960129523000208 (cit. on p. 10).

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL: https://homotopytypetheory.org/book (cit. on pp. 7, 10, 165–166).

[VG11] Benno Van Den Berg and Richard Garner. 'Types are weak ω-groupoids'. In: *Proceedings of the london mathematical society* 102.2 (2011), pp. 370–394 (cit. on p. 11).

[Voe14a] Vladimir Voevodsky. *The equivalence axiom and univalent models of type theory. (Talk at CMU on February 4, 2010)*. 2014. arXiv: 1402.5556 [math.LO]. URL: https://arxiv.org/abs/1402.5556 (cit. on p. 10).

[Voe14b] Vladimir Voevodsky. 'The origins and motivations of univalent foundations'. In: *The Institute Letter* (2014), pp. 8–9 (cit. on p. 10).

[Wea24] Matthew Zachary Weaver. 'Bicubical Directed Type Theory'. PhD thesis. Princeton University, 2024 (cit. on p. 11).

[Wei22a] Jonathan Weinberger. 'A Synthetic Perspective on (∞,1)-Category Theory: Fibrational and Semantic Aspects'. In: (2022). DOI: 10.26083/TUPRINTS-00020716. URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/20716 (cit. on p. 11).

[Wei22b] Jonathan Weinberger. *Strict stability of extension types*. 2022. arXiv: 2203.07194 [math.CT]. URL: https://arxiv.org/abs/2203.07194 (cit. on p. 11).

[Wei24a] Jonathan Weinberger. 'Internal sums for synthetic fibered (∞, 1)-categories'. In: *Journal of Pure and Applied Algebra* 228.9 (2024), p. 107659 (cit. on p. 11).

[Wei24b] Jonathan Weinberger. 'Two-sided cartesian fibrations of synthetic (∞, 1)-categories'. In: *Journal of Homotopy and Related Structures* 19.3 (2024), pp. 297–378 (cit. on p. 11).

[WL20] Matthew Z. Weaver and Daniel R. Licata.
'A Constructive Model of Directed Univalence in Bicubical Sets'.
In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20.
Saarbrücken, Germany: Association for Computing Machinery, 2020, pp. 915–928. ISBN: 9781450371049.
DOI: 10.1145/3373718.3394794.
URL: https://doi.org/10.1145/3373718.3394794
(cit. on p. 11).

Appendix A

Generalized Algebraic Theories

A.1 Basic structures

A.1.1 Sets

onegat

◇
▹ U

pseudoAgda

```
record Set-Alg where
  X : Set
```

A.1.2 Pointed Sets

onegat

◇
▹ U
▹ El 0

pseudoAgda

```
record 𝔓-Alg where
  X : Set
  x : X
```

A.1.3 Bipointed Sets

oneGAT

```
◇
▹ U
▹ El 0
▹ El 1
```

pseudoAgda

```
record 𝔅-Alg where
  X : Set
  x₀ x₁ : X
```

A.1.4 Natural Number Algebras

oneGAT

```
◇
▹ U
▹ El 0
▹ Π 1 (El 2)
```

pseudoAgda

```
record 𝔑-Alg where
  Nat : Set
  zero : Nat
  succ : Nat ⟶ Nat
```

A.1.5 Even-Odd Algebras

ONEGAT

```
◇
▸ U
▸ U
▸ El 1
▸ Π 2 (El 2)
▸ Π 2 (El 4)
```

PSEUDOAGDA

```
record 𝔈𝔒-Alg where
  E : Set
  O : Set
  z : E
  s : E ⟶ O
  s' : O ⟶ E
```

A.1.6 Monoids

ONEGAT

```
◇
▸ U
▸ El 0
▸ Π 1 (Π 2 (El 3))
▸ Π 2 (Eq (1 @ 2 @ 0) 0)
▸ Π 3 (Eq (2 @ 0 @ 3) 0)
▸ Π 4 (Π 5 (Π 6 (Eq (5 @ 2 @ (5 @ 1 @ 0)) (5 @ (5 @ 2 @ 1) @ 0))))
```

PSEUDOAGDA

```
record 𝔐on-Alg where
  M : Set
  u : M
  m : M ⟶ M ⟶ M
  lunit : (x : M) ⟶ m u x = x
  runit : (x : M) ⟶ m x u = x
  assoc : (x : M) ⟶ (y : M) ⟶ (z : M) ⟶ m x (m y z) = m (m x y) z
```

A.1.7 Groups

oneGAT

```
◇
▹ U
▹ El 0
▹ Π 1 (Π 2 (El 3))
▹ Π 2 (Eq (1 @ 2 @ 0) 0)
▹ Π 3 (Eq (2 @ 0 @ 3) 0)
▹ Π 4 (Π 5 (Π 6 (Eq (5 @ 2 @ (5 @ 1 @ 0)) (5 @ (5 @ 2 @ 1) @ 0))))
▹ Π 5 (El 6)
▹ Π 6 (Eq (5 @ (1 @ 0) @ 0) 6)
▹ Π 7 (Eq (6 @ 0 @ (2 @ 0)) 7)
```

pseudoAgda

```
record Grp-Alg where
  M : Set
  u : M
  m : M → M → M
  lunit : (x : M) → m u x = x
  runit : (x : M) → m x u = x
  assoc : (x : M) → (y : M) → (z : M) → m x (m y z) = m (m x y) z
  inv : M → M
  linv : (x : M) → m (inv x) x = u
  rinv : (x : M) → m x (inv x) = u
```

A.2 Quiver-like structures

A.2.1 Quivers

oneGAT

```
◇
▹ U
▹ Π 0 (Π 1 U)
```

pseudoAgda

```
record 𝔔uiv-Alg where
  V : Set
  E : V ⟶ V ⟶ Set
```

A.2.2 Reflexive Quivers

oneGAT

```
◇
▹ U
▹ Π 0 (Π 1 U)
▹ Π 1 (El (1 @ 0 @ 0))
```

pseudoAgda

```
record r𝔔uiv-Alg where
  V : Set
  E : V ⟶ V ⟶ Set
  r : (v : V) ⟶ E v v
```

A.2.3 Preorders

oneGAT

```
◇
▹ U
▹ Π 0 (Π 1 U)
▹ Π 1 (Π 2 (Π (2 @ 1 @ 0) (Π (3 @ 2 @ 1) (Eq 1 0))))
▹ Π 2 (El (2 @ 0 @ 0))
▹ Π 3 (Π 4 (Π 5 (Π (5 @ 2 @ 1) (Π (6 @ 2 @ 1) (El (7 @ 4 @ 2))))))
```

pseudoAgda

```
record PreOrd-Alg where
  X : Set
  leq : X → X → Set
  leqη : (x : X) → (x' : X) → (p : leq x x') → (q : leq x x') → p = q
  rfl : (x : X) → leq x x
  trns : (x : X) → (y : X) → (z : X) → leq x y → leq y z → leq x z
```

A.2.4 Setoids

oneGAT

```
◇
▹ U
▹ Π 0 (Π 1 U)
▹ Π 1 (Π 2 (Π (2 @ 1 @ 0) (Π (3 @ 2 @ 1) (Eq 1 0))))
▹ Π 2 (El (2 @ 0 @ 0))
▹ Π 3 (Π 4 (Π (4 @ 1 @ 0) (El (5 @ 1 @ 2))))
▹ Π 4 (Π 5 (Π 6 (Π (6 @ 2 @ 1) (Π (7 @ 2 @ 1) (El (8 @ 4 @ 2))))))
```

pseudoAgda

```
record Setoid-Alg where
  X : Set
  eq : X → X → Set
  eqη : (x : X) → (x' : X) → (p : eq x x') → (q : eq x x') → p = q
  rfl : (x : X) → eq x x
  sym : (x : X) → (y : X) → eq x y → eq y x
  trns : (x : X) → (y : X) → (z : X) → eq x y → eq y z → eq x z
```

A.2.5 Categories

oneGAT

```
◇
▹ U
▹ Π 0 (Π 1 U)
▹ Π 1 (El (1 @ 0 @ 0))
▹ Π 2 (Π 3 (Π 4 (Π (4 @ 1 @ 0) (Π (5 @ 3 @ 2) (El (6 @ 4 @ 2))))))
▹ Π 3 (Π 4 (Π (4 @ 1 @ 0) (Eq (3 @ 2 @ 1 @ 1 @ (4 @ 1) @ 0) 0)))
▹ Π 4 (Π 5 (Π (5 @ 1 @ 0) (Eq (4 @ 2 @ 2 @ 1 @ 0 @ (5 @ 2)) 0)))
▹ Π 5 (Π 6 (Π 7 (Π 8 (Π (8 @ 3 @ 2) (Π (9 @ 3 @ 2) (Π (10 @ 3 @ 2) (Eq
    (9 @ 6 @ 5 @ 3 @ 0 @ (9 @ 6 @ 5 @ 4 @ 1 @ 2)) (9 @ 6 @ 4 @ 3
    @ (9 @ 5 @ 4 @ 3 @ 0 @ 1) @ 2))))))))
```

pseudoAgda

```
record 𝔈at-Alg where
  Obj : Set
  Hom : Obj → Obj → Set
  id : (X : Obj) → Hom X X
  comp : {X Y Z : Obj} → Hom Y Z → Hom X Y → Hom X Z
  lunit : {X Y : Obj} → (f : Hom X Y) → comp (id Y) f = f
  runit : {X Y : Obj} → (f : Hom X Y) → comp f (id X) = f
  assoc : {W X Y Z : Obj} →
    (e : Hom W X) → (f : Hom X Y) → (g : Hom Y Z) →
    comp g (comp f e) = comp (comp g f) e
```

A.2.6 Groupoids

ONEGAT

```
◇
▹ U
▹ Π 0 (Π 1 U)
▹ Π 1 (El (1 @ 0 @ 0))
▹ Π 2 (Π 3 (Π 4 (Π (4 @ 1 @ 0) (Π (5 @ 3 @ 2) (El (6 @ 4 @ 2))))))
▹ Π 3 (Π 4 (Π (4 @ 1 @ 0) (Eq (3 @ 2 @ 1 @ 1 @ (4 @ 1) @ 0) 0)))
▹ Π 4 (Π 5 (Π (5 @ 1 @ 0) (Eq (4 @ 2 @ 2 @ 1 @ 0 @ (5 @ 2)) 0)))
▹ Π 5 (Π 6 (Π 7 (Π 8 (Π (8 @ 3 @ 2) (Π (9 @ 3 @ 2) (Π (10 @ 3 @ 2) (Eq
    (9 @ 6 @ 5 @ 3 @ 0 @ (9 @ 6 @ 5 @ 4 @ 1 @ 2)) (9 @ 6 @ 4 @ 3
    @ (9 @ 5 @ 4 @ 3 @ 0 @ 1) @ 2))))))))
▹ Π 6 (Π 7 (Π (7 @ 1 @ 0) (El (8 @ 1 @ 2))))
▹ Π 7 (Π 8 (Π (8 @ 1 @ 0) (Eq (7 @ 2 @ 1 @ 2 @ (3 @ 2 @ 1 @ 0) @ 0)
    (8 @ 2))))
▹ Π 8 (Π 9 (Π (9 @ 1 @ 0) (Eq (8 @ 1 @ 2 @ 1 @ 0 @ (4 @ 2 @ 1 @ 0))
    (9 @ 1))))
```

PSEUDOAGDA

```
record 𝔊𝔯𝔭𝔡-Alg where
  Obj : Set
  Hom : Obj → Obj → Set
  id : (X : Obj) → Hom X X
  comp : {X Y Z : Obj} → Hom Y Z → Hom X Y → Hom X Z
  lunit : {X Y : Obj} → (f : Hom X Y) → comp (id Y) f = f
  runit : {X Y : Obj} → (f : Hom X Y) → comp f (id X) = f
  assoc : {W X Y Z : Obj} →
    (e : Hom W X) → (f : Hom X Y) → (g : Hom Y Z) →
    comp g (comp f e) = comp (comp g f) e
  inv : {X Y : Obj} → Hom X Y → Hom Y X
  linv : {X Y : Obj} → (f : Hom X Y) → comp (inv f) f = id X
  rinv : {X Y : Obj} → (f : Hom X Y) → comp f (inv f) = id Y
```

A.3 Models of Type Theory

A.3.1 Categories with Families (CwFs)

onegat

𝔈at
▹ El 6
▹ Π 7 (El (7 @ 0 @ 1))
▹ Π 8 (Π (8 @ 0 @ 2) (Eq 0 (2 @ 1)))
▹ Π 9 U
▹ Π 10 (Π 11 (Π (11 @ 1 @ 0) (Π (3 @ 1) (El (4 @ 3)))))
▹ Π 11 (Π (2 @ 0) (Eq (2 @ 1 @ 1 @ (11 @ 1) @ 0) 0))
▹ Π 12 (Π 13 (Π 14 (Π (5 @ 0) (Π (15 @ 3 @ 2) (Π (16 @ 3 @ 2) (Eq (7 @ 4 @ 3 @ 0 @ (7 @ 5 @ 4 @ 1 @ 2)) (7 @ 5 @ 3 @ (15 @ 5 @ 4 @ 3 @ 0 @ 1) @ 2)))))))
▹ Π 13 (Π (4 @ 0) U)
▹ Π 14 (Π 15 (Π (6 @ 0) (Π (16 @ 2 @ 1) (Π (4 @ 2 @ 1) (El (5 @ 4 @ (8 @ 4 @ 3 @ 1 @ 2)))))))
▹ Π 15 (Π (6 @ 0) (Π (3 @ 1 @ 0) (Eq (transp (6 @ 2 @ 1) (3 @ 2 @ 2 @ 1 @ (16 @ 2) @ 0)) 0)))
▹ Π 16 (Π 17 (Π 18 (Π (9 @ 0) (Π (6 @ 1 @ 0) (Π (20 @ 4 @ 3) (Π (21 @ 4 @ 3) (Eq (transp (10 @ 6 @ 5 @ 4 @ 3 @ 0 @ 1) (8 @ 5 @ 4 @ 3 @ 0 @ (8 @ 6 @ 5 @ (12 @ 5 @ 4 @ 0 @ 3) @ 1 @ 2))) (8 @ 6 @ 4 @ 3 @ (20 @ 6 @ 5 @ 4 @ 0 @ 1) @ 2))))))))
▹ Π 17 (Π (8 @ 0) (El 19))
▹ Π 18 (Π 19 (Π (10 @ 0) (Π (20 @ 2 @ 1) (Π (8 @ 3 @ (11 @ 3 @ 2 @ 0 @ 1)) (El (22 @ 4 @ (5 @ 3 @ 2)))))))
▹ Π 19 (Π 20 (Π 21 (Π (12 @ 0) (Π (22 @ 2 @ 1) (Π (10 @ 3 @ (13 @ 3 @ 2 @ 0 @ 1)) (Π (24 @ 5 @ 4) (Eq (23 @ 6 @ 5 @ (8 @ 4 @ 3) @ (7 @ 5 @ 4 @ 3 @ 2 @ 1) @ 0) (7 @ 6 @ 4 @ 3 @ (23 @ 6 @ 5 @ 4 @ 2 @ 0) @ (transp (13 @ 6 @ 5 @ 4 @ 3 @ 2 @ 0) (11 @ 6 @ 5 @ (15 @ 5 @ 4 @ 2 @ 3) @ 0 @ 1)))))))))))
▹ Π 20 (Π (11 @ 0) (El (21 @ (4 @ 1 @ 0) @ 1)))
▹ Π 21 (Π (12 @ 0) (El (9 @ (5 @ 1 @ 0) @ (12 @ (5 @ 1 @ 0) @ 1 @ (2 @ 1 @ 0) @ 0))))
▹ Π 22 (Π 23 (Π (14 @ 0) (Π (24 @ 2 @ 1) (Π (12 @ 3 @ (15 @ 3 @ 2 @ 0 @ 1)) (Eq (24 @ 4 @ (9 @ 3 @ 2) @ 3 @ (6 @ 3 @ 2) @ (8 @ 4 @ 3 @ 2 @ 1 @ 0)) 1)))))
▹ Π 23 (Π 24 (Π (15 @ 0) (Π (25 @ 2 @ 1) (Π (13 @ 3 @ (16 @ 3 @ 2 @ 0 @ 1)) (Eq (transp (5 @ 4 @ 3 @ 2 @ 1 @ 0) (transp (15 @ 4 @ (10 @ 3 @ 2) @ 3 @ 2 @ (7 @ 3 @ 2) @ (9 @ 4 @ 3 @ 2 @ 1 @ 0)) (13 @ 4 @ (10 @ 3 @ 2) @ (17 @ 4 @ 3 @ 1 @ 2) @ (9 @ 4 @ 3 @ 2 @ 1 @ 0) @ (6 @ 3 @ 2)))) 0)))))
▹ Π 24 (Π (15 @ 0) (Eq (7 @ (8 @ 1 @ 0) @ 1 @ 0 @ (5 @ 1 @ 0) @ (4 @ 1 @ 0)) (24 @ (8 @ 1 @ 0))))

pseudoAgda

```
record 𝔈𝔴𝔉-Alg where
  Con : Set
  Sub : Con → Con → Set
  id : (X : Con) → Sub X X
  comp : {X Y Z : Con} → Sub Y Z → Sub X Y → Sub X Z
  lunit : {X Y : Con} → (f : Sub X Y) → comp (id Y) f = f
  runit : {X Y : Con} → (f : Sub X Y) → comp f (id X) = f
  assoc : {W X Y Z : Con} →
    (e : Sub W X) → (f : Sub X Y) → (g : Sub Y Z) →
    comp g (comp f e) = comp (comp g f) e
  empty : Con
  ϵ : (Γ : Con) → Sub Γ empty
  ηϵ : {Γ : Con} → (f : Sub Γ empty) → f = ϵ Γ
  Ty : Con → Set
  substTy : {Δ Γ : Con} → Sub Δ Γ → Ty Γ → Ty Δ
  idTy : {Γ : Con} → (A : Ty Γ) → substTy (id Γ) A = A
  compTy : {Θ Δ Γ : Con} → (A : Ty Γ) →
    (δ : Sub Θ Δ) → (γ : Sub Δ Γ) →
    substTy γ (substTy δ A) = substTy (comp γ δ) A
  Tm : (Γ : Con) → Ty Γ → Set
  substTm : {Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → Tm Γ A → Tm Δ (substTy γ A)
  idTm : {Γ : Con}{A : Ty Γ} → (t : Tm Γ A) → substTm (id Γ) t = t
  compTm : {Θ Δ Γ : Con}{A : Ty Γ} → (t : Tm Γ A) →
    (δ : Sub Θ Δ) → (γ : Sub Δ Γ) →
    substTm γ (substTm δ t) = substTm (comp γ δ) t
  ext : (Γ : Con) → Ty Γ → Con
  pair : {Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → Tm Δ (substTy γ A) → Sub Δ (ext Γ A)
  pair_nat : {Θ Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → (t : Tm Δ (substTy γ A)) → (δ : Sub Θ Δ) →
    comp (pair γ t) δ = pair (comp γ δ) (substTm δ t)
  p : {Γ : Con} → (A : Ty Γ) → Sub (ext Γ A) Γ
  v : {Γ : Con} → (A : Ty Γ) → Tm (ext Γ A) (substTy (p A) A)
  ext_β₁ : {Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → (t : Tm Δ (substTy γ A)) →
    comp (p A) (pair γ t) = γ
  ext_β₂ : {Δ Γ : Con}{A : Ty Γ} →
    (γ : Sub Δ Γ) → (t : Tm Δ (substTy γ A)) →
    substTm (pair γ t) (v A) = t
  ext_η : {Γ : Con}{A : Ty Γ} → pair (p A) (v A) = id (ext Γ A)
```

A.3.2 CwFs /w unit type

oneGAT

$\mathfrak{CwF}$
- Π 25 (El (16 @ 0))
- Π 26 (Π 27 (Π (27 @ 1 @ 0) (Eq (18 @ 2 @ 1 @ 0 @ (3 @ 1)) (3 @ 2))))
- Π 27 (El (14 @ 0 @ (2 @ 0)))
- Π 28 (Π 29 (Π (29 @ 1 @ 0) (Eq (transp (4 @ 2 @ 1 @ 0) (16 @ 2 @ 1 @ (5 @ 1) @ 0 @ (3 @ 1))) (3 @ 2))))
- Π 29 (Π (20 @ (12 @ 0 @ (4 @ 0))) (Π (17 @ 1 @ (20 @ 1 @ (13 @ 1 @ (5 @ 1)) @ (12 @ 1 @ 1 @ (5 @ 1) @ (29 @ 1) @ (3 @ 1)) @ 0)) (El (18 @ (14 @ 2 @ (6 @ 2)) @ 1))))
- Π 30 (Π (21 @ (13 @ 0 @ (5 @ 0))) (Π (18 @ 1 @ (21 @ 1 @ (14 @ 1 @ (6 @ 1)) @ (13 @ 1 @ 1 @ (6 @ 1) @ (30 @ 1) @ (4 @ 1)) @ 0)) (Eq (18 @ 2 @ (15 @ 2 @ (7 @ 2)) @ 1 @ (14 @ 2 @ 2 @ (7 @ 2) @ (31 @ 2) @ (5 @ 2)) @ (3 @ 2 @ 1 @ 0)) 0)))

A.3.3 CwFs /w bool type

oneGAT

$\mathfrak{CwF}$
- ▹ Π 25 (El (16 @ 0))
- ▹ Π 26 (Π 27 (Π (27 @ 1 @ 0) (Eq (18 @ 2 @ 1 @ 0 @ (3 @ 1)) (3 @ 2))))
- ▹ Π 27 (El (14 @ 0 @ (2 @ 0)))
- ▹ Π 28 (Π 29 (Π (29 @ 1 @ 0) (Eq (transp (4 @ 2 @ 1 @ 0) (16 @ 2 @ 1 @ (5 @ 1) @ 0 @ (3 @ 1))) (3 @ 2))))
- ▹ Π 29 (El (16 @ 0 @ (4 @ 0)))
- ▹ Π 30 (Π 31 (Π (31 @ 1 @ 0) (Eq (transp (6 @ 2 @ 1 @ 0) (18 @ 2 @ 1 @ (7 @ 1) @ 0 @ (3 @ 1))) (3 @ 2))))
- ▹ Π 31 (Π (22 @ (14 @ 0 @ (6 @ 0))) (Π (19 @ 1 @ (22 @ 1 @ (15 @ 1 @ (7 @ 1)) @ (14 @ 1 @ 1 @ (7 @ 1) @ (31 @ 1) @ (5 @ 1)) @ 0)) (Π (20 @ 2 @ (23 @ 2 @ (16 @ 2 @ (8 @ 2)) @ (15 @ 2 @ 2 @ (8 @ 2) @ (32 @ 2) @ (4 @ 2)) @ 1)) (El (21 @ (17 @ 3 @ (9 @ 3)) @ 2)))))
- ▹ Π 32 (Π (23 @ (15 @ 0 @ (7 @ 0))) (Π (20 @ 1 @ (23 @ 1 @ (16 @ 1 @ (8 @ 1)) @ (15 @ 1 @ 1 @ (8 @ 1) @ (32 @ 1) @ (6 @ 1)) @ 0)) (Π (21 @ 2 @ (24 @ 2 @ (17 @ 2 @ (9 @ 2)) @ (16 @ 2 @ 2 @ (9 @ 2) @ (33 @ 2) @ (5 @ 2)) @ 1)) (Eq (21 @ 3 @ (18 @ 3 @ (10 @ 3)) @ 2 @ (17 @ 3 @ 3 @ (10 @ 3) @ (34 @ 3) @ (8 @ 3)) @ (4 @ 3 @ 2 @ 1 @ 0)) 1))))
- ▹ Π 33 (Π (24 @ (16 @ 0 @ (8 @ 0))) (Π (21 @ 1 @ (24 @ 1 @ (17 @ 1 @ (9 @ 1)) @ (16 @ 1 @ 1 @ (9 @ 1) @ (33 @ 1) @ (7 @ 1)) @ 0)) (Π (22 @ 2 @ (25 @ 2 @ (18 @ 2 @ (10 @ 2)) @ (17 @ 2 @ 2 @ (10 @ 2) @ (34 @ 2) @ (6 @ 2)) @ 1)) (Eq (22 @ 3 @ (19 @ 3 @ (11 @ 3)) @ 2 @ (18 @ 3 @ 3 @ (11 @ 3) @ (35 @ 3) @ (7 @ 3)) @ (5 @ 3 @ 2 @ 1 @ 0)) 0))))

A.3.4 CwFs /w Π-type

oneGAT

$\mathfrak{CwF}$
- ▹ Π 25 (Π (16 @ 0) (Π (17 @ (9 @ 1 @ 0)) (El (18 @ 2))))
- ▹ Π 26 (Π 27 (Π (27 @ 1 @ 0) (Π (19 @ 1) (Π (20 @ (12 @ 2 @ 0)) (Eq (20 @ 4 @ 3 @ 2 @ (5 @ 3 @ 1 @ 0)) (5 @ 4 @ (20 @ 4 @ 3 @ 2 @ 1) @ (20 @ (13 @ 4 @ (20 @ 4 @ 3 @ 2 @ 1)) @ (13 @ 3 @ 1) @ (12 @ (13 @ 4 @ (20 @ 4 @ 3 @ 2 @ 1)) @ 3 @ 1 @ (28 @ (13 @ 4 @ (20 @ 4 @ 3 @ 2 @ 1)) @ 4 @ 3 @ 2 @ (10 @ 4 @ (20 @ 4 @ 3 @ 2 @ 1))) @ (9 @ 4 @ (20 @ 4 @ 3 @ 2 @ 1))) @ 0)))))))
- ▹ Π 27 (Π (18 @ 0) (Π (19 @ (11 @ 1 @ 0)) (Π (16 @ (12 @ 2 @ 1) @ 0) (El (17 @ 3 @ (5 @ 3 @ 2 @ 1))))))
- ▹ Π 28 (Π (19 @ 0) (Π (20 @ (12 @ 1 @ 0)) (Π (17 @ 2 @ (5 @ 2 @ 1 @ 0)) (Π (18 @ 3 @ 2) (El (19 @ 4 @ (22 @ 4 @ (15 @ 4 @ 3) @ (14 @ 4 @ 4 @ 3 @ (31 @ 4) @ (transp^{-1} (21 @ 4 @ 3) (0))) @ 2)))))))
- ▹ Π 29 (Π 30 (Π (30 @ 1 @ 0) (Π (22 @ 1) (Π (23 @ (15 @ 2 @ 0)) (Π (20 @ (16 @ 3 @ 1) @ 0) (Eq (transp (8 @ 5 @ 4 @ 3 @ 2 @ 1) (20 @ 5 @ 4 @ 3 @ (9 @ 4 @ 2 @ 1) @ (7 @ 4 @ 2 @ 1 @ 0))) (7 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2) @ (24 @ (17 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2)) @ (17 @ 4 @ 2) @ (16 @ (17 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2)) @ 4 @ 2 @ (32 @ (17 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2)) @ 5 @ 4 @ 3 @ (14 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2))) @ (13 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2))) @ 1) @ (20 @ (17 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2)) @ (17 @ 4 @ 2) @ (16 @ (17 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2)) @ 4 @ 2 @ (32 @ (17 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2)) @ 5 @ 4 @ 3 @ (14 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2))) @ (13 @ 5 @ (24 @ 5 @ 4 @ 3 @ 2))) @ 1 @ 0))))))))
- ▹ Π 30 (Π (21 @ 0) (Π (22 @ (14 @ 1 @ 0)) (Π (19 @ (15 @ 2 @ 1) @ 0) (Π (20 @ 3 @ 2) (Eq (6 @ 4 @ 3 @ 2 @ (7 @ 4 @ 3 @ 2 @ 1) @ 0) (20 @ 4 @ (17 @ 4 @ 3) @ 2 @ (16 @ 4 @ 4 @ 3 @ (33 @ 4) @ (transp^{-1} (23 @ 4 @ 3) (0))) @ 1))))))
- ▹ Π 31 (Π (22 @ 0) (Π (23 @ (15 @ 1 @ 0)) (Π (20 @ 2 @ (8 @ 2 @ 1 @ 0)) (Eq (7 @ 3 @ 2 @ 1 @ (6 @ (17 @ 3 @ 2) @ (24 @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ 2) @ (24 @ (17 @ (17 @ 3 @ 2) @ (24 @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ 2)) @ (17 @ 3 @ 2) @ (16 @ (17 @ (17 @ 3 @ 2) @ (24 @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ 2)) @ 3 @ 2 @ (32 @ (17 @ (17 @ 3 @ 2) @ (24 @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ 2)) @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ (14 @ (17 @ 3 @ 2) @ (24 @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ 2))) @ (13 @ (17 @ 3 @ 2) @ (24 @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ 2))) @ 1) @ (transp (8 @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ 2 @ 1) (20 @ (17 @ 3 @ 2) @ 3 @ (14 @ 3 @ 2) @ (9 @ 3 @ 2 @ 1) @ 0)) @ (13 @ 3 @ 2))) 0))))

A.4 Models of Polarized and Directed Type Theory

A.4.1 Polarized Categories with Families (PCwFs)

ONEGAT

```
ℭ𝔴𝔉
▹ Π 25 (El 26)
▹ Π 26 (Π 27 (Π (27 @ 1 @ 0) (El (28 @ (3 @ 2) @ (3 @ 1)))))
▹ Π 27 (Π (18 @ 0) (El (19 @ 1)))
▹ Eq (2 @ 21) 21
▹ Π 29 (Eq (3 @ 0 @ 0 @ (28 @ 0)) (28 @ (4 @ 0)))
▹ Π 30 (Π 31 (Π 32 (Π (32 @ 2 @ 1) (Π (33 @ 2 @ 1) (Eq (8 @ 4 @ 2 @
    (32 @ 4 @ 3 @ 2 @ 0 @ 1)) (32 @ (9 @ 4) @ (9 @ 3) @ (9 @ 2) @ (8
    @ 3 @ 2 @ 0) @ (8 @ 4 @ 3 @ 1)))))))
▹ Π 31 (Π 32 (Π (32 @ 1 @ 0) (Π (24 @ 1) (Eq (7 @ 3 @ (24 @ 3 @ 2 @ 1
    @ 0)) (24 @ 3 @ 2 @ 1 @ (7 @ 2 @ 0)))))))
▹ Π 32 (Eq (7 @ (7 @ 0)) 0)
▹ Π 33 (Π 34 (Π (34 @ 1 @ 0) (Eq (transp (3 @ 1) (transp (3 @ 2) (9 @
    (10 @ 2) @ (10 @ 1) @ (9 @ 2 @ 1 @ 0)))) 0)))
▹ Π 34 (Π (25 @ 0) (Eq (8 @ 1 @ (8 @ 1 @ 0)) 0))
```

PSEUDOAGDA

```
record 𝔓ℭ𝔴𝔉-Alg where
  ...
  neg_Con : Con → Con
  neg_Sub : {Δ Γ : Con} → Sub Δ Γ → Sub (neg_Con Δ) (neg_Con Γ)
  neg_Ty : {Γ : Con} → Ty Γ → Ty Γ
  neg_empty : neg_Con empty = empty
  neg_id : {Γ : Con} → neg_Sub (id Γ) = id (neg_Con Γ)
  neg_comp : {Θ Δ Γ : Con} → (δ : Sub Θ Δ) → (γ : Sub Δ Γ) →
    neg_Sub (comp γ δ) = comp (neg_Sub γ) (neg_Sub δ)
  neg_nat : {Δ Γ : Con} → (γ : Sub Δ Γ) → (A : Ty Γ) →
    neg_Ty (substTy γ A) = substTy γ (neg_Ty A)
  invl_Con : (Γ : Con) → neg_Con (neg_Con Γ) = Γ
  invl_Sub : {Δ Γ : Con} → (γ : Sub Δ Γ) → neg_Sub (neg_Sub γ) = γ
  invl_Ty : {Γ : Con} → (A : Ty Γ) → neg_Ty (neg_Ty A) = A
```