First Steps in Synthetic Tait Computability

The Objective Metatheory of Cubical Type Theory

Jonathan Sterling

CMU-CS-21-142

October 2021

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

Thesis Committee:
Robert Harper, Chair*
Jeremy Avigad*
Karl Crary*
Lars Birkedal[†]
Kuen-Bang Hou (Favonia)[‡]

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

* Carnegie Mellon University   [†] Aarhus University   [‡] University of Minnesota

Abstract

The implementation and semantics of dependent type theories can be studied in a syntax-independent way: the *objective metatheory* of dependent type theories exploits the universal properties of their syntactic categories to endow them with computational content, mathematical meaning, and practical implementation (normalization, type checking, elaboration). The semantic methods of the objective metatheory inform the design and implementation of *correct-by-construction* elaboration algorithms, promising a principled interface between real proof assistants and ideal mathematics.

In this dissertation, I add *synthetic Tait computability* to the arsenal of the objective metatheorist. Synthetic Tait computability is a mathematical machine to reduce difficult problems of type theory and programming languages to trivial theorems of topos theory. First employed by Sterling and Harper to reconstruct the theory of program modules and their phase separated parametricity, synthetic Tait computability is deployed here to resolve the last major open question in the syntactic metatheory of cubical type theory: normalization of open terms.

I dedicate this dissertation to my mother LeeAnn, who stood by me in times of great danger and difficulty. I am forever grateful.

ACKNOWLEDGMENTS

When Bob took me under his wing, I was an unlikely candidate to pursue a doctorate in computer science: my background was in linguistics and humanities, and my knowledge of computer science was essentially practical in extent by virtue of my having reluctantly taken a number of engineering jobs to pay off my creditors. I spent many tired evenings in 2013–2014 watching Bob's recorded lectures on Homotopy Type Theory, a restorative practice that followed day after day of debugging broken code that I didn't understand. Bob's lectures convinced me with certainty that I wanted to become involved in this new and exciting field of research, and the opportunity soon presented itself after I attended the Oregon Programming Languages Summer School and met Bob for the first time. Over the past several years, Bob has believed in me, encouraged me, and most importantly, tolerated my (at times) ignorant exuberance and bombast. For that I am deeply grateful.

Lars Birkedal has played a pivotal role in my development as a scientist; I am grateful to Lars for many things, but most of all for teaching me how to question my assumptions and orient long-term theoretical goals around their scientific impact. Lars taught me that only a far-sighted and singular focus on *what will be useful to actual people* can chart the "shining path" through the infinite junkyard of abstractions, tools, and theories.

Many other people have played a role in making my time in Pittsburgh unforgettable. I am sincerely grateful to Steve Awodey for creating an exhilerating and memorable environment in the local HoTT Seminar, welcoming eminent visitor after eminent visitor. The seminar room in Baker Hall is the place where I learned many of the beautiful things that made this dissertation possible. I am especially thankful to Jonas Frey for teaching me category theory, and to Mathieu Anel for opening my mind to the geometric side of topos theory. Let me not forget Deb Cavlovich, who is the unspoken center of gravity in the computer science department; none of us could do our work without her.

I could never have made it through the last five years without my friends and colleagues whom I met in Pittsburgh, a steadily accumulating crew of disrepute — Adrien Guatto, Anders Mörtberg, Bruno Bentzen, Carlo Angiuli, Daniel Gratzer, Evan Cavallo, Favonia, Felix Cherubini, Joe Tassarotti, Jonas Frey, and Mathieu Anel have all influenced my thinking greatly, whether over beers at the Cage, Primanti's sandwiches, or games of pool at the Sports Bar.

Sarah Itzkoff has been my constant friend and confidante nearly since I got here. Carlo

and Daniel have been my loyal collaborators over the past several years without whom it would have been impossible to execute my research program. Bas Spitters originally suggested that I look into Artin gluing, starting me off on a fascinating trajectory that I could not have predicted. In addition to my committee, I owe a deep debt of gratitude to Andrej Bauer, Ulrik Buchholtz, David Thrane Christiansen, Bob Constable, Thierry Coquand, Peter Dybjer, Marcelo Fiore, Alex Kavvos, Dan Licata, Conor McBride, Darin Morrison, Yue Niu, Frank Pfenning, Andy Pitts, Mike Shulman, Léo Stefanesco, Thomas Streicher, Amin Timany, Matt Weaver, and Colin Zwanziger for their encouragement, advice, and inspiration. I am thankful to Daniel Gratzer, the participants of the 2021 Categorical Type Theory seminar at Aarhus, and Ivan Di Liberti for their feedback on a preliminary draft of this dissertation.

In my other life, there are dozens of people whom I should thank but whose names cannot be safely written down. I nonetheless hope this acknowledgment touches you in the way that you have touched me; we will see each other again, because the struggle for emancipation stetches out ahead of us and the trend is pitched toward confrontation.

I thank Pyrmira who has been my cherished partner and companion through the final stages of my degree and beyond. Last and not least, I thank my parents Dan and LeeAnn who have been steadfast in their love and support of me even at times when my life has veered in a frightening direction. I hope that I have made you proud and honored your efforts in raising me.

Jon Sterling
Yachats, Oregon
August 2021

CONTENTS

LIST OF FIGURES

Part I

Dependent Type Theory

CHAPTER 0

CONSPECTUS ON TYPE THEORY

§0.0. INSTRUCTIONS TO THE READER

❦ **(0.0∗1)** *What knowledge do we assume?* We assume knowledge of category theory at an undergraduate level, as might be obtained from an introductory textbook such as that of Awodey [Awo10]. At times we will use category theory of a more advanced nature, but in such cases we attempt to keep our presentation as self-contained as possible; when necessary, we provide suitable references.

⁂ **(0.0∗2)** In order to support greater ease of reference, units of text are numbered with greater granularity than is typical in contemporary mathematics. Consequently we relax the usual delineation of *definition – theorem – proof*, but place certain sigils in the margin to indicate the flavor of a node:

◼ theorems, lemmas, and definitions
⯒ notations and abbreviations
♟ exercises for the reader
◆ examples and illustrations
♒ computations or explicit constructions
⁂ big picture thoughts
☯ bigger picture thoughts
♜ historical remarks and discussion of the literature
♆ warnings of subtle territory

☯ **(0.0∗3)** It is not recommended to read this dissertation in a *linear* style; readers who do so are likely to get stuck. Readers who skim the entire text multiple times through iterative deepening will get the most out of it the most quickly.

◆ **(0.0∗4)** We err on the side of giving many examples rather than presenting the bare minimum needed to substantiate our claims: not every example will be meaningful to every reader. The reader must not despair when they encounter an example they do not understand, but should instead skip over it! A reader who understands every example likely does not have any need to read this dissertation. The purpose of using diverse examples is to disseminate the ideas contained herein to a broad audience with differing backgrounds.

♟ **(0.0∗5)** Exercises are rare, and are intended to serve as "moments of truth": a reader who finds they can't complete an exercise may wish to return to earlier chapters, or consult the suggested references.

☯ **(0.0∗6)** My style of doing mathematics is not to attack an intimidating problem directly, but instead to build up a rather vast library of smaller results at a high level of generality, and retreat into this pleasant "countryside" until the target simply crumbles from exhaustion. This principle is stated most elegantly by the great military theorist Carl von Clausewitz, perhaps the most supple mind that the bourgeoisie produced in their history:

> We have considered the voluntary retreat into the heart of the country as a
> particular indirect form of defence through which it is expected the enemy will
> be destroyed, not so much by the sword as by exhaustion from his own efforts.
> In this case, therefore, a great battle is either not supposed, or it is assumed
> to take place when the enemy's forces are considerably reduced. [CGM11]

The weapons forged while patiently exploring the countryside of type theory can be brought to the battlefield again and again without needing to be re-fashioned, an entirely

new state of affairs that I consider a significant advance in the practice of type theoretic metatheory. A downside of my approach is that one must willingly participate in a "long march" through matters that appear from ground-level to be quite alien to the problem at hand, but which can be seen from a high enough altitude to lead the way to its solution. I hope that the strength of the results contained in this dissertation will purchase from my readers some patience and open-mindedness to the unfamiliar.

§0.1. WHAT DOES TYPE THEORY DO?

(0.1∗1) The main result of this dissertation is to prove a very technical metatheorem for a very technical type theory (cubical type theory). But what is the point? It is worth exploring why type theorists have increasingly gravitated toward cubical type theory in recent years. We start by surveying the main applications of dependent type theory.

§0.1.1. Applications to mathematics

(0.1.1∗1) The earliest awakenings of dependent type theory, *e.g.* those of Bishop [Bis69], de Bruijn [de 70], Martin-Löf [Mar71; Mar75b], and Scott [Sco70], emphasized the role of type theory as an abstract language for doing mathematics. The relationship between type theory and constructivity was historically muddled, due to the interests some of its creators, leading to an understandable if confused skepticism from mathematical quarters.

Today it is better to understand the role of type theory in mathematics in a broader way: type theory is a *synthetic* treatment of the (pre-)mathematical notion of a collection, complementing the historically influential analytic understanding in terms of trees well-ordered by '∈' (*i.e.* sets). Type theory exhibits a form of *axiomatic freedom*, whereby it may be refined in one direction or another to correspond more closely to this or that variation on the notion of collection [AH18]. For instance, one may add rules to type theory that make types behave more like *discrete* collections; in an orthogonal direction, one may enrich type theory with principles that make types behave like *higher-dimensional* collections, as in Homotopy Type Theory (HoTT).

(0.1.1∗2) *What is the actual difference between type theory and set theory?* Type theory is often compared to set theory by noting that the latter is "untyped", and so it is possible to view an entity as an element of $\mathbb{N}$ and as an element of $\mathbb{R}$ simultaneously — whereas type theory forces one to insert a "coercion" from one type to the other. This view of the relationship between set theory and mathematical practice is not borne out in reality. Although one may indeed define the naturals to be the subset of the reals spanned by positive whole numbers, a typical mathematical development will have multiple incompatible encodings in play simultaneously — and it is possible to silently regard a symbol n representing a natural number as an element of *any* of these encodings. Therefore

the untyped nature of set theory does not in fact explain the everyday mathematical
practice of viewing an entity through multiple lenses; on the other hand, the type theoretic
notion of an implicit coercion does explain this practice [Luo97; Rey80].

We have argued that untyped set theory is *used* by mathematicians as if it were typed;
therefore style-of-use cannot be the actual dividing line between type theory and set
theory. It is better to say that set theory is a particular concrete model for the notion of
a collection, whereas type theory is the *language* of collections.[1]

(0.1.1∗3) *Mechanization of mathematics.* Mechanizing mathematics has been an ambition
for many type theoretic implementation projects such as Nuprl, Coq, and Lean. In
terms of scale and depth, the most significant mathematical artifacts in Nuprl include
Bickford's mechanization of the second chapter of *Constructive Analysis* [BB85] and
Bickford's mechanization of basic category theory and presheaf models of (cubical) type
theory [Bic18]. Beginning in the current millennium, Coq has been used to mechanize
results in discrete mathematics and group theory whose magnitude defied existing type
theoretic proof assistants, namely the Four Color Theorem [Gon08] and the Feit–Thompson
Theorem [Gon+13].

Lean has distinguished itself by emphasizing the mechanization of present-day research-
level mathematics, as in the recent mechanization of perfectoid spaces [BCM20] as well as
the mathematically deeper "Liquid Tensor Experiment", pointing the way to a possible
future in which proof assistants may take a leading role in the development of new
mathematics rather than trailing behind as janitors of well-known results.

§0.1.2. Applications to computer science

(0.1.2∗1) According to a certain perspective, a *constructive* theorem is one that is
witnessed by a program that could in principle be executed by a human or a computer.
This view of constructivity has played a deep and pervasive role in the development of
dependent type theory and its applications in the tradition of Per Martin-Löf, especially
in the aftermath of Girard's discovery of an inconsistency in the former's original type
theory. In this latter period, Martin-Löf promoted the view of type theory as a common
language for both computer programming and constructive mathematics [Mar79; Mar84].

Martin-Löf's computational–mathematical perspective was developed further and in
parallel by Constable's team at Cornell University, producing computerized implementa-
tions of type theory such as PL/CV3 [CZ84] and Nuprl [Con+86], unified languages in
which one could develop and execute or extract computer programs, correctness proofs,
and even some general mathematics simultaneously. In Sweden, the ALF and Agda
projects continued to develop Martin-Löf's vision from an intensional point of view [MN94;
Nor09]; a perspective on dependently typed programming based on pattern matching was

---

[1] Of course, one person's model is another person's language; set theory could be the "language of trees".

pioneered by McBride [McB99] and McBride and McKinna [MM04], and operationalized in the Epigram project [Bra+11]. Building on his doctoral work in Epigram [Bra05], Brady developed Idris, a dependently typed strict functional programming language that aims to be used for writing real programs [Bra13; Bra21]; the first significant "real program" written in Idris is Idris itself.

In a parallel but equally fruitful line of work, the French School in their Coq project has transformed the Calculus of Constructions (a dependently typed version of Girard's System F) into a very powerful type theory that increasingly converges with the Swedish picture depicted above [Coq16]. Building on these advances, De Moura and his collaborators have developed a new proof assistant called Lean [DU21; Mou+15], which is simultaneously a language for mechanizing mathematics *and* a general purpose programming language; indeed, much of Lean's current implementation is written in Lean.

§0.1.2.1. Recursion in dependent type theory

(0.1.2.1∗1) One aspect of dependent type theory that has hampered its adoption in computer programming is its emphasis of total functions over partial functions; a notion of partiality that supports general recursion is needed for this application, but the usual semantic accounts of partiality do not immediately play well with the rich logical structure of dependent type theory. This did not stop multiple investigations into partiality in type theory from going forward, which we survey in **(0.1.2.1∗2)**, **(0.1.2.1∗3)**, **(0.1.2.1∗6)** and **(0.1.2.1∗8)** through **(0.1.2.1∗9)** in conceptual rather than historical order.

(0.1.2.1∗2) *Synthetic domain theory.* Weary of the increasingly esoteric character of classical domain theory and the search for well-behaved categories of domains, Dana Scott proposed in 1980 that one should search for a "category of sets" that contains a good category of domains as a full subcategory. The advantage of such a *synthetic* treatment of domain theory is that the notion of a continuous map could be dispensed with, as things would be arranged to ensure that *any* function between these non-standard sets is continuous.

Scott's pronouncement sparked an entirely new field of research called *synthetic domain theory* or SDT [FR97; Fre+92; Hyl91; Pho91; Reu95; Reu96; Reu99; RS93; RS99; Ros86; Tay91];[2] a somewhat different approach to synthetic domain theory was carried out in isolation from this tradition by the Nuprl team, discussed below in **(0.1.2.1∗3)** through **(0.1.2.1∗5)**. The high points of SDT included the fact that any model of SDT contains a full subcategory of predomains that is cartesian closed, complete, cocomplete, and closed under powerdomains — meaning that it supports everything needed for doing workaday denotational semantics. This state of affairs led Taylor to make his famous pronouncement:

---

[2] This dissertation is named *en hommage* to the famous paper of Hyland, *First steps in synthetic domain theory* [Hyl91].

> *My personal view is that the study of domain theory by bit-picking should be brought to a close.* [Tay91]

While ideas from SDT remain important and continue to influence present-day work, *e.g.* that of Matache, Moss, and Staton [MMS21], it seems that SDT as a field has collapsed under its own weight with a number of important questions remaining open and blocking further progress, as noted by Bauer, Birkedal, and Scott [BBS04]. In spite of this, the program of SDT can still be regarded a success in at least one respect: it provides the quickest way to get a well-behaved category of domains.

(0.1.2.1∗3) *Nuprl's synthetic domain theory.* Nuprl's computational type theory internalizes many properties of untyped computation and realizability; as such, it is already possible in the basic version of Nuprl to define total functions using unrestricted fixed points (*e.g.* those formed using the Y combinator) by first constructing a general recursive realizer and then proving by induction that it is well-typed and terminating. Constable and Smith [CS87] investigated the extension of Nuprl's computational type theory with *partial types* $\bar{A}$ (sometimes called a "bar type") that provide a codomain for genuinely partial functions. To prove properties of partial functions defined by general recursion, a fixed point induction principle is provided. As an implementation of Scott's dictum that "domains are sets", Nuprl's treatment of partiality can be appropriately referred to as a form of synthetic domain theory, albeit one very different from the theories usually considered under that name **(0.1.2.1∗2)**.

(0.1.2.1∗4) *Subtleties of admissible types in Nuprl.* In domain theory, a partial order A supports fixed points of Scott-continuous functions $A \to A$ when it has a bottom element and is closed under directed suprema (or colimits of ω-chains); an object satisfying only the latter condition might be called a *predomain*, whereas an object satisfying both conditions can be called a *domain*. A Nuprl type is called *admissible* when it satisfies a property analogous to being closed under colimits of ω-chains. In Nuprl, every bar type $\bar{A}$ contains the non-terminating computation, but $\bar{A}$ need not be admissible unless A is admissible. This is different from the state of affairs in conventional domain theory, where the *lifted* type $A_\perp$ freely adds not only a bottom element but also closure under directed suprema.

(0.1.2.1∗5) *Dependent types in synthetic domain theories.* As in ordinary synthetic domain theory, Nuprl's admissible types are closed under a number of type constructors — notably including dependent products $(x : A) \to B(x)$, and *not* including dependent sums $(x : A) \times B(x)$. In essence this is because one must restrict families of types to be suitably continuous in their variation over the base, a proof-relevant version of the admissibility condition for subsets of domains in classical domain theory. One way to understand this phenomenon is to observe that the property of being an admissible type does not take dependency into account and is therefore unsuitable in a dependently typed scenario.

A notable contribution of Crary [Cra98] was to generalize Nuprl's admissibity property

to what he called *predicate-admissibility*, a notion that is closed under dependent sums. From a more abstract perspective, Jacobs [Jac99] observes that it is possible to model dependent sums and products in the fibration $\mathbf{CFam}(\mathbf{Dcpo}) \longrightarrow \mathbf{Dcpo}$ of *continuous* families of dcpos, *i.e.* functors $D \to \mathbf{Dcpo}^{\mathbf{EP}}$ out of a dcpo D that take directed suprema to colimits. It would serve our field to investigate the relationship between these two approaches.

(0.1.2.1∗6) *Synthetic guarded domain theory.* In the first decades of the millennium, new life was breathed into the search for a synthetic domain theory by the emergent needs of computer scientists investigating the meaning of higher-order store, *i.e.* storage of functions that may themselves access the store [Ahm04; Bir+11b; BST10]. The essence of these new techniques, sometimes referred to as "step-indexing", is to solve recursive equations in an *approximate* way that guards each unfolding step with an abstract "breadcrumb". Birkedal, Møgelberg, Schwinghammer, and Støvring [Bir+11a] noted that any locale whose frame of opens has a well-founded basis can serve as a model for a *synthetic **guarded** domain theory* (SGDT) that abstracts this point of view.

Most subsequent work has emphasized the role of SGDT as a metalanguage for operational models of programming languages [BB18; Jun+18; Jun+15; Spi+21], but some authors have continued to raise the banner of denotational semantics in the setting of guarded domains [MP16; MV19; MV21; Pav16; PMB15]. The type theoretic aspects of SGDT have been explored by a number of authors under the banner of *guarded dependent type theory* [Bir+11a; Biz16; BBM14; Biz+16; BM15; BM20; Clo+15; SH18]; recent work suggests that a combination of guarded type theory with cubical type theory may prove fruitful [Bir+16; VV20], and the results of the present dissertation are likely to be important for establishing the metatheoretic properties of such a combination.

Unlike ordinary synthetic domain theory, which supports computation by extraction through realizability models as in the work of Hyland [Hyl91] and Phoa [Pho91], it appears that the type theories based on SGDT can be computed with directly.

(0.1.2.1∗7) *SGDT as a theory of domains.* We suggest a few high-level reasons why guarded domains should be considered mathematically interesting and useful (besides the fact that they make certain constructions possible). SGDT goes further in one aspect toward realizing Scott's dictum ("domains are sets") than prior attempts did. Although SDT determines various non-equivalent *full subcategories* that may be thought of as predomains (*e.g.* complete Σ-spaces, replete types, well-complete types, *etc.*), the better-behaved notions among these are notoriously difficult to characterize concretely. As Phoa [Pho91] pointed out, it seemed that none of these was likely to be the final story for predomains.

In contrast, *every* type A in SGDT may be seen as a *guarded predomain*, supporting guarded fixed points of functions $\blacktriangleright A \to A$, where $\blacktriangleright$ is the *later modality* — the type-level "breadcrumb" mentioned in **(0.1.2.1∗6)**. This is already strikingly better-behaved than either the classic domain theory or the synthetic domain theory, whether in Scott's tradition **(0.1.2.1∗2)** or that of Nuprl **(0.1.2.1∗3)**: the guarded predomains are closed

under dependent sums and even extensional equality types!

🌿 **(0.1.2.1∗8)** *Mathematics of infinity.* Per Martin-Löf's "mathematics of infinity" [Mar90; Pal93] aims to account for infinite recursion by extending type theory with a hierarchy of fixed point combinators for types equipped with a chosen element:

$$\frac{a : \mathrm{A} \qquad x : \mathrm{A} \vdash f(x) : \mathrm{A}}{\mathsf{fix}_i(a,f) : \mathrm{A} \qquad \mathsf{fix}_i(a,f) \equiv f(\mathsf{fix}_{i+1}(a,f)) : \mathrm{A}} \ (i \in \mathbb{N})$$

The purpose of the indexing is to avoid inconsistency; Martin-Löf [Mar90] notes that having only one such fixed point combinator $\mathsf{fix}(a,f)$ satisfying $\mathsf{fix}(a,f) \equiv f(\mathsf{fix}(a,f))$ is inconsistent, even though one avoids the obvious inconsistency of having recursion in an uninhabited type. Unfortunately, Martin-Löf's account of the meaning of infinite recursion defies the important *canonicity* property, which essentially states that in the absense of variables a definable element of the sum $\mathrm{A} + \mathrm{B}$ must arise from either a definable element of A or a definable element of B. It was therefore unclear in what sense Martin-Löf's extended type theory could serve as a programming language, for which the canonicity property is usually indispensible.

🌿 **(0.1.2.1∗9)** *General recursion in total type theory.* It is well-known that general recursive functions can be encoded in a total setting by means of their *accessibility predicates*, indexed inductive structures that consolidate all the recursive calls that are made at given input [Nor88]. Bove and Capretta [BC05] refine and operationalize this perspective as a very simple recipe to encode a general recursive function in total type theory, which we illustrate on the quick-sort algorithm:[3]

1) Define an inductive predicate $\mathcal{A}_{\mathsf{qs}} : \mathsf{List} \to \mathcal{U}$ that sends every possible list to the type of recursive calls that it induces:

$$\frac{}{\mathcal{A}_{\mathsf{qs}}(\mathsf{nil})} \qquad \frac{\mathcal{A}_{\mathsf{qs}}(\mathsf{filter}((< x),xs)) \qquad \mathcal{A}_{\mathsf{qs}}(\mathsf{filter}((\geq x),xs))}{\mathcal{A}_{\mathsf{qs}}(x :: xs)}$$

2) Define the quick-sort algorithm $\mathsf{qs}' : (xs : \mathsf{List}) \to \mathcal{A}_{\mathsf{qs}}(xs) \to \mathsf{List}$ by structural recursion on the second argument.
3) Exhibit a section of $\mathcal{A}_{\mathsf{qs}}$, *i.e.* a function $(xs : \mathsf{List}) \to \mathcal{A}_{\mathsf{qs}}(xs)$.
4) From the above, obtain the quick-sort function $\mathsf{qs} : \mathsf{List} \to \mathsf{List}$.

One spurious objection to the Bove–Capretta account of general recursion is that the resulting code of qs will execute very inefficiently in two steps: first it will construct an element of $\mathcal{A}_{\mathsf{qs}}$ and then it will deconstruct it. As results of Blum [Blu67] imply, the cost of executing this first stage can be *arbitrarily* bad in comparison to the actual general

---

[3] Christiansen and Brady [CB16] employ Idris's elaborator reflection to define a routine that executes the Bove–Capretta translation automatically.

recursive algorithm we wished to encode. But accessibility proofs and their computations can always be systematically erased from extracted code leaving only the underlying general recursive algorithm, as pointed out by Brady, McBride, and McKinna [BMM03].[4]

Objections pertaining to expressivity are also unsuccessful: Bove and Capretta [BC05] prove that their encoding covers *all* partial recursive functions in the sense of recursion theory. Of course there is a practical matter pertaining to the plumbing and maintenance of accessibility proofs in this encoding, but McBride [McB12] points out that all of these matters can be conveniently bundled into an *interaction tree* or *resumption* monad corresponding to a given dependent partial function space. Such a representation, however, does raise the question of bisimilarity and bisimulation; it may be that the work of either Altenkirch, Danielsson, and Kraus [ADK17] can be adapted to provide a satisfactory notion of program equivalence in such a setting.

(0.1.2.1∗10) *General recursive cost analysis in total type theory.* Just as the lack of actual general recursion in dependent type theory does not obstruct general recursive programming **(0.1.2.1∗9)**, it likewise does not inhibit the analysis of such programs. In an extreme example, Niu, Sterling, Grodin, and Harper [Niu+21] demonstrate how to program and prove cost bounds for general recursive algorithms in a total type theory called **calf** (the **c**ost-**a**ware **l**ogical **f**ramework) in the absence of a general fixed point combinator, employing a refinement of the method of Bove and Capretta to define general recursive algorithms by recursion on their own call graphs. The resulting complexity analysis pertains *not* to the cost of naïvely executing the termination proofs but rather to the cost of the intended general recursive algorithm.

(0.1.2.1∗11) Based on the arguments of **(0.1.2.1∗9)** and **(0.1.2.1∗10)**, one is forced to conclude that as far as *recursive programming* is concerned, there is no factual need for a separate account of partiality featuring a fixed point combinator. Of course, the other main use of recursion and partiality is to solve domain equations in order to define operational and denotational models of programming languages; this need is *not* addressed by the methods of Bove and Capretta [BC05]. Therefore in spite of the results of *op. cit.*, the struggle for a general purpose synthetic domain theory continues.

§0.1.2.2. Logical frameworks: syntactic and semantic

(0.1.2.2∗1) Historically the *meta*-aspects of logic were understood by encoding logic into something other than logic, *e.g.* the encoding of logic as strings of symbols.[5] The big surprise for many scientific workers in the latter half of the 20th century was that the meta-aspects of logic are themselves essentially logical: logics can be studied by immersion into bigger logics, and this method is robust enough to encompass all non-junk theorems.

---

[4] See also Brady [Bra05, Chapter 4] for further discussion.   [5] A regrettable convention that appears still even in the most progressive and up-to-date textbooks, complete with "parenthesis-counting" junk-theorems.

Because the syntax and binding structure of dependent type theory is itself somewhat complex, type theorists very early on abstracted it by working internally to an ambient type theory of some kind — for instance, Martin-Löf's introduction of the *system of arities* to abstract away variable binding and substitution appeared already in his 1980 lectures in Munich, less than three months after his Padova lectures that omitted them [Mar84, Preface]. In even earlier work, Aczel [Acz78] had introduced a lightweight framework second-order abstract syntax that would inspire and inform many subsequent explanations *and* concrete implementations of syntax, *e.g.* those of Fiore and Mahmoud [FM10], Fiore, Plotkin, and Turi [FPT99], and Harper [Har16]. A type theory that is used as a metalanguage for specifying another type theory is often called a *logical framework.*

(0.1.2.2∗2) *Syntactic logical frameworks.* By the end of the 1980s, two trends in logical frameworks had emerged. On the one hand, there were the *syntactic logical frameworks* inspired by Martin-Löf's system of arities, most notably the Edinburgh Logical Framework ("LF" for short) of Harper, Honsell, and Plotkin [HHP93], which has been used most notably to provide the first fully rigorous and mechanized definition of a general purpose higher-order programming language [CH09; LCH07]. Syntactic logical frameworks aim to capture the scoping and well-sortedness of object-language expressions, judgments, and derivations. In these frameworks, the LF-level function space captures both binding of variables in expressions and binding of hypotheses in derivations.

(0.1.2.2∗3) *Adequacy in syntactic logical frameworks.* A presentation in a syntactic LF formalizes not a language but a particular formalism presenting that language. For instance, consider the LF encoding of a language that has types, terms, and judgments $\boxed{\Gamma \vdash M : A}$ and $\boxed{\Gamma \vdash M \equiv N : A}$:

$$
\begin{aligned}
\mathsf{tp}, \mathsf{tm} &: \square \\
\mathsf{isOf} &: \mathsf{tm} \to \mathsf{tp} \to \square \\
\mathsf{isEq} &: \mathsf{tm} \to \mathsf{tm} \to \mathsf{tp} \to \square \\
\mathsf{isEq/refl} &: (A : \mathsf{tp}, M : \mathsf{tm}) \to \mathsf{isOf}(M,A) \to \mathsf{isEq}(M,M,A) \\
\mathsf{isEq/symm} &: (A : \mathsf{tp}, M : \mathsf{tm}, N : \mathsf{tm}) \to \mathsf{isOf}(M,A) \to \mathsf{isOf}(N,A) \to \mathsf{isEq}(M,N,A) \to \mathsf{isEq}(N,M,A) \\
\mathsf{isEq/trans} &: (A : \mathsf{tp}, L : \mathsf{tm}, M : \mathsf{tm}, N : \mathsf{tm}) \to \mathsf{isOf}(L,A) \to \mathsf{isOf}(M,A) \to \mathsf{isOf}(N,A) \\
&\quad \to \mathsf{isEq}(L,M,A) \\
&\quad \to \mathsf{isEq}(M,N,A) \\
&\quad \to \mathsf{isEq}(L,N,A)
\end{aligned}
$$

One must add dozens of constants to the **isEq** judgment to ensure that it is a congruence for all the constructs of the encoded language. A "model" of this theory would consist of a set of raw types, a set of raw terms, an indexed set of typing derivations, and an indexed set of equality derivations. In contrast, an actual model of the language being presented would be nothing more than a set of types ⟦**tp**⟧ and a family of sets of terms

$[\![tm]\!]$ indexed in $[\![tp]\!]$. In this sense, a syntactic logical framework is meant to be used to study the properties of specific formalisms presenting a language, not the language itself.

Because the LF presentation of judgments and derivations differs significantly from what it is meant to encode (the latter involving contexts and variables and turnstiles), it is necessary to prove an *adequacy* metatheorem for every LF presentation: in a specified class of LF contexts, equivalence classes of LF terms must be in *bijection* with derivations from the original formalism. These theorems are tractable because syntactic LFs are designed to support a *canonical forms* or *normalization* property: there is a presentation of the LF in which every equivalence class of typed terms is a singleton [HP05; Wat+04].

Both the distance from models described above and the necessity to prove sometimes difficult adequacy theorems seems to have slowed the adoption of syntactic LFs in spite of their proven utility. Both of these downsides are addressed in the trend of *semantic logical frameworks*, which evince an immediate connection to the intended notion of model and therefore trivialize the question of adequacy in the commonly encountered cases.

♣ **(0.1.2.2∗4)** *Semantic logical frameworks.* An alternative to syntactic logical frameworks are what Harper [Har21] refers to as *semantic logical frameworks*. Semantic LFs differ from their syntactical counterparts in only one technical respect: a presentation is allowed to equational laws in addition to constants. Users of semantic LFs then employ these equational laws to define presentations of languages themselves, rather than presentations of formalisms that themselves present languages. The most famous semantic LF was that of Martin-Löf [NPS90]. The precise understanding of Martin-Löf's framework, however, remained too informal and under-specified for its employment in sensitive metatheoretic work, delaying the advent of semantic logical frameworks (*e.g.* those of Altenkirch and Kaposi [AK16b], Cartmell [Car86], Gratzer and Sterling [GS20], Harper [Har21], and Uemura [Uem19; Uem21]) as a serious competitor to conventional techniques.

◆ **(0.1.2.2∗5)** Returning to our example of a language that has both types and terms, the corresponding semantic presentation begin with only two constants, with no need to specify any rules for equality whatsoever:

$$\mathsf{tp} : \square$$
$$\mathsf{tm} : \mathsf{tp} \to \square$$

One could add, for instance, a type of booleans with the appropriate equational laws:

$$\mathsf{bool} : \mathsf{tp}$$
$$\mathsf{tt}, \mathsf{ff} : \mathsf{tm}(\mathsf{bool})$$
$$\mathsf{if} : (A : \mathsf{tp}) \to \mathsf{tm}(\mathsf{bool}) \to \mathsf{tm}(A) \to \mathsf{tm}(A) \to \mathsf{tm}(A)$$
$$\mathsf{if}/\mathsf{tt} : (A : \mathsf{tp}, t : \mathsf{tm}(A), f : \mathsf{tm}(A)) \to \mathsf{if}(A,\mathsf{tt},t,f) \equiv_{\mathsf{tm}(A)} t$$
$$\mathsf{if}/\mathsf{ff} : (A : \mathsf{tp}, t : \mathsf{tm}(A), f : \mathsf{tm}(A)) \to \mathsf{if}(A,\mathsf{ff},t,f) \equiv_{\mathsf{tm}(A)} f$$

$$\mathsf{if/uniq} : (A : \mathsf{tp}, a : \mathsf{tm(bool)} \to \mathsf{tm}(A), b : \mathsf{tm(bool)}) \to a(b) \equiv_{\mathsf{tm}(A)} \mathsf{if}(A, b, a(\mathsf{tt}), a(\mathsf{ff}))$$

※ **(0.1.2.2∗6)** *Models of an LF presentation.* A model of the language presented in **(0.1.2.2∗5)** consists of a set of types, a set of terms indexed in the set of types, closed under a type that behaves like the booleans. This is nothing more than an informal rephrasing of the formal specification of **(0.1.2.2∗5)**, however. The close connection between semantic LF presentations and their models enables us to "cut the link" that bound us previously to conventional formal presentations and obligated us to prove onerous adequacy results. Indeed, any conventional formal presentation that is not equivalent to our LF presentation must surely be semantically wrong! What is new in the setting of semantic LFs is that the LF presentation is sufficiently authoritative that it is finally up to purveyors of conventional formalisms to establish equivalence (soundness and completeness) relative to the LF presentation rather than the other way around.

※ **(0.1.2.2∗7)** *Syntactic adequacy for semantic LFs.* Some semantic logical frameworks, such as that of Uemura [Uem19; Uem21], maintain a very close connection to the traditional syntax of type theory by distinguishing between contexts and judgments and restricting to the "level" of binders accordingly. For such a framework, the syntactic adequacy statement is evident in the authoritative sense described above **(0.1.2.2∗6)** — any conventional formalism that deviates from the LF presentation in Uemura's framework is simply wrong.

Gratzer and Sterling [GS20] have argued that it is significantly less technical to use a semantic logical framework that does not distinguish between contexts and judgments and places no restriction whatsoever on binders (after both the Edinburgh LF and Martin-Löf's LF); for such an encoding, however, syntactic adequacy becomes again a non-trivial question — *a priori* it may have been the case that allowing binders of higher level makes it possible to derive a strictly new judgment that doesn't mention binders of higher level. To demonstrate that this perverse scenario does not obtain, *op. cit.* proved a syntactic adequacy theorem relative to Uemura's framework using semantic methods. Adequacy theorems for semantic LFs must be carried out using different methods from those that apply to syntactic LFs **(0.1.2.2∗3)**, because there is no hope of normalization or canonical forms in the presence of arbitrary equational hypotheses; the adequacy result of *op. cit.* is, however, a very simple Artin gluing argument *à la* Lafont [Laf88].

※ **(0.1.2.2∗8)** *The future of logical frameworks.* It appears that semantic LFs address the underlying reasons for the community's hesitancy to choose logical frameworks over hand-crafting formalisms. In addition to adequacy, which is a non-trivial obligation for any real language, there is the fact that syntactic LFs deal mainly with well-sortedness and well-scopedness, things that many practitioners in programming languages (rightly or wrongly) were not particularly worried about in the first place. Semantic LFs go beyond this and address aspects of programming language design that most practitioners would agree are very difficult to get right: equational reasoning and the relationship to models,

e.g. soundness, completeness, *etc.*

Not all is rosy — the semantic LFs currently lack a viable implementation strategy. Syntactic LFs can can be implemented in easy-to-use proof assistants such as Twelf, Beluga, and Abella [Gac08; PS99; PD10], a possibility that boils down to the fact that they enjoy normalization and therefore decidable conversion and type checking. For this reason the present generation of semantic LFs is mainly used for pencil-and-paper proofs, with some notable exceptions [AK16a; AK16b].

§0.2. WHAT DOES TYPE THEORY NEED?

※ **(0.2∗1)** In order to perform adequately for its intended applications, many requirements are placed on type theories. We conveniently divide these into *semantic* and *syntactic* properties, defined below **(0.2∗2)**.

◼ **(0.2∗2)** A *semantic* property is one that is expected to hold of *all* models of a given type theory. A *syntactic* property is one that is expected to hold specifically of the syntactic or term model of a type theory, rather than all models. While a true semantic property is rightfully referred to as a *theorem*, it is common to refer to true syntactic properties as *metatheorems*. The study of syntactic properties is called *metatheory*.

♆ **(0.2∗3)** Most syntactic properties in the sense of **(0.2∗2)** pertain to semantics, *e.g.* **(0.2.2∗3)** speaks of the syntax having a certain kind of model, which is not a statement about all models.

※ **(0.2∗4)** Structural proof theorists may appreciate the following analogy: semantic properties are similar to derivabilities, whereas syntactic properties are similar to admissibilities.

§0.2.1. Semantic properties (theorems)

◆ **(0.2.1∗1)** *Function extensionality*. Most type theories have an *equality type connective* $\mathsf{Eq}_A(M,N)$ that internalizes the judgment that M and N can be identified. In order for this connective to do its job, it needs to correctly characterize equality for each type A; in particular, the map from the equality type $\mathsf{Eq}_{A \to B}(f,g)$ governing equality of functions to the dependent product type $(x : A) \to \mathsf{Eq}_B(f(x),g(x))$ governing *pointwise equality* of functions must have an inverse, called *function extensionality*. This property is necessary for most applications of dependent type theory, but it fails for many variants of type theory that are implemented in proof assistants today, such as Agda, Coq, and Idris.

The reason for the frequent lack of function extensionality is that it has been historically difficult to combine function extensionality, closed term computation **(0.2.2∗4)**, and open term computation **(0.2.2∗5)**. For instance, Coq and Agda have both closed and open term computation but lack function extensionality; on the other hand, Lean has function extensionality but lacks both closed and open term computation; Nuprl has

function extensionality and closed computation, but lacks open computation. Function extensionality, however, is non-optional for both mathematics and program verification tasks; consequently, practitioners tend to either avoid built-in equality types or add axioms that destroy the computational properties of type theory.

◇ **(0.2.1∗2)** *Function comprehension.* In ordinary mathematics, every total relation $R \subseteq A \times B$ is the graph of a unique function $f : A \to B$. It is sometimes said that this property, which we call *function comprehension* following the terminology of Shulman,[6] holds in classical mathematics but not in constructive mathematics, but these claims are mistaken: it is absolutely the case that in mainstream constructive mathematics (*e.g.* that of Bishop [BB85], or the mathematics of an elementary topos [LS86]), total relations determine functions. Unfortunately, the status of function comprehension is somewhat murky in traditional type theory.

The difficulty is in defining what we mean by a "total relation", which boils down to the more basic question of what a "proposition" is (since a relation is a propositional function). There is only one mathematically correct notion of proposition for type theory:

> *A proposition is a type* A *that is equipped with an element of the dependent product type* $(x,y : A) \to \mathsf{Eq}_A(x,y)$, i.e. *a type that has at most one element.*

If "total" is then taken in the corresponding sense, then type theory has function comprehension. Unfortunately, this is *not* the notion of proposition employed in most versions of dependent type theory; for instance, Coq has a distinct universe Prop of propositions that is (with small exceptions) quite sequestered from the language of types, and as a result it is easy to find total relations that do not determine definable functions in Coq. The simplest example is the generic one: the obvious total relation $R \subseteq \{n : \mathbb{N} \to \mathsf{Prop} \mid \mathsf{isSingleton}(n)\} \times \mathbb{N}$ that identifies a singleton subset of the natural numbers with the natural number that it contains does not correspond to a definable function. Another way to view the failure of function comprehension is as follows: it is not the case that all simultaneously injective and surjective functions have inverses in Coq.

Unfortunately, function comprehension is needed in order to get almost any mathematical formalization off the ground. The failure of function comprehension in Coq is often dealt with in two ways: by adding axioms that disrupt Coq's computational properties, or by restricting to types that are sufficiently finitary and discrete as to be *projective*, validating the much stronger axiom of choice; the latter approach is taken by the developers of the MATHEMATICAL COMPONENTS formal library in Coq, which has notably been used to mechanize the Four Color Theorem and the Odd Order Theorem [Gon08; Gon+13]. Working with projective types is possible in some areas of mathematics, but it is of course not a possibility for developing general mathematics constructively; the

---

[6] It is more traditionally referred to as *unique choice* or *definite description.*

failure of function comprehension *vis-à-vis* Prop has predictably slowed the development of constructive mathematics in systems like Coq.

◆ **(0.2.1∗3)** *Propositional univalence.* In mathematics, two propositions are the same when they imply each other; we refer to this property of propositions, traditionally called *propositional extensionality*, as "propositional univalence" after Voevodsky. Propositional univalence has a number of important consequences, including the effectivity of quotients which we discuss below **(0.2.1∗4)**. While propositional univalence is not necessarily needed for the use of type theory as a programming language, it is non-negotiable for most mathematical applications. Echoing the theme of **(0.2.1∗2)**, it is often possible to get around the failure of propositional univalence in conventional type systems (such as Martin-Löf Type Theory and the Calculus of Inductive Constructions) when formalizing sufficiently finitary mathematics, *e.g.* by using *boolean reflection*.

◆ **(0.2.1∗4)** *Effective quotients.* If A is a set and R is an equivalence relation on A, then the quotient $A \twoheadrightarrow A/R$ is the "universal solution" to identifying elements related by R. In other words, a function $[-]_R : A/R \longrightarrow B$ is exactly the same as a function $A \longrightarrow B$ that sends any *a,b* such that *a* R *b* to equal elements of B. This universal property, however, is not sufficient for most use-cases of quotients in mathematics; an additional structural principle is needed for quotients to be actually useful:

  Effectivity. For any *a,b* : A, if $[a]_R = [b]_R$ in A/R then *a* R *b*.

Effectivity follows from propositional univalence. Why do we need it? A useful example is provided by Cavallo [Cav21]. We might consider quotienting the type $\mathbb{Z}^{\mathbb{Z}}$ of sequences by the following equivalence relation that expresses the "eventual equality" of two sequences:

$$\alpha \sim \beta \Leftrightarrow \exists n.\forall m > n.\alpha(m) = \beta(m)$$

When one does not have effectivity of quotients, one cannot conclude from the fact two sequences α,β are identified in $\mathbb{Z}^{\mathbb{Z}}/\sim$ that there is in fact a prefix after which α,β behave the same! A more sophisticated version of this scenario ensures that even basic analysis cannot be developed in a constructive setting without effective quotients.

It is well-known that not all quotients are effective in popular computability-inspired models such as partial equivalence relations (PERs); Nuprl was one of the first type theoretic proof assistants to introduce quotients, but the language of Nuprl has always been moored to its intended model in PERs. Consequently the quotients in Nuprl cannot be used for mathematical applications, a difficulty that inspired a number of attempts at mitigation [Nog02]; it seems that the ultimate mitigation has been to avoid Nuprl's built-in quotients altogether, as in Bickford's formalization [Bic16; Bic] of the second chapter of *Constructive Analysis* [BB85] as well as the Nuprl formalization of Euclid's straitedge and compass geometry [KBC19]. A similar phenomenon is noted in the Nuprl formalization of some constructive algebra [Gro]. In other words, Nuprl users (informally) use the setoid encoding of quotients in order to develop constructive mathematics.

From the point of view of mathematics, it is not particularly surprising that setoids are required in Nuprl to develop mathematics that uses quotients; setoids are an example of an *exact completion*, a universal way to add well-behaved quotients to a category. A different exact completion is used in *categorical realizability* to build a usable universe of mathematics on top of PERs, by embedding PERs (with poor closure properties) into a universe of non-PERs (with good closure properties). The fully-faithful nature of this embedding ensures that any result in the extended universe that mentions only PERs is actually true of PERs.

✳ **(0.2.1∗5)** *Quotients are not useless.* Bishop contends [Bis67, p. 60] that defining the reals by *quotienting* their presentations (regular sequences of rationals) is either pointless or incorrect — pointless because one could simply carry around proofs that various equivalence relations are preserved, or incorrect because it may obstruct the definition of certain functions. Of course, there is no quotient whatsoever that would fail to be "pointless" in Bishop's sense and yet most practicing mathematicians would be surprised to find out that quotients are not useful.

Indeed, following Bishop's logic there would be no place in linear algebra for the tensor product of vector spaces — since one may of course just prove that some map is bilinear. Likewise, not even the cartesian product of sets escapes what we might refer to as "Bishop-pointlessness", considering that one can simply speak of functions that take multiple arguments. It does not take much investigation to see that this is not a perspective of much merit within the world of modern-day mathematics, constructive or not. Kellison, Bickford, and Constable [KBC19] argue for Bishop's case, observing that their Nuprl proof of the *magnifying glass postulate* (co-transitivity of apartness) is validated for presentations of real numbers but not for their equivalence classes. Far from taking this as evidence that the reals should not be quotiented, it is better to view it as evidence that the magnifying glass postulate is *not* a statement about the real numbers.

Our view is that it is correct to take a quotient when it (1) assists the encoding of geometrical intuitions in mathematical language, and (2) satisfies the effectivity property **(0.2.1∗4)**, and incorrect otherwise; because many quotients fail to be effective in Nuprl, they are best avoided altogether in that setting. Given his comments, we note with irony that Bishop's setoid-based foundation of mathematics is one in which all quotients *are* effective, and hence there is no rational reason to avoid them.

⚘ **(0.2.1∗6)** The Computational Higher Type Theory/CHTT program of Angiuli, Hou (Favonia), and Harper [AHH17] can be seen to be *another* way to correct the inapplicability of PERs to mathematics by replacing them with *cubical* PERs, a generalization that in some sense "builds in" an iterated version of setoids. Indeed, CHTT is compatible with effective quotients (and much more) despite being based on PERs [Cav21; CH18; CH19].

◆ **(0.2.1∗7)** *General univalence.* Propositional univalence generates many of the important

structural properties of sets in mathematics (both constructive and classical) — for instance, propositional univalence ensures that quotients by equivalence relations behave correctly **(0.2.1∗4)**. Propositional univalence can be generalized from propositions-and-biimplications to sets-and-bijections, as in the ground-breaking paper of Hofmann and Streicher [HS98]; this "set-level univalence" then generates important structural properties of not only sets, but also groupoids. The piercing insight of Voevodsky [Voe06] was to observe that propositions, sets, and groupoids are just the first few levels of an infinite hierarchy of "homotopy levels"; the appropriate generalization of propositional and set-level univalence to the entire hierarchy, then, can generate the important structural properties of ∞-groupoids or *homotopy types* at any dimension. From this perspective, univalence — the fact that equivalences between types are the same as equations between them — is the ultimate extensionality principle to civilize the mathematical universe.

Different levels of univalence are needed for different applications. We pointed out that propositional univalence is needed to get almost any "ordinary" mathematics off the ground, but we argue that in the future we should seek to develop type theories that are compatible with general univalence. Just as propositional univalence ensures that quotients by equivalence relations are well-behaved, general univalence ensures an analogous property for *arbitary colimits*, resolving many anomalies in low-dimensional mathematics.

§0.2.2. Syntactic properties (metatheorems)

※ **(0.2.2∗1)** In §0.2.1 we have discussed a number of *semantic* properties that experience has shown to be important for type theories and their applications. There is a complementary range of important *syntactic* properties that one must also satisfy that we discuss in **(0.2.2∗2)** through **(0.2.2∗5)**; we recall from **(0.2∗2)** that a syntactic property is one that is specific to the syntactic model rather than one that pertains to all models of the theory.

◆ **(0.2.2∗2)** *Consistency.* The simplest and most important syntactic property is *consistency*, *i.e.* that there exists an empty type, *i.e.* a type E such that there does not exist a closed term $\cdot \vdash M : E$. The consistency property is often taken to mean that a theory cannot prove things that are false; that is not entirely correct, because "things that are false" presupposes a particular intended semantics. It is better to say that a consistent theory is one that is *compatible* with a semantics that can tell the difference between true and false. There are many ways to prove consistency for a theory; the most direct is to exhibit an "intended model" **(0.2.2∗3)**, but consistency also often follows (in a much more complex way) from purely syntactical/computational considerations such as **(0.2.2∗4)** and **(0.2.2∗5)**.

◆ **(0.2.2∗3)** *Intended models.* Type theories are not used for their own sake; one uses type theory as an *abstraction* of some actual domain of interest. For instance, we may use type theory as a language for speaking about sets, or spaces, or automata, *etc.*; to substantiate

these applications, it is necessary to ensure that the domain of interest is actually closed under type theoretic operations (*e.g.* function, products, *etc.*). To do so is the same as constructing a model of type theory using the intended domain as raw material; then, a theorem stated in terms of types can be read as a statement about (*e.g.*) spaces, and a simple type theoretic proof can be unraveled to a much more complex proof about spaces. We break with tradition in the type theoretic community (*e.g.* the philosophical work of Martin-Löf [Mar84; Mar96]) by not accepting the idea of a single "intended model"; instead we think of the intended model as being conditional on the desired applications.

◆ **(0.2.2∗4)** *Closed term computation.* Martin-Löf's profound identification of mathematics with computer programming [Mar79] expresses a hypothesis that terms in type theory can be run like programs **(0.1.2∗1)**. In particular, if you have a closed term $\cdot \vdash M : \mathsf{nat}$, then there is an actual natural number $m \in \mathbb{N}$ such that $\cdot \vdash M \equiv \bar{m} : \mathsf{nat}$ is derivable; moreover, we should have some effective way to compute m. The existence of the number $m \in \mathbb{N}$ is often referred to as *canonicity*; if a particular procedure is chosen to exhibit such an m, then the canonicity result can be seen as a form of *computational adequacy* in the sense of Plotkin for the term model of type theory relative to this procedure. When a canonicity proof is given in a constructive metatheory, it automatically contains within it an algorithm; canonicity proofs in non-constructive metatheories may yet evince algorithms for recursion-theoretic reasons.

Not all type theories support closed term computation; in the past it has been necessary in many cases to sacrifice computation of closed terms in order to support various semantic properties, such as function comprehension **(0.2.1∗2)** and propositional univalence **(0.2.1∗3)**. One of the benefits of cubical type theory is that these semantic properties can be obtained without sacrificing closed term computation, a result proved independently by Angiuli, Hou (Favonia), and Harper [AHH18] and Huber [Hub18]. For this reason, moving toward cubical type theory appears to be non-optional for those who wish type theory to be used as a language for both mathematics and programming at the same time.

◆ **(0.2.2∗5)** *Open term computation.* Closed term computation **(0.2.2∗4)** is a prerequisite for the use of type theory as a programming language, but this property is not sufficient to facilitate the *implementation* such a compiler and a type checker for such a language. In order to write a type checker, it is necessary to *also* perform "symbolic computation" of open terms, *i.e.* terms that have free variables. There are many possible notions of symbolic computation, but only ones with respect to which the term model has computational adequacy are useful — in other words, it must be the case that for two terms $\Gamma \vdash M,N : A$ we have $\Gamma \vdash M \equiv N : A$ if and only if they are taken to the same result under symbolic computation. When this adequacy property holds, the symbolic computation is referred to as *normalization*. There are two main consequences of normalization that facilitate the implementation of compilers and type checkers for dependent type theory:

1) *Inversion laws.* For instance, if $\Gamma \vdash A \to B \equiv A' \to B'$, then both $\Gamma \vdash A \equiv A'$ and

$\Gamma \vdash B = B'$.

2) *Decidability of judgmmental equality.* There is an algorithm to determine whether or not $\Gamma \vdash A \equiv B$ for any two types $\Gamma \vdash A, B$.

Decidability of judgmental equality is needed to implement the *conversion rule*, which closes the typing judgment $\Gamma \vdash a : A$ under type equality. Although decidability gets the most attention, from our point of view the inversion laws are of more fundamental importance for type checking algorithms. Consider the following elaboration clause for λ-abstractions:

$$\mathbf{synth}(\Gamma \vdash \mathtt{M(N)}) \Rightarrow$$
$$(M, A \to B) \leftarrow \mathbf{synth}(\Gamma \vdash \mathtt{M});$$
$$N \leftarrow \mathbf{check}(\Gamma \vdash \mathtt{N} : A);$$
$$\mathbf{return}\ (\mathsf{app}(A, B, M, N), B)$$

The above is deterministic only by virtue of the invertibility of the congruence rule for $(\to)$: there is a unique type $\Gamma \vdash A$ that is the domain of the function type $\Gamma \vdash A \to B$.

(0.2.2∗6) *Denotational proofs of positive and negative syntactic properties.* We mentioned that the consistency property **(0.2.2∗2)** can be proved directly by means of model constructions / denotational semantics; for instance, consistency follows immediately if we can give a model of a type theory in sets where the type **empty** is interpreted by the empty set. The broader phenomenon in play here is that *negative* conditions on syntax can nearly always be proved directly by means of a denotational "counter-model", *i.e.* a model that contains a counterexample. It is more subtle to use denotational semantics to prove *positive* properties of syntax like **(0.2.2∗4)** and **(0.2.2∗5)**; here one must employ models that combine syntax with semantics, which is the main topic of this dissertation.

§0.3. THE *CUBICAL HYPOTHESIS* CONFIRMED

(0.3∗1) The history of dependent type theory can be oriented in terms of the contradiction between syntax and semantics, or to be more precise, the difficulties reconciling the semantical goals of type theory (§0.2.1) with the syntactic goals (§0.2.2). Significant progress was made with the introduction of Homotopy Type Theory [Uni13], which achieved for the first time all the semantical goals in a convincing way; unfortunately, this was carried out by sacrificing *en passant* the syntactic properties that type theorists have found to be important (for applications and implementation). Cubical type theory was designed with the explicit aim to present these semantic properties in a way that preserves the desirable syntactic properties, a research program that we might refer to as the "Cubical Hypothesis".

(0.3∗2) In particular, cubical type theory was originally inspired by a constructive model of general univalence **(0.2.1∗7)** in cubical sets [BCH14]. The constructivity of this model suggested (but did not by itself imply) that one could design a version of

type theory in which terms could be run as programs **(0.2.2∗4)**; this conjecture was verified by Angiuli, Hou (Favonia), and Harper [AHH18] and Huber [Hub18]. After this, two important syntactical properties remained conjectures: the existence of a *standard model* of cubical type theory in homotopy types **(0.2.2∗3)**, and normalization / open term computation **(0.2.2∗5)**. The standard model conjecture was resolved in the 2019 *tour de force* by Awodey, Cavallo, Coquand, Riehl, and Sattler with their *equivariant* model — a result that had yet to be published at the time this dissertation went to press. The contribution of this dissertation is to resolve the normalization conjecture.

※ **(0.3∗3)** Now that the last conjectures in the syntactic metatheory of cubical type theory are resolved, we can finally regard the Cubical Hypothesis confirmed. We do not know what the type theories of the future will look like, but we will insist as a programmatic matter that they carry forward the advances made under the cubical banner.

§0.4. COMPUTATION: DYNAMIC AND STATIC

※ **(0.4∗1)** In the study of programming languages, it is quite common to distinguish between the *statics* and the *dynamics* of a language — as presented by Harper [Har16], the statics of a language include the rules for deducing judgments of a grammatical nature such as $\boxed{\Gamma \vdash a : A}$, which expresses that a is a well-formed element of type A. In this American School consensus on programming languages, the *equality* of programs is not usually considered as part of the statics but instead emerges from the *dynamics*.

※ **(0.4∗2)** The dynamics of a language ("how it computes") are typically given by a transition relation $\boxed{a \longmapsto a'}$ and a distinguished collection of terminal states $\boxed{a \ \textit{final}}$ defined on untyped raw terms without free variables.[7] The statics are united with the dynamics in the following fundamental theorem **(0.4∗3)**.

■ **(0.4∗3)** *Type safety.* Let a be a raw term with no free variables, and let A be a type.

    1) *Progress.* If $\cdot \vdash a : A$, then either a *final* or there exists some a' such that $a \longmapsto a'$.
    2) *Preservation.* If $\cdot \vdash a : A$ and $a \longmapsto a'$, then $\cdot \vdash a' : A$.

§0.4.1. Dynamic equivalence of programs

■ **(0.4.1∗1)** *Observational poles.* When a program is run, the main thing it produces is heat. In order to have a meaningful notion of program equivalence, it is *a priori* necessary that there be at least some measure (an "observational pole") that is capable of distinguishing two programs. We define an observational pole to be a type O together with a subset $\mathcal{O} \subseteq \{a \mid \cdot \vdash a : O\}$ that is closed under *head expansion* in the sense that $a' \in \mathcal{O}$ and $a \longmapsto a'$ together imply $a \in \mathcal{O}$.

---

[7] Here, we understand "raw term" to refer to α-equivalence classes of well-scoped terms.

◆ **(0.4.1∗2)** *Non-termination.* If a language has non-termination and a unit type $\mathbb{1}$, then the collection of terminating programs $a : \mathbb{1}$ can be used as an observational pole.

◆ **(0.4.1∗3)** *Answer types.* In pure languages, the observational pole is defined on a sufficiently discrete *answer type* as explained by Harper [Har16].

$$\overline{\Gamma \vdash \mathsf{yes} : \mathsf{ans}} \qquad\qquad\qquad \overline{\Gamma \vdash \mathsf{no} : \mathsf{ans}}$$

We may define $\mathcal{O}_{\mathsf{ans}}$ to be the set of terms that evaluate to yes.

◼ **(0.4.1∗4)** *Category of term boundaries.* We define a *term boundary* to be a pair $\Gamma \rhd A$ with Γ a context and A a type; a morphism of term boundaries $(\Gamma \rhd A) \longrightarrow (\Delta \rhd B)$ is essentially a term $\Delta \vdash \mathcal{C}[\bullet] : B$ with a single hole $\bullet$ that can be filled by any term $\Gamma \vdash a : A$.[8] A morphism of term boundaries is often called an *evaluation context*.

◼ **(0.4.1∗5)** *Operational equivalence.* A canonical notion of equivalence arises from any observational pole $(O, \mathcal{O})$. Fixing two terms $\Gamma \vdash b, b' : B$, we say that b is observationally equivalent to b' when for all morphisms of term boundaries $\mathcal{C}[\bullet] : (\Gamma \rhd B) \longrightarrow (\cdot \rhd O)$, we have $\mathcal{C}[b] \in \mathcal{O}$ if and only if $\mathcal{C}[b'] \in \mathcal{O}$. We will write $\boxed{\Gamma \vdash_{\mathcal{O}} b \simeq b' : B}$ to mean that b and b' are observationally equivalent relative to the pole $(O, \mathcal{O})$.

◼ **(0.4.1∗6)** It is a standard theorem of operational semantics that observational equivalence is the *coarsest consistent congruence* relative to the chosen pole $(O, \mathcal{O})$. In this setting, a relation R is a called a congruence when it is preserved by all the term formers of the language; and it is consistent when $a \mathrel{R} a'$ implies $(a \in \mathcal{O}) \Leftrightarrow (a' \in \mathcal{O})$ for any $\cdot \vdash a, a' : O$.

※ **(0.4.1∗7)** The fact that observational equivalence is the coarsest consistent congruence relative to a given pole expresses the sense in which observational equivalence is *universal*. The universality of observational equivalence is the reason why theorists of programming languages consider it "the" definition of program equivalence. We will argue that while observational equivalence is a reliable intuitive model for equivalence of programs up to dynamics, it *does not* address the equally pressing question of *static* equivalence.

§0.4.2. Static or "judgmental" equivalence of programs

※ **(0.4.2∗1)** We pointed out in **(0.4∗1)** that the *statics* of an American School programming language typically contains only the laws for constructing well-typed terms, but evinces no notion of program equivalence. Hence a programming language in this sense is a *pure* theory $\mathbb{T}$, a theory presented by generators but no relations.

It is by now standard practice to study such a theory in terms of its *syntactic category* $\mathcal{C}_{\mathbb{T}}$, whose objects are given by contexts Γ and whose morphisms are given by simultaneous

---

[8] The precise definitions depend on the language, and can be found in a standard text such as Harper [Har16].

substitutions $\Delta \longrightarrow \Gamma$. Unfortunately, the syntactic category of such a "pure" theory has no recognizable structure at all except finite products (given by context extension): even if $\mathbb{T}$ has function types in the usual sense, the syntactic category $\mathcal{C}_{\mathbb{T}}$ does not have exponentials! This is because the β/η laws of the function type are present only in the dynamics, and hence do not affect the syntactic category at all.

(0.4.2∗2) Confronted with this unfortunate fact, it is tempting to define $\mathcal{C}_{\mathbb{T}}$ by quotienting the simultaneous substitutions under observational equivalence **(0.4.1∗5)** relative to a suitable pole **(0.4.1∗1)**. On the bright side, the resulting category *does* have the desired structure (*e.g.* exponentials); unfortunately, it doesn't take much investigation to see that this idea is *not* viable: most interpretations $\mathcal{C}_{\mathbb{T}} \longrightarrow \mathcal{E}$ add functions that are not definable in $\mathbb{T}$, but such functions induce new observations that will disrespect operational equivalence. Hence quotienting the syntactic category under operational equivalence has the effect of degenerating the category of models of a given theory.

(0.4.2∗3) One possible response to this state of affairs is to ignore the question of models (and therefore the question of static equivalence) altogether — there is *one* model, and it is the operational model. This point of view, while respectable, exhibits certain fundamental limitations that make it unsuitable for the study of modern programming languages. Indeed, it is not difficult to see that a well-behaved notion of static equivalence is *non-optional* for any language that exhibits type dependency, including Standard ML and OCaml via their module systems, and Haskell via its *type families* mechanism! In either case, a **conversion rule** governing static equivalence cannot be avoided, here phrased in terms of program module typing:

$$\text{CONVERSION} \quad \frac{\Gamma \vdash M : \sigma \qquad \overbrace{\Gamma \vdash \sigma \equiv \tau \; sig}^{\text{static equiv.}}}{\Gamma \vdash M : \tau}$$

While the conversion rule above speaks only of static equivalence of signatures, signatures in ML languages involve not only types but also programs (*e.g.* via singletons). Hence one way or another, it is not optional to account for equivalence of programs; ML languages do this by contracting the collection of programs of a given type to a point under static equivalence in accordance with the *phase distinction* [HMM90; SH21], whereas ordinary dependent type theories must provide a more fine-grained approach to static program equivalence that identifies only β/η-equivalent programs.

(0.4.2∗4) *Judgmental equality.* We will usually refer to the notion of equivalence governed by the conversion rule as *judgmental equality* rather than "static equivalence".

(0.4.2∗5) *Design constraints for judgmental equality.* Three important considerations inform the design of judgmental equality for a type theory or a programming language; these considerations arise from usability, implementability, and applicability respectively.

1) *Realistic.* Judgmental equality must identify things that are "obviously" the same: for instance, it is non-optional to identify $\Gamma \vdash \pi_1 \langle a,b \rangle \equiv a : A$.

2) *Feasible.* We insist that judgmental equality be not only decidable, but also *feasible* in the usual case.[9]

3) *Stable.* Judgmental equality must be stable under extension of the language with new constants and new computations: hence, "observational" notions of equivalence such as **(0.4.1∗5)** that identify two elements on the basis that there is no way to distinguish them are ruled out *a priori*.

※ **(0.4.2∗6)** Balancing the design constraints **(0.4.2∗5)** leads to the following theses concerning the theory of judgmental equality at specific type connectives:

1) For "negative" types (functions, products, limits, *etc.*), judgmental equality should include both β- and η-laws: modern methods (such as normalization by evaluation or hereditary substitutions) easily decide the full equational theory for such types.

2) For "positive" types (sums, tensor products, colimits, *etc.*), judgmental equality should include only β-laws: the η-laws are in some cases decidable, but never feasible.

Because there are some identifications that cannot be implemented by judgmental equality, we also conclude that it is necessary to include an additional form of identification for which proofs are provided by the user. This is usually achieved by adding an *identification* type connective $\mathsf{Id}_A(a,a')$ to a language with the following introduction rule:

$$\frac{\Gamma \vdash a \equiv a' : A}{\Gamma \vdash \mathsf{refl}_{A,a,a'} : \mathsf{Id}_A(a,a')}$$

Inverting the introduction rule above is called *equality reflection*, and this is obviously ruled out by the design constraint of feasibility **(0.4.2∗5)**. Hence one of the main questions investigated by type theorists in the present day is the design of identification type connectives that identify enough things without disrupting other important properties of the language; the state of the art in this area is *cubical type theory* [Ang+19; AHH18; Coh+17], and the main contribution of this dissertation is to prove that judgmental equality in cubical type theory is both realistic and decidable, the latter being a necessary step toward establishing feasibility.

§0.5. SUBJECTIVE METATHEORY: THE MATHEMATICS OF FORMALISMS

▪ **(0.5∗1)** We promote the term *subjective metatheory* for the traditional study of the mathematical properties of formalisms, as opposed to the study of the type theories they

---

[9] Well-known complexity-theoretic results ensure that no useful notion of judgmental equality can be feasible in the worst case, but decades of experience embodied in both ML languages and proof assistants like Agda and Coq have shown that the pathological cases never arise in practice. Deciding β/η-equivalence for dependent type theoretic terms that arise in practice has proved to be completely unproblematic.

present. The subjective metatheory of a formalism for type theory often involves defining reduction relations on raw terms and establishing the admissibility of various rules or reasoning principles (*e.g.* cut or substitution, weakening, strengthening, subject reduction, *etc.*), with an eye toward establishing the following results relative to the presentation:

1) *Canonicity*, the existence of canonical representatives of judgmental equivalence classes for closed terms of base type [AHH17; Hub18; LH12].
2) *Normalization*, the existence of canonical representatives of judgmental equivalence classes for open terms of arbitrary type [Abe09; Abe13; AAD07; ACP09; AVW17; GSB19a].
3) *Sound and complete interpretation*, or the construction of an initial model of type theory by constraining and then quotienting the raw terms of the formalism [Str91].
4) *Elaboration*, the translation of a convenient surface language into type theory [Bra13; Bra+11; CH09; HS00; McB99; Mil+97].

(0.5∗2) Soundness and completeness for a formalism will usually be obtained as a consequence of the normalization theorem, rather than the other way around. The difficulty is to establish the coherence of the interpretation under the very common circumstance that the raw terms of type theory carry less information than the derivations, as explained by Streicher [Str91]. This aspect of the subjective metatheory is simultaneously the most laborious and the least informative; it can be side-stepped entirely in the *objective metatheory* which we detail below.

§0.6. OBJECTIVE METATHEORY: A SYNTAX-INVARIANT PERSPECTIVE

(0.6∗1) We coin the term *objective metatheory* to refer to the study of *presentation-invariant* structures over type theories, *i.e.* structures on type theory that are inherited by all formalisms presenting it. Many aspects of type theory that have historically been studied in a subjective way may also be studied objectively, including canonicity, normalization, conservativity, decidability of equivalence, coherence, and even elaboration.

(0.6∗2) The objective metatheory distinguishes itself from the subjective not only in its invariance, but also through the emphasis of structure over property. In fact, it is only by passing from property (proof-irrelevant structure) to proper structure (proof-relevant structure) that it becomes possible to entertain invariant accounts of the metatheory of type theory, as we will see in Chapter 4.

(0.6∗3) We are inspired by Lawvere's use of the prefix "objective" to refer to the study of the laws of reality (including as a special case the deductive processes of "matter-that-thinks", *i.e.* humans), in contrast to the pure study of the laws of thought [Law94; LS09; Sch00]; Lawvere's distinction, applied in the context of categorical algebra and continuum dynamics, draws from Hegel's opposition of the subjective and the objective [Heg69].

(0.6∗4) In the context of type theory, the study of deduction in formalisms is subjective in the sense that a formal deduction is a thinking subject's reflection on a real process — for instance, in a formalism for Euclidean geometry, the deduction of a formal line from a pair of formal points is a subjective reflection of the objective process of drawing this line, which exists regardless of what rules of construction are distinguished as "basic" or "primitive", and which are distinguished as derived.

(0.6∗5) The subjective metatheory in logic and type theory corresponds to "choosing a coordinate system" or "choosing a basis" in other areas of mathematics; historically, this has been a necessary step in carrying out calculations of a determinate nature, but recent years have furnished evidence that many calculations (including normalization, the computation of canonical representatives of equivalence classes) can be carried out without choosing any "coordinate system" beforehand [Coq19; Fio02]. This is what we mean by the *objective metatheory.*

(0.6∗6) Subjective reflections of real processes may, as a whole, exhibit anomalies and points of tension that mystify and complicate the study of the objects they present. For instance, the bitter controversy between implicit and explicit substitutions in type theory is purely subjective: at the objective level, substitution is characterized by a universal property which can be presented in either an implicit or explicit way without affecting any substantive metatheorem.

§0.6.1. Lawvere theories and objective algebra

(0.6.1∗1) Lawvere laid the groundwork for the *objective metatheory* in his doctoral thesis [Law63] by distinguishing theories from their presentations for the first time, and going on to develop a new kind of categorical model theory called the *functorial semantics.* Aiming to be intelligible to the existing community studying universal algebra, Lawvere restricted his attention to unityped algebraic theories, but his ideas have been extended in the subsequent years to account for sophisticated binding structures as well as dependent sorts [Car78; FH10; FM10; PV07; Uem19].

(0.6.1∗2) Originally, an algebraic theory was a pair (O,E) in which O is a set of operation symbols equipped with arities, and E is a set of equational sequents $\langle n \vdash s \equiv t\rangle$ in which s,t are *terms* of arity n; the terms of arity n are generated inductively by variables x_i with $i < n$ and operations $o(s_0, \ldots, s_{m-1})$ where $o \in \mathsf{O}_m$ and each s_i is a term of arity n.

(0.6.1∗3) Following Lawvere [Law63], we refer to the pair (O,E) as an *equational presentation* of a theory rather than as a theory. The theory that (O,E) presents will be a structure that forgets the difference between generating operations o and derived terms s. Each presentation in this sense generates a collection of *algebras*, which are structures equipped

with operations corresponding to O, obeying the equations in E. Between algebras is a class of homomorphisms, yielding the structure of a category.

◆ **(0.6.1∗4)** *Equational presentation of monoids.* The algebraic theory of monoids can be presented by the following operations and equations:

$$\mathsf{O}_n = \begin{cases} \{\varepsilon\} & n = 0 \\ \{\odot\} & n = 2 \\ \varnothing & \text{otherwise} \end{cases}$$

$$\mathsf{E} = \left\{ \begin{array}{l} \langle 1 \vdash \odot(\varepsilon, x_0) \equiv x_0 \rangle, \\ \langle 1 \vdash \odot(x_0, \varepsilon) \equiv x_0 \rangle, \\ \langle 3 \vdash \odot(x_0, \odot(x_1, x_2)) \equiv \odot(\odot(x_0, x_1), x_2) \rangle \end{array} \right\}$$

The category of *algebras* for this equational presentation is exactly the category of monoids in the ordinary sense!

※ **(0.6.1∗5)** An important structural property of modern algebra is that there is a *equivalence* (an "isomorphism up to isomorphism") between the category of equational presentations and the category of theories: a theory can be presented in many different ways, but these presentations are all isomorphic. The minor differences between presentations that are flattened out at the level of theories often constitute the main source of the difficulty in developing the *subjective* metatheory — located for instance in the very fragile presupposition-admissibility lemmas, *etc.* Consequently, experienced practitioners of the subjective metatheory have accumulated numerous heuristic design principles for obtaining "good" presentations that evince simpler proofs of normalization, decidability of equivalence, coherence, *etc.*

◆ **(0.6.1∗6)** *Cut elimination.* Although any two sequent calculus presentations of intuitionistic first-order logic are necessarily isomorphic in a precise sense, it is much easier to decide equality of proofs in a presentation for which the general *cut* and *identity* rules are admissible but not derivable; most structural proof theorists and type theorists have accordingly developed an attuned awareness of minor differences in presentations that are likely to facilitate or disrupt the admissibility of cut and identity.

◆ **(0.6.1∗7)** The strength of the *objective metatheory* lies in providing tools that are robust under these differences; these tools include (for example) generalized algebraic theories and logical frameworks [Car78; HHP93; Uem19], as well as Artin gluing [Cro93; Fio02; Joh02; KHS19; MS93; SS18; Tay99]. A weapon more recently added to the arsenal of the objective metatheorist is the homology theory of term rewriting systems [Ike19; MM16], a computational invariant that reflects lower bounds on the *number* of operations and equations required to present a given theory. Thus, the invariance of the objective metatheory does *not* prevent its use for studying non-invariant objects.

(0.6.1∗8) According to our terminology, an equational presentation is a specific kind of formalism; other kinds of formalism are possible, and indeed, one of the background motivations of this dissertation is to promote formalisms that present a theory by elaboration rather than by equational constraints. The most famous exemplar of this latter kind of formalism is the surface language of Standard ML.

§0.6.2. Algebraic type theory and logical frameworks

(0.6.2∗1) Since the early 1990s, many type theorists have argued that type theories should not be defined first as "hand-crafted" formalisms, but should instead be given as *mathematical objects* in some simpler framework — for instance, Martin-Löf's Logical Framework (MLLF) or the Edinburgh Logical Framework (LF), or the (binding-free) discipline of Generalized Algebraic Theories (GATs).

(0.6.2∗2) Type theorists have preferred logical frameworks for a number of reasons, including the fact that some logical frameworks (mainly MLLF and LF) enable object-level binders to be represented by a meta-level function space. This facility removes the difficulty and bureaucracy associated with specifying binding structure and substitutions, but it is by no means the only reason to adopt logical frameworks. At the most basic level, logical frameworks free their adopters from the endemic difficulties of the subjective metatheory that pertain to the admissibility of presuppositions and closure under conversion.

(0.6.2∗3) Unfortunately, neither the model theory of MLLF nor even that of LF has ever been satisfactorily worked out: this would entail definitions of categories of models, initial algebras, a functorial semantics in the sense of Lawvere [Law63], as well as an upgrade of the standard Lawvere–Gabriel–Ulmer duality between theories and models. Until this year, these shortcomings have rendered a fully mathematical account of the metatheory of dependent type theories somewhat out of reach.

(0.6.2∗4) Luckily, significant progress has been made by researchers in the past two years: for instance, Uemura defines a logical framework generalizing MLLF that may be used to specify almost any (non-modal) type theory, automatically equipped with a functorial semantics and a syntactic model [Uem19]. The universal property of the syntactic model is the critical ingredient to execute the objective metatheory and obtain canonicity, normalization, *etc*.

In a similar spirit, Gratzer and Sterling [GS20] propose the use of free locally Cartesian closed categories as an alternative way to develop the syntactic categories of type theories and their functorial semantics; while Uemura emphasizes the universal property of the syntactic model, Gratzer and Sterling emphasize the universal property of the syntactic category. We develop an informal and type theoretic version of the perspective of Gratzer and Sterling in Chapter 1.

§0.6.3. Canonical forms and computability: structure *vs.* property

(0.6.3∗1) In the subjective metatheory, the notion of "canonical form" was expressed as a *property* of *raw* terms. For instance, in operational semantics one defines the judgment ⊢ M *value*, and it is clear that this notion of canonical form or value cannot be invariant under judgmental equality without degenerating — for if it were, one would have ⊢ $(\lambda x.x)(5)$ *value.*

(0.6.3∗2) The objective counterpart to this notion, which we will continue to refer to as "canonical form", is to be given as a *structure* over the denotations of *typed* terms in any model $\mathcal{C}$; in the usual case, when $\mathcal{C}$ is the syntactic/generic model of the type theory, the notion of canonical form can be seen to be a structure over equivalence classes of typed terms. One ceases to speak of "M is canonical" (which has no sense if M is an equivalence class of terms) but rather "m is a canonical form of M".

(0.6.3∗3) Adopting a notion of canonical form that is compatible with abstract terms (we mean, typed terms quotiented up to judgmental equality) confers significant advantages. Many of the highly technical aspects of canonicity, normalization, and parametricity proofs can be boiled down to pushing around proofs of properties that are completely non-trivial for raw terms, but totally automatic for abstract terms. For instance, the burdensome *coherent expansion* conditions endemic to the subjective metatheory of cubical type theory [Ang19; AHH17; Hub18] may be totally dispensed with in the objective metatheory, as illustrated in joint work with Angiuli and Gratzer [SAG19; SAG20].

(0.6.3∗4) The notion of canonical forms as structures rather than properties goes back to Peter Freyd's application of *sconing* (gluing along the global sections functor) in 1978, though it can be argued that the structural perspective was latent in the work of Martin-Löf [Mar75a]: already in 1975, Martin-Löf summarizes a computability model in which a dependent sum is used to express the concept of a term together with a witness of computability.

(0.6.3∗5) The "book proof" of canonicity for typed λ-calculus has long been based on structures of canonical forms over equivalence classes of typed terms, and this perspective was extended by Fiore [Fio02] to full normalization, providing the objective counterpart to normalization by evaluation.

(0.6.3∗6) While a number of scientists (including Awodey, Spitters, and others) have long promoted the idea of developing a version of the gluing technique that applies to dependent type theories, the technical realization of this idea was greatly delayed by a pervading misconception within parts of the type theoretic community that partial equivalence relations over raw terms were required; to the author's knowledge, the first application of gluing to dependent type theory appeared in the paper of Shulman [Shu15b], but it was not until the note of Coquand [Coq18] was made public that the technology

became nearly universally understood within the traditional type theoretic community. This dissertation is, in some sense, a product of Coquand's expository work.

(0.6.3∗7) Already in 2016, however, Altenkirch and Kaposi [AK16a] had presented an objective version of normalization by evaluation (without using the language of gluing) for dependent type theory. Since 2018, a number of authors including myself have participated in developing the theory and applications of gluing for dependent type theories with universes [Coq18; Coq19; CHS19; Gra21; KHS19; KS19; Ste18; SA21; SAG19; SAG20; SH21].

§0.6.4. Tait's method, then and now

(0.6.4∗1) How do you prove that every term may be equipped with a canonical form? This question is answered by the *method of computability* introduced by Tait [Tai67] and refined by Girard and Martin-Löf; although the technical details have changed greatly over time (even during the writing of this document), the main idea remains unchanged. The essence of Tait's computability method is to construct an interpretation of the type theory in which each type is rendered as some kind of predicate on the elements of that type, and each operation must preserve this predicate. Depending on which metatheorem one wishes to prove (*e.g.* canonicity for closed terms, canonicity for open terms, parametricity, *etc.*), a number of parameters of the computability model may be tweaked.

(0.6.4∗2) *The classical method of computability.* Classically, computability predicates were construed as families of *propositions* over raw terms (or typed raw terms, not taken up to judgmental equality), indexed in renamings; there is a great deal of technical difficulty in verifying that these objects are well-defined, including:

1) For most metatheorems (but not all), all computable elements must be well-typed.
2) Computability must be closed under judgmental equality.
3) The computability predicates are closed functorially under a class of substitutions.

(0.6.4∗3) Defined in this style, the computability technique does *not* give rise directly to a model of the theory at hand: in other words, it is not the case that in the "semantic domain" (raw terms satisfying some predicates), a β-redex is mathematically the same as its contractum — hence there is really no sense in which this style of computability can be called *semantics*. In all cases it is possible to quotient everything after the fact in order to obtain a model, but we are skeptical that much leverage is gained in this way; the power of the model theory is grasped through direct use, rather than worship from afar.

(0.6.4∗4) The situation becomes strictly more difficult in the context of dependent type theory with universes: it is usually not possible to prove normalization (or even closed canonicity) using ordinary computability predicates over raw terms, and so one has

typically passed to a more complex argument involving *two* semantic constructions of approximately equal difficulty:[10]

1) A computability interpretation, in which one considers "pairs of equal computable elements" rather than computable elements. This interpretation of type theory into partial equivalence relations is enough to establish the *completeness* part of normalization. This stage involves an ingenious but quite subtle fixed point construction about which a number of difficult closure and saturation conditions must be established — one of the main obstacles to a manageable proof of normalization, as can be seen from the technical report of Gratzer, Sterling, and Birkedal [GSB19b].

2) A binary logical relation between the computability interpretation and the raw syntax of the formalism, to establish the *soundness* part of normalization. Here one requires an additional sequence of somewhat technical saturation conditions.

(0.6.4∗5) Recent examples of proving normalization for dependent type theory with universes using *subjective* computability include Gratzer, Sterling, and Birkedal [GSB19a] and Wieczorek and Biernacki [WB18]. There appears to be only a limited extent by which this style of construction can be simplified or modularized, but we note that Abel, Öhman, and Vezzosi [AÖV17] present some improvements.

(0.6.4∗6) *Objective computability, or computability structures.* All of the difficulties described above may be dealt with simultaneously in the context of the objective metatheory, using a *computability structures* technique inspired by Artin gluing; computability families might alternatively be referred to as *proof-relevant computability predicates.* Computability structures exhibit a few main differences from earlier approaches:

1) Computability is a structure rather than a property: the use of structure rather than property allows us to finally overcome the reliance on raw terms and reduction relations; overcoming reduction will be of great importance in the objective metatheory of cubical type theory, in which the standard *untyped head expansion* property of semantic equality is replaced by a much more technical one that involves non-trivial equality and typing conditions.

2) Computability is defined relative to an arbitrary model $\mathcal{C}$; in particular, $\mathcal{C}$ may be the initial model and in this case one may speak of an *equivalence class of typed terms* being computable. One does not consider raw terms at any time.

(0.6.4∗7) A consequence of working with abstract terms (equivalence classes of typed terms) is that the computability families construction does in fact give rise to a *model* of the theory; then, the difficulty in establishing the soundness of normalization for dependent type theory can be discharged using basic algebra: namely, the universal property of the *initial* model $\mathcal{C}$.

---

[10] It is worth noting that in the special case of dependent type theories without universes, such as logical frameworks, much simpler proofs are available, *e.g.* the argument of Harper and Pfenning [HP05].

§0.7. TOWARDS PRINCIPLED IMPLEMENTATION OF PROOF ASSISTANTS

(0.7∗1) Proof assistants comprise many components: concrete syntax, abstract syntax, equivalence checking, type checking, unification, and elaboration (to name a few). While the importance of minimizing the "trusted base" of a proof assistant has long been emphasized as a way to mitigate purely technical errors in implementation (*e.g.* a unification module that emits ill-typed unifiers), comparatively little effort has been spent on ensuring that these "trusted core languages" are actually mathematically meaningful.

(0.7∗2) This question of mathematical meaning is not merely philosophical or hypothetical: interpretation of most type-theoretic formalisms into mathematical models relies on normalization results that have *not* been proved for the core language actually in use.[11] Current tools (rewriting, Kripke logical relations, *etc.*) render the array of necessary metatheorems nearly intractable for the complex formalisms that underlie many proof assistants used today (including Coq, Lean, Agda, Idris, and `red`tt).

(0.7∗3) The algebraic approach to type theory and its objective metatheory promise to at least partly resolve these difficulties, by significantly simplifying the verification of standard metatheorems using modern mathematical tools, and by giving rise to a new style of un-annotated formalism which derives its equational theory directly from the annotated internal language via elaboration, in a manner inspired by the elaborative semantics of Standard ML [CH09; HS00; LCH07].

---

[11] For instance, although an idealized version of Coq's core language is known to be strongly normalizing, there is no corresponding proof for the core language actually in use; this is not a matter of nit-picking, because the "real Coq" uses typed conversion rules that are not currently compatible with the techniques used to prove normalization for Predicative Calculus of Inductive Constructions (pCIC).

CHAPTER 1

OBJECTIVE SYNTAX OF DEPENDENT TYPES

§1.1. TYPE THEORIES AS CATEGORIES OF JUDGMENTS

(1.1∗1) In the study of simple type theory, it is conventional to avoid making distinctions between types, contexts, and judgments: there is an rational basis to this tradition, which is that all these concepts have the same expressivity. More sophisticated type theories, however, force a careful analysis of the distinction between a judgment and a type [Mar87b; Mar96], primarily because the language of judgments carries additional structure that is not found in the language of types.

  1) *Hypothetico-general judgment.* Martin-Löf's description of the *hypothetico-general* judgment $x : E \vdash F[x]$ as a dependent product in the language of judgments [Mar87a] is independent of whether types or contexts are closed under dependent product.

  2) *Judgmental equality.* For any two objects a,b of the same sort, the meaningfulness of a judgment of equality $a \equiv b$ is unconditional; but the closure of the language of judgments under equality judgments does not entail the closure of types under equality types.

(1.1∗2) Hence it is not forced to think of types (or the contexts they generate) as the primary objects of dependent type theory. The objectification of a general type theory as a category in the spirit of Lawvere [Law63] should, then, evince not a category of types/contexts but rather a category of judgments. The presence of hypothetical judgments

Categorical	Judgmental
$\mathrm{E} : \mathcal{T}$	a judgment-form presupposing nothing
$\mathrm{F} : \mathcal{T}_{/\mathrm{E}}$	a judgment-form presupposing E
$\mathrm{F} \longrightarrow \mathrm{G} : \mathcal{T}_{/\mathrm{E}}$	an E-generic deduction of G from F
$\alpha^* : \mathcal{T}_{/\mathrm{F}} \longrightarrow \mathcal{T}_{/\mathrm{E}}$	instantiation of presupposition
$\alpha_* : \mathcal{T}_{/\mathrm{E}} \longrightarrow \mathcal{T}_{/\mathrm{F}}$	hypothetico-general judgment
$\alpha_! : \mathcal{T}_{/\mathrm{E}} \longrightarrow \mathcal{T}_{/\mathrm{F}}$	dependent sum of judgments

Table 1.1: A dictionary of concepts between a type theory and its category of judgments.

of arbitrary order [Mar87a; NPS90; Sch87], as well as the presence of equality judgments forces a working definition of a *type theory* à la Lawvere. The judgmental structure of a type theory will be concentrated in a *locally Cartesian closed* category $\mathcal{T} : \mathbf{LCCC}$, for which we have a dictionary of concepts in Table 1.1.

§1.2. JUDGMENTS AS TYPES AND THE LOGICAL FRAMEWORK

(1.2∗1) If a type theory is to be studied through its category of judgments, how do we *define* this category? Because the language of judgments includes both equality and dependent products, it is itself a form of (extensional) type theory which we will refer to as the *logical framework*; then an ordinary type theory can be conveniently specified as a "signature" in this logical framework. In this section we will describe informally how to construct such signatures via a *meaning explanation* in the sense of Martin-Löf [Mar96].

(1.2∗2) The logical framework that we develop informally here has its roots in the generalized algebraic theories of Cartmell [Car78], Martin-Löf's logical framework [Mar87a; NPS90], the Edinburgh logical framework [HHP93; HL07], and the logical frameworks of Harper [Har21] and Uemura [Uem19]. There are many differences in the design of these logical frameworks; we point out the major design decisions below:

1) Like [Har21; HHP93; NPS90; Uem19] and unlike [Car78], the notions of hypothetical judgment and variable are built into the framework rather than being axiomatized in object theories.
2) Like [HHP93; NPS90] but unlike [Uem19], we place no restriction on which judgments induce a hypothetical form of judgment. This perspective is also adopted in the recent unpublished work of Harper [Har21].
3) Like [Car78; Har21; NPS90; Uem19] we support theories with equations, unlike [HHP93] which is restricted to "pure" theories.
4) Like [Har21; Uem19] and unlike [Car78; NPS90], equations in a theory are treated as generators of a first class *equality type* as opposed to being a separate top-level

construct.

Our design decisions are meant to ensure both convenience and a straightforward connection to free locally Cartesian closed categories.

§1.2.1. Rules for forming signatures

◼ **(1.2.1∗1)** The fundamental notions of the logical framework are those of a *signature*, and an *implementation* of a signature; given a signature $\mathbb{S}$, we will write $U : \mathbb{S}$ to mean that U is an implementation of the signature $\mathbb{S}$. A form of signature is laid down by specifying exactly what its implementations are.

Certain signatures will be distinguished as *judgments*; the purpose of this stratification is to support a "signature of judgments" **(1.2.1∗6)**. An implementation of a judgment $\mathbb{J}$ will be referred to as a *deduction*.

◼ **(1.2.1∗2)** *The trivial judgment.* There is a judgment $\mathbb{1}$ that has exactly one deduction.

◼ **(1.2.1∗3)** *Dependent sum.* If $\mathbb{S}$ is a signature and $\mathbb{T}(x)$ is a signature assuming $x : \mathbb{S}$, then there exists a signature $\sum_{x:\mathbb{S}}\mathbb{T}(x)$ such that an element of $\sum_{x:\mathbb{S}}\mathbb{T}(x)$ is uniquely determined by a pair (U,V) where $U : \mathbb{S}$ and $V : \mathbb{T}(U)$. Moreover if both $\mathbb{S}$ and $\mathbb{T}(x)$ are judgments, then so is $\sum_{x:\mathbb{S}}\mathbb{T}(x)$.

◼ **(1.2.1∗4)** *Dependent product.* If $\mathbb{J}$ is a **judgment** and $\mathbb{T}(x)$ is a signature assuming $x : \mathbb{J}$, then $\prod_{x:\mathbb{J}}\mathbb{T}(x)$ is a signature whose implementations are uniquely determined by implementations $U(x) : \mathbb{T}(x)$ assuming $x : \mathbb{J}$. Moreover, if $\mathbb{T}(x)$ is a judgment then so is $\prod_{x:\mathbb{J}}\mathbb{T}(x)$.

◼ **(1.2.1∗5)** *Equality of implementations.* If $\mathbb{S}$ is a signature and $U,V : \mathbb{S}$ are implementations of $\mathbb{S}$, then there is a signature $U =_{\mathbb{S}} V$ whose implementations are uniquely determined by deductions of $\mathbb{1}$ such that U is equal to V. If $\mathbb{S}$ is a judgment, then so is $U =_{\mathbb{S}} V$.

◼ **(1.2.1∗6)** *Signature of judgments.* There is a signature $\square$ whose implementations are uniquely determined by judgments $\mathbb{J}$.

✒ **(1.2.1∗7)** We will write $\mathbb{J} \to \mathbb{T}$ for the degenerate dependent product $\prod_{\_:\mathbb{J}}\mathbb{T}$.

§1.3. ABSTRACT AND CONCRETE SYNTAX OF TYPE THEORY

※ **(1.3∗1)** What is *abstract syntax*? The abstract syntax of the type theory presented by a signature $\mathbb{S}$ is nothing more than the elements of judgments that can be written down under the assumption of an implementation $\alpha : \mathbb{S}$. A judgment of this type theory is just a function $\mathbb{J} : \mathbb{S} \to \square$, and a deduction of the judgment $\mathbb{J}$ from the judgment $\mathbb{K}$ is just a dependent function $U(\alpha) : \mathbb{J}(\alpha) \to \mathbb{K}(\alpha)$ over $\alpha : \mathbb{S}$. In §1.4 we will develop a more precise mathematical account of the abstract syntax of the type theory presented by $\mathbb{S}$.

※ **(1.3∗2)** The notion of "syntax" that we have described above is extremely general and well-adapted to informal but rigorous manipulation. However, type theory is meant to be *implemented* as a program in a computer: therefore we must also have a kind of **concrete syntax** that can be built up out of trees that are typed into a computerized proof assistant (like Agda). A large part of the subject of type theory is studying the translation of these concrete trees into abstract syntax — this process, called *elaboration*, must not only determine whether a given tree corresponds to an expression of type theory, but it must also fill in bureaucratic details that are too painful for the user to worry about explicitly.[1]

※ **(1.3∗3)** The science of elaboration is the pursuit of maximal usability and interactivity of the interface between a human scientist and a computerized proof assistant. Elaboration is the art of "making possible the impossible": while it is undecidable whether two type theoretic objects in the abstract syntax we presented so far are equal or unequal, we may find *restrictions* on each type theoretic language under which equality is decidable.

※ **(1.3∗4)** The role of contexts in type theory is, in part, to formalize such a restriction; we will find that by carefully choosing which judgments are contexts, we may obtain a decision procedure for equality of type theoretic objects that only make assumptions of contexts, and not of other judgments.

◆ **(1.3∗5)** For instance, if the equality judgment $U =_\mathbb{J} V$ is a context, then it is not likely to be possible to decide the equality of objects defined relative to a context [CCD17]; this fact, often discussed in the context of a formal rule of deduction called *equality reflection*, does not imply that equality is a dangerous or philosophically suspicious concept: it rather shows the importance of carefully choosing what kinds of assumptions (contexts) are allowed. Our perspective, inherited from the Edinburgh tradition of logical frameworks, is that such restrictions should not be part of the type theory itself but instead should be imposed on it as part of the statements of specific metatheorems. We return to the question of contexts in our discussion of *figure shapes* in §4.1.

§1.4. CATEGORIES OF SIGNATURES AND JUDGMENTS

■ **(1.4∗1)** We may define a category **SIG** of signatures: an object of **SIG** is a signature $\mathbb{S}$ defined using the rules of §1.2.1; a morphism $\mathbb{S} \longrightarrow \mathbb{T}$ is an implementation $U(x) : \mathbb{T}$ assuming $x : \mathbb{S}$.

◑ **(1.4∗2)** Each slice $\mathbf{SIG}_{/\mathbb{S}}$ can be thought of as the language of signatures *relative to* or *over* $\mathbb{S}$. We will develop a vocabulary for these slices below.

■ **(1.4∗3)** **SIG** has all finite limits: the terminal object is given by the trivial judgment $\mathbb{1}$, and pullbacks are given by a combination of dependent sum and equality.

---

[1] Conceptually it is best to think of elaboration as a generalization of *parsing*.

■ **(1.4∗4)** *Judgment classifier.* Consider the projection map $\sum_{\mathbb{J}:\square}\mathbb{J}\longrightarrow\square:\mathbf{SIG}$ that takes a deduction of a judgment to that judgment. We will write $\square:\mathbf{SIG}_{/\square}$ for this family when viewed as an object of the slice.

■ **(1.4∗5)** An object $\mathbb{T}:\mathbf{SIG}_{/\mathbb{S}}$ is called *judgmental* when there exists a cartesian morphism $\mathbb{T}\longrightarrow\square:\mathbf{SIG}^{\rightarrow}$.

≋ **(1.4∗6)** Unfolding **(1.4∗5)** explicitly, we have asked for a pullback square in the following configuration, which we may write in either the language of **SIG** or in the language of the codomain functor $\mathbf{SIG}^{\rightarrow}\longrightarrow\mathbf{SIG}$ below:

$$
\begin{array}{ccc}
\mathbb{T} \dashrightarrow \sum_{\mathbb{J}:\square}\mathbb{J} & \qquad & \mathbb{T} \longrightarrow \square \\
\downarrow \quad\lrcorner\qquad \downarrow & \qquad & \downarrow \quad\lrcorner\qquad \downarrow \\
\mathbb{S} \dashrightarrow \square & \qquad & \mathbb{S} \dashrightarrow \square
\end{array}
$$

Hence the judgmental signatures over $\mathbb{S}$ are exactly the families that can be encoded by an actual judgment $\mathbb{S}\longrightarrow\square$.

■ **(1.4∗7)** **SIG** has dependent products along judgmental families: if $f:\mathbb{T}\longrightarrow\mathbb{S}$ is judgmental (arises by pullback from $\square$), then the pullback functor $f^*:\mathbf{SIG}_{/\mathbb{S}}\longrightarrow\mathbf{SIG}_{/\mathbb{T}}$ has a right adjoint $f_*:\mathbf{SIG}_{/\mathbb{T}}\longrightarrow\mathbf{SIG}_{/\mathbb{S}}$.

■ **(1.4∗8)** *Category of judgments.* Given a signature $\mathbb{S}:\mathbf{SIG}$, we define its *category of judgments* $\mathcal{T}_{\mathbb{S}}$ to be the full subcategory of $\mathbf{SIG}_{/\mathbb{S}}$ spanned by judgmental objects. In the future we may simply write $\mathcal{T}$ when the signature is understood.

■ **(1.4∗9)** The category of judgments $\mathcal{T}_{\mathbb{S}}$ is locally Cartesian closed, via **(1.4∗3)** and **(1.4∗7)**.

◷ **(1.4∗10)** The category $\mathcal{T}_{\mathbb{S}}$ gives rise to a *functorial semantics* in the sense of Lawvere [Law63]; if $\mathcal{E}$ is a locally Cartesian closed category, then locally Cartesian closed functors $\mathcal{T}_{\mathbb{S}}\longrightarrow\mathcal{E}$ are equivalent to *models* of $\mathbb{S}$ in $\mathcal{E}$, in the sense that such a functor assigns a morphism to every judgment and a triangle to every deduction.

§1.5. CONSERVATIVITY OF "META-SIGNATURES"

※ **(1.5∗1)** *Meta-signatures.* So far we have discussed signatures and judgments; the collection of judgments itself is a signature, but we cannot have a signature of all signatures. We show in this section that a rich notion of "meta-signature" that includes a "meta-signature of all signatures" is a *conservative extension* of the language of judgments and signatures. The purpose of such a notion is to give life to *schematic* constructions that compose signatures like lego bricks at a high level of generality.

■ **(1.5∗2)** *The Yoneda lemma.* We have a fully faithful embedding $\mathsf{y}:\mathbf{SIG}\hookrightarrow\mathrm{Pr}(\mathbf{SIG})$ from the category of signatures into the category of presheaves on signatures, sending each

$\mathbb{S} : \mathbf{SIG}$ to the representable presheaf $\mathrm{Hom}_{\mathbf{SIG}}(-,\mathbb{S})$. Our perspective will be that an object of $\mathrm{Pr}(\mathbf{SIG})$ is some kind of "meta-signature", and these include (by the Yoneda lemma) all the actual signatures.

■ **(1.5∗3)** A natural transformation $f : \mathrm{A} \longrightarrow \mathrm{B} : \mathrm{Pr}(\mathbf{SIG})$ is called *representable* when every fiber of f over a representable object is again representable, in the sense of the following pullback square:

$$
\begin{array}{ccc}
\mathsf{y}(\mathbb{T}) & \longrightarrow & \mathrm{A} \\
\downarrow & & \downarrow{\scriptstyle f} \\
\mathsf{y}(\mathbb{S}) & \longrightarrow & \mathrm{B}
\end{array}
$$

The notion of representable natural transformation is employed by Awodey [Awo18] to provide a category theoretic reformulation of the notion of *categories with families* [Dyb96]. It is appropriate to think of a representable natural transformation as a *universe* that (weakly) classifies some class of families originating the base category.

■ **(1.5∗4)** Awodey [Awo18] has given a very useful recipe to construct a universe of all families of signatures as a representable natural transformation $\mathsf{sig} : \mathfrak{Sig}' \longrightarrow \mathfrak{Sig}$ in $\mathrm{Pr}(\mathbf{SIG})$:

$$\mathfrak{Sig}' = \coprod_{f \in \mathbf{SIG}^{\rightarrow}} \mathsf{y}(\mathsf{dom}f)$$
$$\mathfrak{Sig} = \coprod_{f \in \mathbf{SIG}^{\rightarrow}} \mathsf{y}(\mathsf{cod}f)$$
$$\mathsf{sig} = \coprod_{f \in \mathbf{SIG}^{\rightarrow}} \mathsf{y}(f)$$

We observe that sig is a representable map, replicating Awodey's argument [Awo18, Proposition 23]. Fixing a signature $\mathbb{S}$ and a morphism $\mathrm{T} : \mathsf{y}(\mathbb{S}) \longrightarrow \mathfrak{Sig}$, we intend to show that the pullback below lies in the image of the Yoneda embedding:

$$
\begin{array}{ccc}
\mathfrak{Sig}' \times_{\mathfrak{Sig}} \mathsf{y}(\mathbb{S}) & \longrightarrow & \mathfrak{Sig}' \\
\downarrow & & \downarrow{\scriptstyle \mathsf{sig}} \\
\mathsf{y}(\mathbb{S}) & \xrightarrow{\ \mathrm{T}\ } & \mathfrak{Sig}
\end{array}
$$

By the Yoneda lemma the morphism $\mathrm{T} : \mathsf{y}(\mathbb{S}) \longrightarrow \mathfrak{Sig}$ is uniquely determined by a

cospan of the following form in **SIG**, whose pullback we depict in dotted lines:

$$
\begin{array}{ccc}
\mathbb{S}' & \xrightarrow{\quad q_\mathrm{T} \quad} & \mathrm{dom}\,f_\mathrm{T} \\
{\scriptstyle p_\mathrm{T}}\Big\downarrow & \lrcorner & \Big\downarrow{\scriptstyle f_\mathrm{T}} \\
\mathbb{S} & \xrightarrow[\quad g_\mathrm{T} \quad]{} & \mathrm{cod}\,f_\mathrm{T}
\end{array}
$$

Again by the Yoneda lemma, the map $q_\mathrm{T} : \mathbb{S}' \longrightarrow \mathrm{dom}\,f_\mathrm{T}$ determines a map $\mathrm{T}' : \mathsf{y}(\mathbb{S}') \longrightarrow \mathfrak{Sig}'$. It suffices to observe that the following square is cartesian:

$$
\begin{array}{ccc}
\mathsf{y}(\mathbb{S}') & \xrightarrow{\quad \mathrm{T}' \quad} & \mathfrak{Sig}' \\
{\scriptstyle \mathsf{y}(p_\mathrm{T})}\Big\downarrow & & \Big\downarrow{\scriptstyle \mathrm{sig}} \\
\mathsf{y}(\mathbb{S}) & \xrightarrow[\quad \mathrm{T} \quad]{} & \mathfrak{Sig}
\end{array}
$$

▨ **(1.5∗5)** Conversely, any morphism $\mathbb{T} \longrightarrow \mathbb{S} : $ **SIG** has a (non-unique) code $\mathsf{y}(\mathbb{S}) \longrightarrow \mathfrak{Sig}$ in the sense of the following diagram:

$$
\begin{array}{ccc}
\mathsf{y}(\mathbb{T}) & \dashrightarrow & \mathfrak{Sig}' \\
\Big\downarrow & \lrcorner & \Big\downarrow \\
\mathsf{y}(\mathbb{S}) & \dashrightarrow & \mathfrak{Sig}
\end{array}
$$

This is also shown by Awodey [Awo18, Corollary 24]; the upstairs and downstairs maps correspond under the Yoneda lemma to the cospan and upper-right composite of the following square respectively:

$$
\begin{array}{ccc}
\mathbb{T} & \xrightarrow{\quad \mathbf{id}_\mathbb{T} \quad} & \mathbb{T} \\
\Big\downarrow & \lrcorner & \Big\downarrow \\
\mathbb{S} & \xrightarrow[\quad \mathbf{id}_\mathbb{S} \quad]{} & \mathbb{S}
\end{array}
$$

We will not use the following strengthening of **(1.5∗5)**.

▨ **(1.5∗6)** $\mathfrak{Sig}' \longrightarrow \mathfrak{Sig}$ is *generic* for representable maps in the sense that every representable map arises from it by pullback, albeit in a non-unique way. Let $p : \mathrm{A} \longrightarrow \mathrm{B}$ be a

representable map; we wish to exhibit a pullback square in the following configuration:

$$
\begin{array}{ccc}
A & \dashrightarrow & \mathfrak{Sig}' \\
{\scriptstyle p}\downarrow & \lrcorner & \downarrow{\scriptstyle \mathsf{sig}} \\
B & \dashrightarrow & \mathfrak{Sig}
\end{array}
$$

Because p is a representable map, for each $f : \mathsf{y}(\mathbb{S}_f) \longrightarrow B$ we have a pullback square of the following kind:

$$
\begin{array}{ccc}
\mathsf{y}(\mathbb{S}'_f) & \overset{q_f}{\dashrightarrow} & A \\
{\scriptstyle \mathsf{y}(p_f)}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\
\mathsf{y}(\mathbb{S}_f) & \underset{f}{\longrightarrow} & B
\end{array}
$$

Every presheaf is a colimit of representables; in particular, the cocone formed by all morphisms $f : \mathsf{y}(S) \longrightarrow B$ is universal. The universality of colimits states that the cocone formed by all the induced morphisms $q_f : \mathsf{y}(\mathbb{S}'_f) \longrightarrow A$ is also universal. Hence using the universal property of the colimit, we obtain a square $p \longrightarrow \mathsf{sig}$ restricting along each $f : \mathsf{y}(\mathbb{S}_f) \longrightarrow B$ to $\mathsf{y}(p_f)$:

$$
\begin{array}{ccccc}
\mathsf{y}(\mathbb{S}'_f) & \overset{q_f}{\longrightarrow} & A & \overset{\chi'}{\longrightarrow} & \mathfrak{Sig}' \\
{\scriptstyle \mathsf{y}(p_f)}\downarrow & \lrcorner & \downarrow{\scriptstyle p} & & \downarrow{\scriptstyle \mathsf{sig}} \\
\mathsf{y}(\mathbb{S}_f) & \underset{f}{\longrightarrow} & B & \underset{\chi}{\longrightarrow} & \mathfrak{Sig}
\end{array}
\qquad (1.5{*}6{*}1)
$$

The outer square of Diagram 1.5*6*1 is evidently cartesian for each f, but the pullback lemma does not deduce that the right-hand square is cartesian from such an assumption. Using descent for coproducts, the family of diagrams above on the left can be repackaged as a single pullback square in which the downstairs (and hence the upstairs) morphism is

an effective epimorphism:

$$
\begin{array}{ccccccc}
y(\mathbb{S}'_f) & \xrightarrow{\;\mathsf{in}_f\;} & \coprod_f y(\mathbb{S}'_f) & \xrightarrow{\;[f.q_f]\;} & A & \xrightarrow{\;\chi'\;} & \mathfrak{Sig}' \\
{\scriptstyle y(p_f)}\big\downarrow & & \big\downarrow{\scriptstyle \coprod_f y(p_f)} & & \big\downarrow{\scriptstyle p} & & \big\downarrow{\scriptstyle \mathsf{sig}} \\
y(\mathbb{S}_f) & \xrightarrow{\;\mathsf{in}_f\;} & \coprod_f y(\mathbb{S}_f) & \xrightarrow{\;[f.f]\;} & B & \xrightarrow{\;\chi\;} & \mathfrak{Sig}
\end{array}
\qquad (1.5*6*2)
$$

The outer black square of Diagram 1.5*6*2 is cartesian because its restrictions along the coproduct injections are cartesian. Because the left-hand black square is a pullback along an effective epimorphism, the right-hand black square is cartesian too. □

(1.5*7) Further results of Awodey [Awo18, Proposition 28] establish that $\mathfrak{Sig}$ is closed under codes for signature connectives corresponding to dependent sum/product.

(1.5*8) By the Yoneda lemma, anything expressible in the language of **SIG** has a counterpart in $\mathrm{Pr}(\mathbf{SIG})$; because the Yoneda embedding preserves finite limits and dependent products, we are free to naïvely use the internal language of $\mathrm{Pr}(\mathbf{SIG})$ as if it were the language of **SIG**.

Anything written in the language of $\mathrm{Pr}(\mathbf{SIG})$ that doesn't involve meta-signatures can be proved in the language of **SIG**, also by virtue of the Yoneda lemma. This is the sense in which we may freely employ constructs such as the meta-signature **SIG** of all signatures as a convenience in the *interior* of constructions involving signatures; the Yoneda lemma shows that such a construction can be unrolled at the end of the day to a (more difficult) construction that involves only actual signatures.

§1.6. TYPE THEORETIC BUILDING BLOCKS

(1.6*1) We will immediately impose a number of convenient but informal notations for working with (meta)-signatures and judgments in the language of $\mathrm{Pr}(\mathbf{SIG})$. Formally, the objects of an implementation of a signature are accessed *positionally*, by iterated projection and application: in particular, the name of the bound variable x in the dependent sum $\sum_{x:\mathbb{J}}\mathbb{K}(x)$ has no meaning as it is only a placeholder. However, it will be convenient to adopt a notation by which we associate names to positions within a signature.

(1.6∗2) We will use an Agda-style notation for defining (parameterized) signatures and judgments:

> **record** $\mathbb{S}$ $(x : \mathbb{T})$: $\mathfrak{Sig}$ **where**
> u : $\mathbb{J}(x)$
> v : $\mathbb{K}(x,\mathsf{u})$
> w : $\mathbb{L}(x,\mathsf{u},\mathsf{v})$

The above defines a parameterized signature $\mathbb{S} : \mathbb{T} \to \mathfrak{Sig}$. We additionally have the following three projection schemes, using curly braces to indicate an implicit parameter in our notation:

$$-.\mathsf{u} : \{x : \mathbb{T}\} \; \mathbb{S}(x) \to \mathbb{J}(x)$$
$$-.\mathsf{v} : \{x : \mathbb{T}\} \; \textstyle\prod_{y:\mathbb{S}(x)}\mathbb{K}(x,y.\mathsf{u})$$
$$-.\mathsf{w} : \{x : \mathbb{T}\} \; \textstyle\prod_{y:\mathbb{S}(x)}\mathbb{L}(x,y.\mathsf{u},y.\mathsf{v})$$

(1.6∗3) As an example, Martin-Löf's type theory has two forms of judgment A *type* and $a : $ A; we may render this type theory as the following signature:

> **record** $\mathbb{ML}_\varnothing$: $\mathfrak{Sig}$ **where**
> tp : $\square$
> tm : tp $\to \square$

(1.6∗4) A common design pattern is to build up signatures from components, often parameterized in other signatures. These components, which we might call "snippets", are especially useful when specifying a complex type theory.

(1.6∗5) The concept of an isomorphism between two judgments can be expressed as a judgment snippet $\mathsf{Iso}(\mathbb{J},\mathbb{K})$:

> **record** Iso $(\mathbb{J} : \square, \mathbb{K} : \square)$: $\square$ **where**
> intro : $\mathbb{J} \to \mathbb{K}$
> elim : $\mathbb{K} \to \mathbb{J}$
> coh_β : $\prod_{x:\mathbb{J}} x =_\mathbb{J} \mathsf{elim}(\mathsf{intro}(x))$
> coh_η : $\prod_{x:\mathbb{K}} x =_\mathbb{K} \mathsf{intro}(\mathsf{elim}(x))$

We will write $\mathbb{J} \cong \mathbb{K}$ for $\mathsf{Iso}(\mathbb{J},\mathbb{K})$.

(1.6∗6) We define a judgment snippet $\mathsf{fam} : \mathbb{ML}_\varnothing \to \square$ that takes an implementation of the judgmental structure of Martin-Löf type theory to a compound judgment classifying type families.

> **record** fam $(\alpha : \mathbb{ML}_\varnothing)$: $\square$ **where**
> base : $\alpha.\mathsf{tp}$
> fib : $\alpha.\mathsf{tm}(\mathsf{base}) \to \alpha.\mathsf{tp}$

§1.7. MARTIN-LÖF'S TYPE THEORY WITH UNIVERSES

◼ **(1.7∗1)** What do we mean by *Martin-Löf's type theory*? We are referring to a version of dependent type theory with forms of judgment A *type* and $a : A$, equipped with the following structure:

1) Types are closed under dependent product, dependent sum, and booleans.
2) There is a hierarchy of universes with associative and unital lift coercions, each closed under dependent product, dependent sum, booleans, and the smaller universes. The decoding operation of each universe commutes strictly with the chosen codes for type constructors.

We distinguish the above from *extensional* type theory which adds extensional equality types at each universe level. A precise definition of this type theory is presented in Fig. 1.2.[2]

⚒ **(1.7∗2)** Fix an implementation $M : \mathbb{ML}_{ext}$ and define the following constants:

$$\Pi : \prod_{A:M.\mathsf{tp}} \prod_{B:M.\mathsf{tm}(A) \to M.\mathsf{tp}} M.\mathsf{tp}$$

$$\lambda : \prod_{A:M.\mathsf{tp}} \prod_{B:M.\mathsf{tm}(A) \to M.\mathsf{tp}} \prod_{f:\prod_{x:M.\mathsf{tm}(A)} M.\mathsf{tm}(B(x))} M.\mathsf{tm}(\Pi(A,B))$$

$$\mathsf{app} : \prod_{A:M.\mathsf{tp}} \prod_{B:M.\mathsf{tm}(A) \to M.\mathsf{tp}} \prod_{f:M.\mathsf{tm}(\Pi(A,B))} \prod_{x:M.\mathsf{tm}(A)} M.\mathsf{tm}(B(x))$$

◆ **(1.7∗3)** *Externalizing.* The object $\mathbb{ML}_{ext}$ from Fig. 1.2 is a global element of $\mathfrak{Sig} : \mathrm{Pr}(\mathbf{SIG})$, in the sense that it refers only to defined constants; hence, the representability of $\mathfrak{Sig}' \longrightarrow \mathfrak{Sig}$ guarantees that it corresponds to an actual external signature $\mathbb{S} : \mathbf{SIG}$:

$$
\begin{array}{ccc}
\mathsf{y}(\mathbb{S}) & \longrightarrow & \mathfrak{Sig}' \\
\downarrow & \lrcorner & \downarrow \\
\mathsf{y}(\mathbb{1}) & \xrightarrow{\ \mathbb{ML}_{ext}\ } & \mathfrak{Sig}
\end{array}
$$

Hence we define the category of judgments of Martin-Löf's extensional type theory to be $\mathcal{T}_{\mathbb{S}}$, the full subcategory of $\mathbf{SIG}_{/\mathbb{S}}$ spanned by judgmental objects.

---

[2] We take the somewhat nonstandard step of omitting intensional identity types from the definition of Martin-Löf's type theory, because we are mainly investigating both extensional identity types and cubical path types.

record $\mathbb{J}_{\mathsf{bool}}$ $(M : \mathbb{ML}_\varnothing) : \square$ **where**
  open M
  bool : tp
  tt,ff : tm(bool)
  $\mathsf{ind}_{\mathsf{bool}} : \prod_{C:\mathsf{tm(bool)}\to\mathsf{tp}}\prod_{c_0:\mathsf{tm}(C(\mathsf{tt}))}\prod_{c_1:\mathsf{tm}(C(\mathsf{ff}))}\prod_{x:\mathsf{tm(bool)}}\mathsf{tm}(C(x))$
  $\_ : \{C,c_0,c_1\}\ \mathsf{ind}_{\mathsf{bool}}(C,c_0,c_1,\mathsf{tt}) =_{\mathsf{tm}(C(\mathsf{tt}))} c_0$
  $\_ : \{C,c_0,c_1\}\ \mathsf{ind}_{\mathsf{bool}}(C,c_0,c_1,\mathsf{ff}) =_{\mathsf{tm}(C(\mathsf{ff}))} c_1$

record $\mathbb{J}_\Pi$ $(M : \mathbb{ML}_\varnothing, F : \mathsf{fam}(M)) : \square$ **where**
  open M,F
  Π : tp
  $\lambda : \left(\prod_{x:\mathsf{tm(base)}}\mathsf{tm}(\mathsf{fib}(x))\right) \cong \mathsf{tm}(\Pi)$

record $\mathbb{J}_\Sigma$ $(M : \mathbb{ML}_\varnothing, F : \mathsf{fam}(M)) : \square$ **where**
  open M,F
  Σ : tp
  $\mathsf{pair} : \left(\sum_{x:\mathsf{tm(base)}}\mathsf{tm}(\mathsf{fib}(x))\right) \cong \mathsf{tm}(\Sigma)$

record $\mathbb{J}_{\mathsf{eq}}$ $(M : \mathbb{ML}_\varnothing, A : M.\mathsf{tp}, a_0 : M.\mathsf{tm}(A), a_1 : M.\mathsf{tm}(A)) : \square$ **where**
  open M
  eq : tp
  $\mathsf{refl} : (a_0 =_{\mathsf{tm}(A)} a_1) \cong \mathsf{tm}(\mathsf{eq})$

Figure 1.1: Auxiliary judgmental snippets that will be used in the LF signature for Martin-Löf's type theory. For instance, $\mathbb{J}_{\mathsf{bool}}(M)$ expresses the structure of a boolean type atop an implementation M of Martin-Löf's type theory; likewise, $\mathbb{J}_\Pi(M,F)$ expresses the structure of M being closed under the dependent product of the type family F.

record $\mathbb{ML}_{base}$: $\mathfrak{Sig}$ **where**
  $\mathsf{M}_\alpha : \mathbb{ML}_\varnothing$ for each $\alpha : \mathbf{L}$
  $\mathsf{tp}_\alpha, \mathsf{tm}_\alpha := \mathsf{M}_\alpha.\mathsf{tp}, \mathsf{M}_\alpha.\mathsf{tm}$
  $\mathsf{tp}, \mathsf{tm} := \mathsf{tp}_\diamond, \mathsf{tm}_\diamond$

  $\langle\uparrow_\alpha^\beta\rangle : \mathsf{tp}_\alpha \to \mathsf{tp}_\beta$ for each $\alpha \leq \beta$
  $\_ : \{A\}\ \langle\uparrow_\alpha^\alpha\rangle A =_{\mathsf{tp}_\alpha} A$
  $\_ : \{A\}\ \langle\uparrow_\beta^\gamma\rangle\langle\uparrow_\alpha^\beta\rangle A =_{\mathsf{tp}_\gamma} \langle\uparrow_\alpha^\gamma\rangle A$
  $\_ : \{A\}\ \mathsf{tm}_\alpha(A) =_\square \mathsf{tm}(\langle\uparrow_\alpha^\diamond\rangle A)$
  $\_ : \{A,B\}\ \langle\uparrow_\alpha^\beta\rangle A =_{\mathsf{tp}_\beta} \langle\uparrow_\alpha^\beta\rangle B \to A =_{\mathsf{tp}_\alpha} B$

  include $\mathbb{J}_{\mathsf{bool}}(\mathsf{M}_\diamond)$
  include $\prod_{F:\mathsf{fam}(\mathsf{M}_\diamond)}\mathbb{J}_\Pi(\mathsf{M}_\diamond, F)$
  include $\prod_{F:\mathsf{fam}(\mathsf{M}_\diamond)}\mathbb{J}_\Sigma(\mathsf{M}_\diamond, F)$
  $\mathsf{bool}_\diamond, \Pi_\diamond, \Sigma_\diamond := \mathsf{bool}, \Pi, \Sigma$

  $\mathsf{U}_n : \mathsf{tp}_{n+1}$ for each $n < \diamond$
  $\_ : \mathsf{tm}_{n+1}(\mathsf{U}_m) =_\square \mathsf{tp}_m$

  $\Pi_n, \Sigma_n : \mathsf{fam}(\mathsf{tp}_n, \mathsf{tm}_n) \to \mathsf{tp}_n$ for each $n < \diamond$
  $\mathsf{bool}_0 : \mathsf{tp}_0$

  $\_ : \{A,B\}\ \langle\uparrow_\alpha^\beta\rangle\Pi_n(A,B) =_{\mathsf{tp}_\beta} \Pi_\beta(\langle\uparrow_\alpha^\beta\rangle A, \langle\uparrow_\alpha^\beta\rangle \circ B)$
  $\_ : \{A,B\}\ \langle\uparrow_\alpha^\beta\rangle\Sigma_n(A,B) =_{\mathsf{tp}_\beta} \Sigma_\beta(\langle\uparrow_\alpha^\beta\rangle A, \langle\uparrow_\alpha^\beta\rangle \circ B)$

record $\mathbb{ML}_{ext}$: $\mathfrak{Sig}$ **where**
  include $\mathbb{ML}_{base}$
  include $\prod_{A:\mathsf{tp};a_0,a_1:\mathsf{tm}(A)}\mathbb{J}_{\mathsf{eq}}(\mathsf{M}_\diamond, A, a_0, a_1)$
  $\mathsf{eq}_\diamond := \mathsf{eq}$
  $\mathsf{eq}_n : (\sum_{A:\mathsf{tp}_n}\mathsf{tm}_n A \times \mathsf{tm}_n A) \to \mathsf{tp}_n$ for each $n < \diamond$
  $\_ : \{A,a_0,a_1\}\ \langle\uparrow_\alpha^\beta\rangle\mathsf{eq}_\alpha(A,a_0,a_1) =_{\mathsf{tp}_\beta} \mathsf{eq}_\beta(\langle\uparrow_\alpha^\beta\rangle A, a_0, a_1)$

Each universe is closed under the booleans as well as all the smaller universes:

$$\mathsf{bool}_\alpha : \mathsf{tp}_\alpha \qquad \mathsf{U}_{n,\alpha} : \mathsf{tp}_\alpha \quad \text{for each } n < \alpha$$
$$\mathsf{bool}_\alpha = \langle\uparrow_0^\alpha\rangle\mathsf{bool}_0 \qquad \mathsf{U}_{n,\alpha} = \langle\uparrow_{n+1}^\alpha\rangle\mathsf{U}_n$$

Figure 1.2: The LF signature for Martin-Löf's type theory with universes. In the above, α, β range over the poset $\{n \in \mathbb{N}\} \cup \{\diamond\}$, setting $\diamond > n$ for all $n \in \mathbb{N}$.

Part II

Mathematical Background

CHAPTER 2

THE LANGUAGE OF TOPOI

※ **(2.0∗1)** We give an introduction to the language of *topoi* and *logoi* in order to lay the groundwork for the subsequent chapters; our presentation and exposition is greatly influenced by that of Anel and Joyal [AJ21], but we have attempted to emphasize examples that are relevant to computer scientists. This expository chapter contains no new results:[1] it is intended as a *reference* and can be skipped and referred back to at will, depending on the reader's needs and level of experience.

The *topos* is a generalization of topological space that is suitable for developing a broader spectrum of mathematics, including both algebraic geometry and computer science; topoi differ from topological spaces in multiple important ways that make them more broadly applicable. The language of topoi can be learned more thoroughly from standard references [AGV72; Joh02; MM92] supplemented by more recent perspectives [AJ21; Vic07], but we will carefully explain the parts of topos theory that we will need.

---

[1] Aside from some very convenient and strict constructions of modal universes.

※ **(2.0∗2)** In many expositions, the notion of a *sheaf* is defined with respect to a *site*, which is a category equipped with something called a Grothendieck topology; while this very concrete characterization can be useful in practice, it is too technical to lead to good intuitions for the concept of a sheaf or topos theory more generally. Instead, we will use the abstract characterization due to Giraud of a *category of sheaves*, *i.e.* a category that is equivalent to sheaves on a site. We follow the terminology of Anel and Joyal [AJ21] and refer to a category of sheaves as a *logos*, to emphasize the fact that the 2-category of logoi behaves very differently from the 2-category of categories.

▪ **(2.0∗3)** A cocomplete and left exact category $\mathcal{E}$ is a *logos* when it satisfies the following conditions:

1) *Colimits are universal:* given a morphism $f : \mathrm{F} \longrightarrow \mathrm{E} : \mathcal{E}$, the pullback functor $f^* : \mathcal{E}_{/\mathrm{E}} \longrightarrow \mathcal{E}_{/\mathrm{F}}$ is cocontinuous.

2) *Coproducts are disjoint:* the intersection of the coproduct injections is empty in the sense that the following diagram is always cartesian (a pullback):

$$
\begin{array}{ccc}
\mathbf{0}_{\mathcal{E}} & \longrightarrow & \mathrm{E} \\
\downarrow & \lrcorner & \downarrow \\
\mathrm{F} & \longrightarrow & \mathrm{E} + \mathrm{F}
\end{array}
\tag{2.0∗3∗1}
$$

3) *Equivalence relations are effective:* for an equivalence relation $\mathrm{R} \rightarrowtail \mathrm{E} \times \mathrm{E}$ in $\mathcal{E}$, elements of E are identified in the quotient E/R if and only if they are identified by R, in the sense the following diagram is cartesian:

$$
\begin{array}{ccc}
\mathrm{R} & \xrightarrow{\ \pi_1\ } & \mathrm{E} \\
{\scriptstyle \pi_2}\downarrow & \lrcorner & \downarrow{\scriptstyle e} \\
\mathrm{E} & \xtwoheadrightarrow[\ e\ ]{} & \mathrm{E}/\mathrm{R}
\end{array}
\tag{2.0∗3∗2}
$$

4) $\mathcal{E}$ *has a small generating set*: there exists a small set of objects $\check{\mathcal{E}} \subseteq \mathcal{E}$ such that for any two morphisms $f,g : \mathrm{E} \longrightarrow \mathrm{F}$, we have $f = g$ if and only if every diagram of the following form commutes for $\mathrm{C} \in \check{\mathcal{E}}$:

$$
\mathrm{C} \longrightarrow \mathrm{E} \underset{g}{\overset{f}{\rightrightarrows}} \mathrm{F}
\tag{2.0∗3∗3}
$$

☯ **(2.0∗4)** We can give some intuitive justification for why the conditions enumerated in the definition of a logos **(2.0∗3)** are "good". The universality of colimits can be thought of in two ways:

1) It means that the notion of a colimit is *type theoretic* in the sense that it commutes with substitution (pullback).

2) In combination with the condition that $\mathcal{E}$ have a small generating set, it implies (by the adjoint functor theorem) that each pullback functor has a right adjoint. But the right adjoint to pullback is of course dependent product; so the universality of colimits expresses the type theoretic condition that dependent products exist.

The effectivity of equivalence relations ensures that quotients work "correctly"; quotients exist in many categories, but they are essentially useless without the effectivity condition. Partial equivalence relations are an example of a category whose quotients are not usable in practice — leading to the ironic state of affairs in which Nuprl [Con+86] became (by virtue of its semantics in PERs) the first type theoretic proof assistant to provide quotient types, which its main users then proceeded to fastidiously avoid because of usability issues that can ultimately be traced to the lack of effectivity. Setoids on the other hand are an example of a category that *does* have usable quotients.

Finally, the existence of a small generating set is an essentially technical condition that ensures, among other things, that all limits exist and that the hypotheses of the adjoint functor theorem apply.

(2.0∗5) A morphism between logoi $\mathcal{E} \longrightarrow \mathcal{F}$ is simply a functor that preserves finite limits and all colimits.

(2.0∗6) It is a consequence of the axioms **(2.0∗3)** that every logos $\mathcal{E}$ has a *subobject classifier* $\top : \mathbf{1}_\mathcal{E} \rightarrowtail \Omega$, a "universal monomorphism". The universal property of the subobject classifier is that any monomorphism $U \rightarrowtail E$ in $\mathcal{E}$ corresponds to a *unique* "characteristic map" $\chi_U : E \longrightarrow \Omega$ such that the following diagram is cartesian:

$$
\begin{array}{ccc}
U & \longrightarrow & \mathbf{1}_\mathcal{E} \\
\downarrow & \lrcorner & \downarrow \\
E & \dashrightarrow & \Omega \\
& \chi_U &
\end{array}
$$

We now come to the definition of a *topos*.

(2.0∗7) *The topos.* A (Grothendieck) topos $\mathbf{X}$ is defined by a logos called $\mathbf{Set_X}$; a morphism of topoi $f : \mathbf{X} \longrightarrow \mathbf{Y}$ is defined by a morphism $f^* : \mathbf{Set_Y} \longrightarrow \mathbf{Set_X}$ between logoi in the sense of **(2.0∗5)**. The functor f^* is called the *inverse image* part of f; the cocontinuity of f^* ensures that we additionally have a right adjoint functor $f^* \dashv f_* : \mathbf{Set_X} \longrightarrow \mathbf{Set_Y}$ called the *direct image*; it is important to remember that the direct image is only a functor between categories, not a morphism of logoi. We will write **Topos** for the 2-category of (Grothendieck) topoi, where a 2-cell $f \longrightarrow g$ is a natural transformation of inverse image functors $f^* \longrightarrow g^*$.

⚓ **(2.0∗8)** Given a topos **X**, an object of the logos $\mathbf{Set_X}$ is referred to as a *sheaf* on **X**; it is also appropriate to refer to it as a *set* over **X**.

◔ **(2.0∗9)** It may seem as though the concepts of topos and logos are mutually redundant; but it is actually very important to distinguish between them in order to maintain order and correct intuitions. The technical difference is that their morphisms go in the opposite direction, but the motivation is that a topos is not a category but rather a geometrical object (whose constituents are some kind of points), whereas its logos is an algebraic object that can be used to *measure* or *detect* aspects of the topos.

The relationship between topoi and logoi is typical of geometry–algebra dualities:
   ◆ The duality between varieties and ideals exposed in Hilbert's Nullstellensatz [Hil91].
   ◆ The duality between affine schemes and commutative rings in algebraic geometry [Gro60].
   ◆ The duality between locales and their frames of opens [Joh82].

◆ **(2.0∗10)** The *punctual topos* $\mathbb{1}$ is defined by the equation $\mathbf{Set_1} = \mathbf{Set}$. Every topos **X** is equipped with a unique morphism $\mathbf{X} : \mathbf{X} \longrightarrow \mathbb{1}$; the inverse image $\mathbf{X}^* : \mathbf{Set} \longrightarrow \mathbf{Set_X}$ sends a set A to the "constant sheaf" $\coprod_{a \in A} \mathbf{1_{Set_X}}$; the direct image $\mathbf{X_*} : \mathbf{Set_X} \longrightarrow \mathbf{Set}$ is the global sections functor, sending a sheaf $E : \mathbf{Set_X}$ to the hom set $\mathrm{Hom}_{\mathbf{Set_X}}(\mathbf{1_{Set_X}}, E)$.

※ **(2.0∗11)** The type theoretic significance of the subobject classifier **(2.0∗6)** is that it is a *strictly univalent* universe of propositions. While the subobject classifier itself is not preserved by most morphisms of topoi, the existence of a subobject classifier implies all the important compatibility conditions between limits and colimits in a topos (such as universality of colimits, disjointness of coproducts, and effectivity of equivalence relations).[2] For this reason, Lawvere and Tierney have advanced an alternative and more general notion of *elementary topos* based on the presence of a subobject classifier.

Elementary topoi are similar to topoi in the sense described here, but their corresponding categories need not be cocomplete and they may not have a small generating set; consequently, not every elementary topos **X** has a morphism $\mathbf{X} \longrightarrow \mathbb{1}$ where $\mathbb{1}$ is the punctual topos **(2.0∗10)**, though such a morphism is essentially unique if it does exist. However, there is a concept of *bounded morphism* of elementary topoi that reconstructs in greater generality the important aspects of the small generating set in the Giraud definition; writing **BTop** for the category of elementary topoi and bounded morphisms, one can reconstruct the Grothendieck topos theory **Topos** as the slice $\mathbf{BTop}_{/\mathbb{1}}$.

More generally, the relative topos theory $\mathbf{BTop}_{/\mathbf{S}}$ over an elementary topos **S** behaves a lot like Grothendieck topos theory. If **Y** is a bounded **S**-topos, $\mathbf{Set_Y}$ is not necessarily cocomplete when regarded as an ordinary category, but it is cocomplete when regarded as an internal category in $\mathbf{Set_S}$. In fact, there is a precise sense in which any bounded

---

[2] This is related to the way that Martin-Löf type theory needs either a universe or a large elimination to prove that a coproduct type is disjoint.

S-topos is an "internal Grothendieck topos" in **Set$_S$**. Hence, Grothendieck topos theory is just bounded elementary topos theory over $\mathbb{1}$.

■ **(2.0∗12)** A *point* of a topos **X** is defined to be a morphism of topoi $\mathbb{1} \longrightarrow \mathbf{X}$.

◆ **(2.0∗13)** *Presheaves.* Let $\mathcal{C}$ be a small category; a *presheaf* on $\mathcal{C}$ is defined to be a functor $\mathcal{C}^{\mathrm{op}} \longrightarrow \mathbf{Set}$, and the category of presheaves so-defined is written $\mathrm{Pr}(\mathcal{C})$; one observes that $\mathrm{Pr}(\mathcal{C})$ is a logos in the sense of **(2.0∗3)**, hence we have also defined a topos $\widehat{\mathcal{C}}$, setting $\mathbf{Set}_{\widehat{\mathcal{C}}} = \mathrm{Pr}(\mathcal{C})$. There is a fully faithful functor $\mathsf{y}_{\mathcal{C}} : \mathcal{C} \longrightarrow \mathrm{Pr}(\mathcal{C})$ called the *Yoneda embedding*. The presheaf/Yoneda construction has several different universal properties, some of which we explore in **(2.0∗14)** and **(2.3.1∗5)**.

■ **(2.0∗14)** The presheaf construction gives rise to a functor $\mathbf{Cat} \longrightarrow \mathbf{Cat}_{cc}$ from the category of categories to the category of cocomplete categories and cocontinuous functors. This functor is *left adjoint* to the obvious forgetful functor $\mathbf{Cat}_{cc} \longrightarrow \mathbf{Cat}$, hence it is called *free cocompletion.*

※ **(2.0∗15)** The universal property **(2.0∗14)** is category-theoretic rather than topos-theoretic. We will see in §2.2 that the presheaf construction also has a topos-theoretic universal property, due to Diaconescu [Dia75].

§2.1. TOPOI ARE GENERALIZED SPACES

◆ **(2.1∗1)** *Topological spaces.* Let $\mathbf{Esp}_{sob}$ be the category of (sober) topological spaces and continuous maps, and let **Topos** be the category of topoi. We have a *fully faithful* functor $\mathsf{Env} : \mathbf{Esp}_{sob} \longrightarrow \mathbf{Topos}$ sending each topological space to its *enveloping topos*, witnessing the sense in which the language of topoi is a conservative extension of the language of sober topological spaces.[3]

Let $\mathrm{X} : \mathbf{Esp}_{sob}$ be a sober topological space; the enveloping topos $\mathsf{Env}(\mathrm{X})$ is defined by setting $\mathbf{Set}_{\mathsf{Env}(\mathrm{X})}$ to be the full subcategory of $\mathrm{Pr}(\mathcal{O}_{\mathrm{X}})$ spanned by objects E that treat open covers of X as covers, in the sense that unique lifts exist for diagrams like the following when $\mathrm{S} \rightarrowtail \mathsf{y}(\mathrm{U})$ is the *covering sieve* associated to some open cover of U:[4]

$$
\begin{array}{ccc}
\mathrm{S} & \longrightarrow & \mathrm{E} \\
\downarrow & \nearrow^{\exists!} & \downarrow \\
\mathsf{y}(\mathrm{U}) & \longrightarrow & \mathbb{1}_{\mathrm{Pr}(\mathcal{O}_{\mathrm{X}})}
\end{array}
\qquad\qquad (2.1{*}1{*}1)
$$

That the resulting subcategory is in fact a logos is a classical theorem of topos theory; in fact, categories of this form were the first examples of logoi to be exposed. Hence

---

[3] All spaces that arise in actual mathematics and computer science are sober. [4] For an open cover $\mathrm{U} = \cup_i \mathrm{U}_i$, the corresponding sieve S is the family of all open subsets $\mathrm{V} \subseteq \mathrm{U}$ such that $\mathrm{V} \subseteq \mathrm{U}_i$ for some i.

$\mathsf{Env}(X)$ so-defined is a topos; using the fact (which we don't prove here) that Env is fully faithful, it is easy to see that the isomorphism classes of topos theoretic points of $\mathsf{Env}(X)$ can be placed in bijection with the actual points of X.

(2.1∗2) The purpose of the orthogonality condition specified by Diagram 2.1∗1∗1 is to ensure that open covers of X are treated geometrically as covers in $\mathbf{Set}_{\mathsf{Env}(X)}$. This is not automatic, because the Yoneda embedding $\mathcal{O}_X \hookrightarrow \mathrm{Pr}(\mathcal{O}_X)$ preserves no colimits at all! Hence to obtain the correct category of sheaves, one restricts to those presheaves that treat the Yoneda embeddings of open covers as actual covers.

(2.1∗3) An *open* of a topos $\mathbf{X}$ is defined to be a subobject of the terminal sheaf $U \rightarrowtail \mathbf{1}_{\mathbf{Set}_{\mathbf{X}}}$. One sees that the opens have the structure of a *frame*, written $\mathcal{O}_{\mathbf{X}}$.

(2.1∗4) Let $X : \mathbf{Esp}$ be a sober topological space; exhibit an explicit bijection between the isomorphism classes of topos theoretic opens of $\mathsf{Env}(X)$, in the sense of **(2.1∗3)**, and open sets of X in the sense of classical point-set topology.

(2.1∗5) The subobject classifier $\Omega : \mathbf{Set}_{\mathbf{X}}$ can be seen to be an *internal frame* in the language of $\mathbf{Set}_{\mathbf{X}}$. Pushing Ω forward along the terminal map $\mathbf{X} \longrightarrow \mathbb{1}$, one then obtains an ordinary frame $\mathbf{X}_*\Omega$ and this can be seen to be the frame $\mathcal{O}_{\mathbf{X}}$ of opens of $\mathbf{X}$.

(2.1∗6) *Open immersions.* Let U be an open of a topos $\mathbf{X}$; in the same way that an open subset of a topological space yields an open subspace, the topos $\mathbf{X}$ can be restricted to a subtopos $\mathbf{X}_U$ defined by the equation $\mathbf{Set}_{\mathbf{X}_U} = (\mathbf{Set}_{\mathbf{X}})_{/U}$. The pullback map $\mathbf{Set}_{\mathbf{X}} \longrightarrow (\mathbf{Set}_{\mathbf{X}})_{/U}$ is the inverse image part of a morphism of topoi $\mathbf{X}_U \hookrightarrow \mathbf{X}$; any morphism of topoi that is equivalent to one of this form is called an *open immersion.*

(2.1∗7) Open immersions are a subclass of a more general class of morphisms of topoi called *embeddings*. A morphism $f : \mathbf{X} \longrightarrow \mathbf{Y}$ is called an embedding when the direct image functor $f_* : \mathbf{Set}_{\mathbf{X}} \longrightarrow \mathbf{Set}_{\mathbf{Y}}$ is fully faithful.

(2.1∗8) In the case of open immersions, the direct image $(\mathbf{Set}_{\mathbf{X}})_{/U} \hookrightarrow \mathbf{Set}_{\mathbf{X}}$ is the *dependent product* along $U \rightarrowtail \mathbf{1}_{\mathbf{Set}_{\mathbf{X}}}$. The logos $(\mathbf{Set}_{\mathbf{X}})_{/U}$ is equivalently the full subcategory of $\mathbf{Set}_{\mathbf{X}}$ spanned by sheaves E for which the canonical map $E \longrightarrow E^U$ is an isomorphism.

§2.2. CLASSIFYING TOPOI AND GEOMETRIC THEORIES

※ **(2.2∗1)** The collection of points of a topological space X has the structure of a preorder: given two points $x, y \in$ X, we say that "x specializes y" or $x \leq y$ when any open set containing x also contains y. Bound up in the specialization preorder of points of a topological space is an important relationship to logic: it is profitable to think of the opens of a topological space as *propositions* in a certain logical theory $\mathbb{T}$ determined by that space; whereas a point of a topological space can be thought of as a *model* of this logical theory. The preorder structure on opens corresponds to *logical entailment* U $\vdash$ V, and the specialization order on points can be rephrased "model-theoretically" as follows:

$$x \leq y \Leftrightarrow \forall \text{U} : \mathcal{O}_\text{X}.\ x \models \text{U} \implies y \models \text{U}$$

In this sense, the space X can be called the *classifying space* of the theory $\mathbb{T}$, in the sense that points of X correspond to models of $\mathbb{T}$ and specializations of points corresponds to homomorphisms of models. The kind of logic that $\mathbb{T}$ presents is called *propositional geometric logic*, which is built up from relation symbols, set-indexed disjunctions, and finite conjunctions; hence $\mathbb{T}$ is called a *propositional geometric theory*.

The frame of opens $\mathcal{O}_\text{X}$ is therefore the (preordered) *classifying category* or *category of contexts* of the theory $\mathbb{T}$ classified by X; in fact, following Lawvere [Law63] it is often most appropriate to refer to $\mathcal{O}_\text{X}$ itself as the theory.

☋ **(2.2∗2)** The correspondence between topological spaces and propositional geometric theories really only works for *sober* spaces; but an arbitrary topological space *can* be seen as a propositional geometric theory equipped with a distinguished class of models.

◔ **(2.2∗3)** *Non-propositional theories.* What about theories that have sorts and constants? That these cannot be classified by topological spaces can be seen in two (dual) ways:
   1) The category of contexts of a non-propositional theory is not a preorder, hence cannot be of the form $\mathcal{O}_\text{X}$ for a topological space X.
   2) The category of models and homomorphisms of a non-propositional theory is not a preorder, hence it cannot be given by the collection of points $x \in$ X under the specialization order.

The purpose of the invention of topoi by Grothendieck [AGV72] was, in essence, to have classifying spaces for non-propositional theories! When a topos **X** is viewed as a space, its category of points Hom($\mathbb{1}$,**X**) can be viewed as the category of set-theoretical models for a general *geometric theory*, which is a theory axiomatized by sorts, constants, set-indexed disjunctions, finite conjunctions, and existential quantifiers; more generally, a morphism of topoi **Y** $\longrightarrow$ **X** is a model of **X** internal to **Set**$_\textbf{Y}$, *i.e.* a model where sorts are interpreted as sheaves on **Y**. Similarly to how $\mathcal{O}_\text{X}$ acted as a category of contexts for a propositional geometric theory, the logos **Set**$_\textbf{X}$ acts like a category of contexts for the general geometric theory classified by **X**.

▪ **(2.2∗4)** A topos that classifies a propositional geometric theory is called a *locale*; locales are a generalization of sober topological spaces.

◆ **(2.2∗5)** *The empty theory.* The simplest (clearly propositional) geometric theory is the *empty* theory with no relations or axioms at all; the classifying topos of the empty theory is the punctual topos $\mathbb{1}$. This identification provides us a new way to understand the sense in which $\mathbb{1}$ is the terminal topos: up to isomorphism there is exactly one model of the empty theory in any category of sheaves $\mathbf{Set_Y}$, hence there is an essentially unique morphism $\mathbf{Y} \longrightarrow \mathbb{1}$

§2.2.1. The theory of a set

◆ **(2.2.1∗1)** *The theory of a set.* One of the simplest and most useful geometric theories is the theory with a single sort and no operations; a model of this theory in the category of sets would just be a set, and a model in another logos $\mathbf{Set_X}$ would be a sheaf on $\mathbf{X}$. We will write $\mathbb{A}$ for the classifying topos of this theory; it is appropriate to speak of $\mathbb{A}$ as the *topos of sets* (because its points are sets), being very careful to avoid confusing $\mathbf{Set}_\mathbb{A}$ with the totally different *logos* of sets!

◑ **(2.2.1∗2)** *Dualizing object.* $\mathbb{A}$ plays a very important role in topos geometry as a *dualizing object*: in particular, $\mathbb{A}$ translates the algebraic notion of a sheaf into a geometrical notion. The category of morphisms $\mathbf{X} \longrightarrow \mathbb{A}$ forms a logos which is in fact equivalent to $\mathbf{Set_X}$! We will undertand this relationship in a deeper way when we encounter the topos $\mathbb{A}^\bullet$ later on, and introduce the concept of *étale morphism* and *topos étalé* which gives a third perspective on the sheaf concept.

◆ **(2.2.1∗3)** *Abstract variable binding.* If $\mathbb{A}$ is the classifying topos of a single sort, then we already know what it looks like to work inside of $\mathbf{Set}_\mathbb{A}$: it is roughly an intuitionistic type theory extended by a single "abstract type" V with no additional information concerning its elements or how they may be used. The abstractness of V gives us our first glimpse at logoi that axiomatize a notion of *bound variable* or higher-order abstract syntax: here V behaves like a type of variables, and exponentials like $\mathsf{V} \to \mathsf{E}$ might be used to express variable binding.

Many features that we might want from an axiomatization of variable binding are not available, however: for instance, it is not the case that V has decidable equality (hence there can be no internally definable substitution action on syntax trees defined using V at the leaves), and we do not even have an abundance of fresh variables. These defects can be resolved by passing to stronger and stronger theories.

Nevertheless, the logos $\mathbf{Set}_\mathbb{A}$ has been used as a location to study algebraic theories with variable binding, notably by Fiore and Hur [FH10], Fiore and Mahmoud [FM10], and Fiore, Plotkin, and Turi [FPT99] and others. In the author's view, it is the "purest" form of variable binding and has been shown to be very useful.

§2.2.2. The theory of a pointed set and *topoi étalé*

◆ **(2.2.2∗1)** Now we consider the theory generated by a single sort and a constant of that sort, *i.e.* the theory of a pointed set. We will write $\mathbb{A}^\bullet$ for the classifying topos of this theory; the internal language of $\mathbf{Set}_{\mathbb{A}^\bullet}$ must, by definition, have an abstract type V together with a constant $\mathsf{v} : \mathsf{V}$. It can be shown that $\mathbf{Set}_{\mathbb{A}^\bullet}$ is simply the slice $(\mathbf{Set}_\mathbb{A})_{/\mathsf{V}}$ — with this computation in hand, the constant v is just the "variable" (concretely, the identity map $\mathsf{V} \longrightarrow \mathsf{V}$). In general, taking the slice of a category over an object corresponds to *freely adding* an element of that object.

⬇ **(2.2.2∗2)** For a topos $\mathbf{X}$ and a sheaf $\mathrm{E} : \mathbf{Set}_\mathbf{X}$, the topos dual to the slice logos $(\mathbf{Set}_\mathbf{X})_{/\mathrm{E}}$ is so important that we will write $\mathbf{X}_\mathrm{E}$ for it. Most type theorists will recall the very useful fact that when $\mathbf{Set}_\mathbf{X}$ is a category of presheaves $\mathrm{Pr}(\mathcal{C})$, then the slice logos $(\mathbf{Set}_\mathbf{X})_{/\mathrm{E}}$ is as well — we may present $(\mathbf{Set}_\mathbf{X})_{/\mathrm{E}}$ and hence $\mathbf{Set}_{\mathbf{X}_\mathrm{E}}$ as the category of presheaves $\mathrm{Pr}(\mathsf{y}_\mathcal{C} \downarrow \mathrm{E})$ on the category of elements of E.

▪ **(2.2.2∗3)** *Étale maps and topoi étalé.* The "weakening map" $\mathbf{Set}_\mathbf{X} \longrightarrow (\mathbf{Set}_\mathbf{X})_{/\mathrm{E}}$ taking $\mathrm{F} : \mathbf{Set}_\mathbf{X}$ to $\pi_\mathrm{E} : \mathrm{E} \times \mathrm{F} \longrightarrow \mathrm{E}$ can be seen to be the inverse image part of a morphism of topoi $\mathbf{X}_\mathrm{E} \twoheadrightarrow \mathbf{X}$; a morphism (equivalent to one) of this form is called an *étale map* and the domain $\mathbf{X}_\mathrm{E}$ is called the *topos étalé* of E.

▪ **(2.2.2∗4)** Every étale morphism of topoi arises by pullback from the *universal* étale map $\mathbb{A}^\bullet \longrightarrow \mathbb{A}$ in an essentially unique way:

$$
\begin{array}{ccc}
\mathbf{X}_\mathrm{E} & \dashrightarrow & \mathbb{A}^\bullet \\
\downarrow & \lrcorner & \downarrow \\
\mathbf{X} & \dashrightarrow & \mathbb{A} \\
& \mathrm{E} &
\end{array}
$$

☯ **(2.2.2∗5)** The beauty of the *topos étalé* viewpoint is that it provides a geometrical counterpart to the algebraic notion of sheaves. Any construction that makes sense within the logos $\mathbf{Set}_\mathbf{X}$ also makes sense in the full subcategory $\acute{\mathbf{E}}\mathbf{t}_\mathbf{X} \subseteq \mathbf{Topos}_{/\mathbf{X}}$ spanned by topoi étalé over $\mathbf{X}$, *i.e.* étale maps into $\mathbf{X}$; in fact, these two categories are equivalent — the identification (sheaves, topoi étalé, étale morphisms of topoi) generalizes the classical identification (sheaves on a topological space, espaces étalé, local homeomorphisms); on the other hand, there is *no analogue* to the topos-theoretic characteristic map $\mathbf{X} \longrightarrow \mathbb{A}$ in point-set topology, except in the case of a subterminal sheaf (proposition). The existence of characteristic maps for all sheaves is a striking feature of topos geometry, an example of the way topoi provide a more realistic candidate for "general topology" than topological spaces or locales.

※ **(2.2.2∗6)** A computer scientist might observe that the topos of sets $\mathbb{A}$ acts as a *universe* that strictifies the notion of a sheaf *qua* étale morphism — one has a strictly functorial

"substitution action" on characteristic maps of sheaves given by precomposition, in contrast to the pullback perspective which is functorial only in a weak sense.

§2.2.3. The theory of an inhabited set

◆ **(2.2.3∗1)** A subtle variation on the theory of a pointed set is the theory of an *inhabited set*. What is the difference? An inhabited set is one for which "there exists" an element, but we don't keep track of what that element is; this can be axiomatized by a sort V together with the assertion that $\exists x : \mathsf{V}.\top$. The classifying topos for this theory is written $\mathbb{A}^\circ$, and we compute it in a geometrical style in **(2.2.3∗2)** below.

⯆ **(2.2.3∗2)** We can define $\mathbb{A}^\circ$ to be the *image* of the universal étale map $\mathbb{A}^\bullet \dashrightarrow \mathbb{A}$:

$$
\begin{array}{ccc}
\mathbb{A}^\bullet & \xrightarrow{\;\textit{étale map}\;} & \mathbb{A} \\
 & \textit{étale cover} \searrow \quad \nearrow \textit{open immersion} & \\
 & \mathbb{A}^\circ &
\end{array}
$$

$$(2.2.3\ast2\ast1)$$

Above both components of the image factorization are étale; using the identification between topoi étalé and sheaves, we can rephrase Diagram 2.2.3∗2∗1 above as an image factorization *internally* to the logos $\mathbf{Set}_\mathbb{A}$:

$$
\begin{array}{ccc}
\mathsf{V} & \xrightarrow{\hspace{3cm}} & \mathbf{1}_{\mathbf{Set}_\mathbb{A}} \\
 & \textit{cover} \searrow \quad \nearrow \textit{mono} & \\
 & \{\mathbf{1}_{\mathbf{Set}_\mathbb{A}} \mid \exists x : \mathsf{V}.\top\} &
\end{array}
$$

$$(2.2.3\ast2\ast2)$$

§2.2.4. The theory of a proposition/open

◆ **(2.2.4∗1)** *Sierpiński topos.* So far we have seen two ways to think of opens of a topos: as *propositions* or subterminal sheaves $\mathsf{U} \rightarrowtail \mathbf{1}_{\mathbf{Set}_\mathbf{X}}$, or as open immersions of topoi $\mathbf{U} \hookrightarrow \mathbf{X}$. We will now examine a third perspective on opens, via the classifying topos of opens $\mathbb{S}$, also called the *Sierpiński topos*. $\mathbb{S}$ is defined by the equation $\mathbf{Set}_\mathbb{S} = \mathbf{Set}^\rightarrow$, and has two topos theoretic points in the sense of **(2.0∗12)**:

  1) The *open* point $\circ : \mathbb{1} \hookrightarrow \mathbb{S}$, whose inverse image part is the codomain functor $\mathsf{cod} : \mathbf{Set}^\rightarrow \longrightarrow \mathbf{Set}$.
  2) The *closed* point $\bullet : \mathbb{1} \hookrightarrow \mathbb{S}$, whose inverse image part is the domain functor $\mathsf{dom} : \mathbf{Set}^\rightarrow \longrightarrow \mathbf{Set}$.

(2.2.4∗2) The open point $\circ : \mathbb{1} \hookrightarrow \mathbb{S}$ is the *universal* open immersion, in the sense that every open immersion of topoi arises from it by pullback in an essentially unique way. To be precise, let $j : \mathbf{U} \hookrightarrow \mathbf{X}$ be an open immersion; then there is an essentially unique morphism of topoi $\mathbf{X} \longrightarrow \mathbb{S}$ such that the following diagram is cartesian in **Topos**:

$$
\begin{array}{ccc}
\mathbf{U} & \longrightarrow & \mathbb{1} \\
{\scriptstyle j}\downarrow & \lrcorner & \downarrow{\scriptstyle \circ} \\
\mathbf{X} & \dashrightarrow & \mathbb{S} \\
& \scriptstyle \exists! &
\end{array}
$$

(2.2.4∗3) From a geometrical perspective, the Sierpiński topos can be thought of as a *directed interval* $\{\bullet \longrightarrow \circ\}$ in the language of topoi. Recalling **(2.0∗7)**, a morphism of points $\bullet \longrightarrow \circ$ is nothing more than a natural transformation of inverse image functors $\iota : \mathsf{dom} \longrightarrow \mathsf{cod} : \mathrm{Hom}(\mathbf{Set}^{\rightarrow}, \mathbf{Set})$ which we may construct as follows:

$$\iota_f : \mathsf{dom}(f) \longrightarrow \mathsf{cod}(f)$$
$$\iota_f(x) = f(x)$$

Given a topos $\mathbf{X}$, one may form a *cylinder* $\mathbf{X} \times \mathbb{S}$ by Cartesian product equipped with open/closed immersions $\langle \mathbf{id_X}, \circ \rangle : \mathbf{X} \hookrightarrow \mathbf{X} \times \mathbb{S}$ and $\langle \mathbf{id_X}, \bullet \rangle : \mathbf{X} \hookrightarrow \mathbf{X} \times \mathbb{S}$ respectively. Pinching either the closed or open copy of $\mathbf{X}$ along a morphism of topoi is, then, the geometrical content of Artin gluing to be exposed later in §§2.4 and 4.3.

(2.2.4∗4) For the sake of building intuition, it is worth computing what a sheaf on a *cylinder* $\mathbf{X} \times \mathbb{S}$ should be. We may think of a sheaf on a topos as a "set" that varies "continuously" over the points of that topos; hence a sheaf on the Sierpiński topos is a *morphism* of sets, with the domain set covering the closed point and the codomain set covering the open point. Likewise, a sheaf on the cylinder $\mathbf{X} \times \mathbb{S}$ is nothing more than a family of sheaves on $\mathbf{X}$; hence we may compute $\mathbf{Set_{X \times \mathbb{S}}} = \mathbf{Set_X}^{\rightarrow}$.

§2.2.5. The theory of a clock: step-indexing and guarded recursion

(2.2.5∗1) In the theory of programming languages, it is often necessary to find solutions to *domain equations* that cannot be solved in the category of sets. The classical solution to such problems is to move to a category in which such solutions do exist: categories of dcpos or Scott domains form one solution, but more recently the utility of a much simpler construction has been studied by Birkedal, Møgelberg, Schwinghammer, and Støvring [Bir+11a] and adopted by many practitioners: the category of presheaves $\mathrm{Pr}(\omega)$, where ω is the preorder $\{1 \leq 2 \leq 3 \leq \ldots\}$.

As a *logos* it is appropriate to refer to $\mathrm{Pr}(\omega)$ as the "logos of trees" following the traditional terminology; but it is perhaps more appropriate to refer to corresponding *topos*

$\mathbf{K} := \widehat{\omega}$ as the "topos of clocks". Why? Because $\mathbf{X}$-valued points $\mathbf{X} \longrightarrow \mathbf{K}$ of $\mathbf{K}$ are *clocks* in $\mathbf{Set_X}$.

(2.2.5∗2) *The theory of clocks.* A *clock* is a monotone sequence of propositions $\phi_{(i \in \omega)}$ such that the additional axiom $\vdash \bigvee_{i \in \omega} \phi_i$ holds; the intuitive meaning of each proposition ϕ_k is "There are currently k many steps left", and the clock axiom states that the concept of "now" is meaningful. In §2.3.1 we will introduce the tools needed to see that $\mathbf{K}$ is actually the classifying topos of clocks, but we will simply take it on faith here.

§2.2.6. The theory of a vector clock

(2.2.6∗1) A guarded fixed point is a special kind of infinite object that expresses some resource sensitivity, forcing maps out of guarded fixed points to exhibit "causality" when unfolding their arguments — in other words, consumption and production proceed in lock-step. On the other hand, coinductive definitions lack this resource sensitivity — a map out of a coinductive object may unfold its argument as many times as it likes, though in a computational setting, certain continuity principles may be admissible.

The question of whether coinduction can be reduced to guarded recursion was investigated by Atkey and McBride [AM13]; their solution was (in essence) to replace the clock with a *parameterized* clock, *i.e.* a decidable object K of clock names together with a model of the theory of clocks for every $\kappa : \mathsf{K}$. Then a guarded recursive definition over a generic clock $\kappa : \mathsf{K}$ can be turned into a coinductive definition by taking a limit over all $\kappa : \mathsf{K}$. The internal parameterization of Atkey and McBride [AM13] was somewhat *ad hoc* and ill-behaved, but Bizjak and Møgelberg [BM20] studied a better-behaved variant that omits the decidability of the parameterizing object.[5]

(2.2.6∗2) For any finite set E, one can formalize the notion of |E|-many clocks by taking a product $\prod_{e \in E} \mathbf{K}$ in the category of topoi. The *internal* parameterization of Atkey and McBride [AM13] and Bizjak and Møgelberg [BM20] is however not of this form: instead one has internally an *abstract type* K of clock names, which is not necessarily finite nor necessarily infinite, and definitely not discrete.

The first attempts to explain this phenomenon in the language of category theory involved indexing a family of topoi over a category of "clock contexts" [BM15]; while this approach persisted in the literature for some years, it was plain that a better account could be obtained by replacing "(topos of guarded recursion) indexed in clock contexts" with "topos of (guarded recursion indexed in clock contexts)" via the Grothendieck construction — as promoted by Sterling and Harper [SH18] for instance and adopted by Bizjak and Møgelberg [BM20].

The superiority of the latter bracketing was not only located in its support for strict interpretations of dependent type theory: it exposes the fact that the topos models of

---

[5] Their construction is however still *ad hoc* as we will see in **(2.2.6∗8)**.

multi-clock guarded recursion are instances of a very canonical construction, the *lower bag topos* of Vickers [Vic92] — a topos-theoretic categorification of the lower (Hoare) powerdomain from the classical theory of programming languages. In the parlance of Vickers, then, the topoi modeling multi-clock guarded recursion classify the theory of "a bag of clocks", which we might somewhat rudely refer to as the *theory of a vector clock*.

(2.2.6∗3) Powerdomains, once dubbed the "waterloo of denotational semantics", are a tool to model concurrent and non-deterministic computation. If D is a domain, then the lower powerdomain $\mathbf{P_L}D$ is approximately the collection of downward closed subsets of D, putting aside some subtleties. The intuition of powerdomains is that an element of $\mathbf{P_L}D$ captures the collection of values that a non-deterministic computation is taking; the different versions of the powerdomain correspond to different semantics of non-determinism (*e.g.* "may" *vs.* "must", *etc.*).

Aside from the variety of possible powerdomains (lower, upper, mixed, *etc.*), one limitation of powerdomains is that their elements are just subsets ordered by inclusion, and hence transitions between these subsets cannot evince any relationship to (*e.g.*) a concrete thread pool; in contrast to operational semantics of concurrency, the powerdomain semantics is quite lossy. In the language of domains, this is quite unavoidable because the specialization order on the points of a domain obviously cannot carry any data.

(2.2.6∗4) While investigating the denotational semantics of databases, Vickers [Vic92] observed that this limitation could be lifted by replacing domains with topoi, in essence because topoi have *categories* of points rather than *preorders* of points. Given a topos **D**, one can form the *lower bag topos* $\mathbf{B_L D}$ whose points correspond to *indexed families* or *bags* of **D**-points rather than sets. In generality, a morphism $\mathbf{X} \longrightarrow \mathbf{B_L D}$ corresponds to the data of a sheaf $E : \mathbf{Set_X}$ together with a morphism $\mathbf{X}_E \longrightarrow \mathbf{D}$ [Joh02]. One has an obvious projection $p : \mathbf{B_L D} \longrightarrow \mathbb{A}$ that takes a bag of **D**-points to its parameterizing set.

(2.2.6∗5) *Sets.* The lower bag topos $\mathbf{B_L}\mathbb{1}$ of the point is simply the topos of sets $\mathbb{A}$ from §2.2.1: a morphism of topoi $\mathbf{X} \longrightarrow \mathbf{B_L}\mathbb{1}$ is given by a sheaf $E : \mathbf{Set_X}$ together with a morphism $\mathbf{X}_E \longrightarrow \mathbb{1}$, but of course there is exactly one of the latter.

(2.2.6∗6) *Subsets.* Recall the Sierpiński topos $\mathbb{S}$ from **(2.2.4∗1)**; the lower bag topos $\mathbf{B_L}\mathbb{S}$ classifies *subsets*; a morphism of topoi $\mathbf{X} \longrightarrow \mathbf{B_L}\mathbb{S}$ is given by a sheaf $E : \mathbf{Set_X}$ together with a morphism $\mathbf{X}_E \longrightarrow \mathbb{S}$, *i.e.* an open of the topos étalé $\mathbf{X}_E$, *i.e.* a subobject of E.

(2.2.6∗7) *Families of sets.* A variation on **(2.2.6∗6)** can be obtained by replacing $\mathbb{S}$ with $\mathbb{A}$; in this case, the lower bag topos $\mathbf{B_L}\mathbb{A}$ classifies *families of sets*: a morphism of topoi $\mathbf{X} \longrightarrow \mathbf{B_L}\mathbb{A}$ is given by a sheaf $E : \mathbf{Set_X}$ together with a morphism $\mathbf{X}_E \longrightarrow \mathbb{A}$, *i.e.* a sheaf on $\mathbf{X}_E$, *i.e.* a family of sheaves $F \longrightarrow E : \mathbf{Set_X}$.

(2.2.6∗8) *Vector clocks.* We may reconstruct the topos of multi-clock guarded recursion considered by Bizjak and Møgelberg [BM20] as the lower bag topos $\mathbf{B_L}\mathbf{K}$ of the topos

of clocks. A morphism $\mathbf{X} \longrightarrow \mathbf{B_L K}$ consists of a sheaf of clock names $K : \mathbf{Set_X}$ together with a morphism $\mathbf{X}_K \longrightarrow K$, *i.e.* a monotone sequence of subobjects $\varphi_n \rightarrowtail K$ satisfying the geometrical axiom $\kappa : K \mid \cdot \vdash \bigvee_n \kappa \in \varphi_n$.

◆ **(2.2.6∗9)** *Non-empty vector clocks.* It is now quite easy to see how the multi-clock guarded recursion topos of Sterling and Harper [SH18] relates to that of Bizjak and Møgelberg [BM20]. Sterling and Harper chose to validate an additional (*a priori* non-geometric) axiom that ensures a well-behaved intersection over degenerately clock-indexed relations:

$$\forall \psi : \Omega.\ (\forall \kappa : K.\psi) =_\Omega \psi \qquad\qquad (\text{"clock irrelevance"})$$

It is easy enough to see that the above is equivalent to the geometrical axiom $\exists \kappa : K.\top$. This suggests that we can obtain the topos of Sterling and Harper [SH18] from that of Bizjak and Møgelberg [BM20] by some kind of image factorization. Indeed, first we define $K : \mathbf{Set_{B_L K}}$ to be the generic parameterizing object:

$$
\begin{array}{ccc}
(\mathbf{B_L K})_K & \longrightarrow & \mathbb{A}^\bullet \\
\vdots & & \downarrow \\
\mathbf{B_L K} & \xrightarrow{\ \ p\ \ } & \mathbb{A}
\end{array}
$$

Then the topos of Sterling and Harper [SH18] is obtained from the image factorization of the left-hand étale map:

$$
\begin{array}{ccc}
(\mathbf{B_L K})_K & \xrightarrow{\ \text{étale map}\ } & \mathbf{B_L K} \\
 & {}_{\text{étale cover}}\searrow \quad \nearrow_{\text{open immersion}} & \\
 & (\mathbf{B_L K})_{\|K\|} &
\end{array}
$$

§2.3. AFFINE & QUASI-AFFINE TOPOI

※ **(2.3∗1)** Many of the examples of topoi that we have seen so far can be presented by logoi of the form $\Pr(\mathcal{C})$ — such a topos is called *quasi-affine*.[6] A more specific but very

---

[6] Here we adopt the terminology of Anel and Lejay [AL20].

important subtype is the *affine* topoi; an *affine topos* is the dual of a logos that is generated from a category by freely adding all colimits and finite limits (a "free logos"), whereas a *quasi-affine topos* is the dual of a "quasi-free logos", which is a logos that is generated from a category by freely adding all colimits but retaining whatever limits already exist; quasi-affine topoi are sometimes referred to as "algebraic topoi" by analogy with domain theory, where ideal completion plays a similar role to the presheaf construction.

§2.3.1. Free cocompletion, Diaconescu's theorem, and quasi-affine topoi

This exposition draws on joint work with Daniel Gratzer.

(2.3.1∗1) We already noted in **(2.0∗14)** that the presheaf construction can be viewed as a *free cocompletion*, *i.e.* freely adding all colimits to a small category. Unfolding the universal property, this means that arbitrary functors $\mathcal{C} \longrightarrow \mathcal{E}$ for a cocomplete category $\mathcal{E}$ correspond to cocontinuous functors $\mathrm{Pr}(\mathcal{C}) \longrightarrow \mathcal{E}$. When $\mathcal{E}$ is additionally a logos, it is very important to understand what kind of functors $\mathcal{C} \longrightarrow \mathcal{E}$ correspond to morphisms of logoi $\mathrm{Pr}(\mathcal{C}) \longrightarrow \mathcal{E}$. Stated more concisely, we have so far understood presheaves as a free construction that takes a category to a cocomplete category; we now wish to understand presheaves as a free construction that takes a category to a logos. This is the purpose of *Diaconescu's theorem* [Dia75], one of the most useful theorems of category theory.

(2.3.1∗2) A functor $\mathrm{F} : \mathcal{C} \longrightarrow \mathcal{E}$ from a small category $\mathcal{C}$ to a logos $\mathcal{E}$ is called *flat*[7] when it satisfies the following condition for each finite diagram $\mathrm{D}_\bullet : \mathrm{I} \longrightarrow \mathcal{C}$. Let $\mathsf{Cone}(\mathrm{D}_\bullet) := \{(\mathrm{C}, \pi_\bullet) \mid \pi_\bullet : \{\mathrm{C}\} \longrightarrow \mathrm{D}_\bullet\}$ be the collection of cones in $\mathcal{C}$ over $\mathrm{D}_\bullet$. For each cone $(\mathrm{C}, \pi_\bullet)$, there is a canonical map $\mathrm{F}(\mathrm{C}) \longrightarrow \lim_\mathrm{I} \mathrm{F}(\mathrm{D}_\bullet)$ induced by $\mathrm{F}(\pi_\bullet)$. Then, we require the following map to be an epimorphism in $\mathcal{E}$:

$$\coprod_{(\mathrm{C}, \pi_\bullet) \in \mathsf{Cone}(\mathrm{D}_\bullet)} \mathrm{F}(\mathrm{C}) \longrightarrow \lim_\mathrm{I} \mathrm{F}(\mathrm{D}_\bullet)$$

(2.3.1∗3) Let $\mathcal{E}$ be a logos and $\mathrm{F} : \mathcal{C} \longrightarrow \mathcal{E}$ be a functor from an arbitrary category $\mathcal{C}$. The universal property of the free cocompletion $\mathrm{Pr}(\mathcal{C})$ yields an essentially unique cocontinuous *Yoneda extension* $\hat{\mathrm{F}} : \mathrm{Pr}(\mathcal{C}) \longrightarrow \mathcal{E}$ factoring through F:

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{\ \mathrm{F}\ } & \mathcal{E} \\ {\scriptstyle y_\mathcal{C}} \big\downarrow & \nearrow{\scriptstyle \hat{\mathrm{F}}} & \\ \mathrm{Pr}(\mathcal{C}) & & \end{array}$$

If the extension $\hat{\mathrm{F}}$ is additionally left exact, then it is tautologically a morphism of logoi; the content of Diaconescu's theorem **(2.3.1∗5)** is to connect these morphisms of logoi with **(2.3.1∗2)**, expressing the universal property of presheaf logoi.

---

[7] Some others, *e.g.* Mac Lane and Moerdijk [MM92], refer to this as *internal flatness*; Johnstone [Joh02] refers to such functors as *torsors* on $\mathcal{C}^{\mathrm{op}}$.

▪ **(2.3.1∗4)** When $\mathcal{C}$ is finitely complete and $\mathcal{E}$ is a logos, a functor $\mathcal{C} \longrightarrow \mathcal{E}$ is flat if and only if it is left exact.

Proof. Let $F : \mathcal{C} \longrightarrow \mathcal{E}$ be flat; then the Yoneda extension $\hat{F} : \mathrm{Pr}(\mathcal{C}) \longrightarrow \mathcal{E}$ is left exact, but so is F because $F = \hat{F} \circ y_{\mathcal{C}}$ and $y_{\mathcal{C}}$ is continuous. Conversely, supposing that $F : \mathcal{C} \longrightarrow \mathcal{E}$ is left exact, we must argue that F is flat; fixing a finite diagram $D_\bullet : I \longrightarrow \mathcal{C}$, we must check that the canonical map $\coprod_{(C,\pi_\bullet) \in \mathsf{Cone}(D_\bullet)} F(C) \longrightarrow \lim_I F(D_\bullet)$ is an epimorphism. It suffices to exhibit a section for this map, using the left exactness of F and the actual limit cone for $D_\bullet$ in $\mathcal{C}$:

$$\lim_I F(D_\bullet) \xrightarrow{\;\cong\;} F(\lim_I D_\bullet) \xrightarrow{\;\;i\;\;} \coprod_{(C,\pi_\bullet) \in \mathsf{Cone}(D_\bullet)} F(c)$$

with morphism *id* from $\lim_I F(D_\bullet)$ and r to $\lim_I F(D_\bullet)$. □

▪ **(2.3.1∗5)** *Diaconescu's theorem [Dia75].* Morphisms of logoi out of $\mathrm{Pr}(\mathcal{C})$ correspond to *flat* functors out of $\mathcal{C}$, in the sense that the equivalence $\mathrm{Hom}(\mathrm{Pr}(\mathcal{C}),\mathcal{E})_{\mathsf{cc}} = \mathrm{Hom}(\mathcal{C},\mathcal{E})$ restricts to an equivalence $\mathrm{Hom}_{\mathbf{Logos}}(\mathrm{Pr}(\mathcal{C}),\mathcal{E}) = \mathrm{Hom}(\mathcal{C},\mathcal{E})_{\mathsf{flat}}$. In particular, if $\mathcal{C}$ is left exact then morphisms of logoi out of $\mathrm{Pr}(\mathcal{C})$ correspond to left exact functors out of $\mathcal{C}$ because of **(2.3.1∗4)**.

▪ **(2.3.1∗6)** *Quasi-free logos, quasi-affine topos.* A logos (equivalent to one) of the form $\mathrm{Pr}(\mathcal{C})$ for a small category $\mathcal{C}$ is called *quasi-free*. A topos $\mathbf{X}$ such that $\mathbf{Set_X}$ is quasi-free (*i.e.* $\mathbf{X} \simeq \hat{\mathcal{C}}$) is called *quasi-affine*.

☯ **(2.3.1∗7)** Rephrasing Diaconescu's theorem **(2.3.1∗5)** into the language of classifying topoi, we see that a quasi-affine topos $\mathbf{X} \simeq \hat{\mathcal{C}}$ classifies the geometric theory of flat functors out of $\mathcal{C}$. Hence if $\mathcal{C}$ is a left exact category (a finite limit theory) a morphism $\mathbf{Y} \longrightarrow \hat{\mathcal{C}}$ corresponds to a left exact functor $\mathcal{C} \longrightarrow \mathbf{Set_Y}$, *i.e.* a model of the finite limit theory $\mathcal{C}$. In other words, the classifying topos of a finite limit theory $\mathcal{C}$ is the quasi-affine topos $\hat{\mathcal{C}}$ generated by that theory.

§2.3.2. Affine topoi: classifiers of diagrams

▪ **(2.3.2∗1)** *Free logoi.* A free logos is one obtained from a category $\mathcal{C}$ by freely adding all finite limits and then all colimits. The free finite limit completion $\mathcal{C}^{lex}$ of $\mathcal{C}$ can be obtained by taking the opposite of the full subcategory of $\mathrm{Pr}(\mathcal{C}^{\mathrm{op}})$ spanned by *finite colimits of representables*; then all colimits are added to $\mathcal{C}^{lex}$ by taking presheaves, yielding the *free logos* $\mathbf{Set}[\mathcal{C}] := \mathrm{Pr}(\mathcal{C}^{lex})$.

(2.3.2∗2) Taking the free logos on a category $\mathcal{C}$ can be thought of in the following way: one is *freely adding* a diagram of shape $\mathcal{C}$ to the category of sets. An interesting aspect of this characterization is that $\mathbf{Set}[\mathcal{C}]$ does not need to behave like the category of sets — for instance, the law of the excluded middle and the axiom of choice may fail in a free logos.

(2.3.2∗3) *Affine topoi.* An affine topos is a topos $\mathbf{X}$ such that $\mathbf{Set_X}$ is free, *i.e.* there is some small category $\mathcal{C}$ such that $\mathbf{Set_X} \simeq \mathbf{Set}[\mathcal{C}]$.

(2.3.2∗4) In fact, we have already seen our first example of an affine topos, the topos of sets $\mathbb{A}$ from §2.2.1; we will show that $\mathbf{Set}_\mathbb{A}$ is $\mathbf{Set}[\mathbf{1}]$, the free logos on a single generator. This computation will also give us a better understanding of the connection between $\mathbf{Set}_\mathbb{A}$ and variable binding broached in **(2.2.1∗3)**.

First we observe that the free finite limit completion $\mathbf{1}^{lex}$ of the terminal category $\mathbf{1}$ is opposite of the free finite colimit completion $\mathbf{1}^{rex}$ of the same. Generating a finitely cocomplete category from a single object produces the category of finite sets; hence $\mathbf{1}^{lex}$ is $\mathbf{FinSet}^{op}$; hence the free logos $\mathbf{Set}[\mathbf{1}]$ can be computed as the category of presheaves $\mathrm{Pr}(\mathbf{FinSet}^{op})$. The connection to variable binding is as follows: an object of $\mathbf{FinSet}^{op}$ is essentially a context of (untyped) variables, and a morphism in $\mathbf{FinSet}^{op}$ is a substitution! Hence an object of $\mathbf{Set}[\mathbf{1}]$ is a family of sets that varies functorially in contexts and substitutions.

We now check that the free logos $\mathbf{Set}[\mathbf{1}]$ really is the theory of a set by computing relative to an arbitrary logos $\mathcal{E}$:

$$
\begin{aligned}
&\mathrm{Hom}_{\mathbf{Logos}}(\mathbf{Set}[\mathbf{1}],\mathcal{E}) \\
&\quad = \mathrm{Hom}_{\mathbf{Logos}}(\mathrm{Pr}(\mathbf{1}^{lex}),\mathcal{E}) && \text{by definition} \\
&\quad \cong \mathrm{Hom}_{\mathbf{Cat}_{lex}}(\mathbf{1}^{lex},\mathcal{E}) && \text{by Diaconescu's theorem} \\
&\quad \cong \mathrm{Hom}_{\mathbf{Cat}}(\mathbf{1},\mathcal{E}) && \text{by universal property of } -^{lex}
\end{aligned}
$$

Hence morphisms of logoi from $\mathbf{Set}[\mathbf{1}]$ to $\mathcal{E}$ really do correspond to objects of $\mathcal{E}$, so we see that $\mathbf{Set}_\mathbb{A} = \mathbf{Set}[\mathbf{1}]$.

(2.3.2∗5) By analogy with classifier of sets $\mathbb{A}$, we will write $\mathbb{A}^\mathcal{C}$ for the affine topos generated by a category $\mathcal{C}$, *i.e.* we set $\mathbf{Set}_{\mathbb{A}^\mathcal{C}} \coloneqq \mathbf{Set}[\mathcal{C}]$; then we have $\mathbb{A} = \mathbb{A}^\mathbf{1}$, which we will sometimes refer to as "the" affine topos.

(2.3.2∗6) The computation **(2.3.2∗4)** also shows that $\mathbb{A}^\mathcal{C}$ classifies *diagrams* of shape $\mathcal{C}$, by generalizing from $\mathbf{1}$ to $\mathcal{C}$. In other words, a morphism of topoi $\mathbf{X} \longrightarrow \mathbb{A}^\mathcal{C}$ is the same as an arbitrary diagram $\mathcal{C} \longrightarrow \mathbf{Set_X}$.

§2.4. ARTIN GLUING OF OPEN AND CLOSED SUBTOPOI

(2.4∗1) Given an open subset U of a topological space X, we may always reconstruct X by gluing U together with its complement $X_{\backslash U}$. The observation of the Grothendieck school in the early 1960s is that this reconstruction theorem also applies to topoi [AGV72]; it was not until later that the connection to Tait's method of computability was observed by Freyd [Fre78].

(2.4∗2) Let U be an open of a topos $\mathbf{X}$; the *closed complement* of U will be written $\mathbf{X}_{\backslash U}$. We define $\mathbf{X}_{\backslash U}$ by setting $\mathbf{Set}_{\mathbf{X}_{\backslash U}}$ to be the full subcategory of $\mathbf{Set}_{\mathbf{X}}$ spanned by sheaves E that are $(-)^U$-*connected*, in the sense that the canonical map $E^U \longrightarrow 1_{\mathbf{Set}_{\mathbf{X}}}$ is an isomorphism. The inclusion $\mathbf{Set}_{\mathbf{X}_{\backslash U}} \hookrightarrow \mathbf{Set}_{\mathbf{X}}$ is the direct image part of an embedding of topoi $\mathbf{X}_{\backslash U} \hookrightarrow \mathbf{X}$, called a *closed immersion*.

(2.4∗3) The inverse image part of the closed immersion $i : \mathbf{X}_{\backslash U} \hookrightarrow \mathbf{X}$ sends a sheaf $E : \mathbf{Set}_{\mathbf{X}}$ to the following pushout:

$$
\begin{array}{ccc}
E \times U & \longrightarrow & U \\
\downarrow & & \vdots \\
E & \dashrightarrow & i^*E
\end{array}
$$

(2.4∗4) Let $i : \mathbf{X}_{\backslash U} \hookrightarrow \mathbf{X}$ be a closed immersion; based on the computation **(2.4∗3)**, show that for any sheaf $E : \mathbf{Set}_{\mathbf{X}}$ the inverse image i^*E is $(-)^U$-connected in the sense of **(2.4∗2)**.

(2.4∗5) *Artin gluing.* We may reconstruct $\mathbf{X}$ from the complementary open and closed subtopoi corresponding to an open U. We may construct a "boundary" or "fringe" functor $\rho : \mathbf{Set}_{\mathbf{X}_U} \longrightarrow \mathbf{Set}_{\mathbf{X}_{\backslash U}}$ from the direct image of the open immersion followed by the inverse image of the closed immersion:

$$
\begin{array}{ccc}
\mathbf{Set}_{\mathbf{X}_U} & \xrightarrow{\quad \rho \quad} & \mathbf{Set}_{\mathbf{X}_{\backslash U}} \\
& \mathbf{Set}_{\mathbf{X}} &
\end{array}
$$

We immediately observe that the functor ρ is left exact, but it is not necessarily the inverse image part nor the direct image part of a morphism of topoi. In most instances of gluing that bear on the metatheory of type theory, ρ will in fact be involved in a morphism of topoi, in which case some more geometrical constructions become available as we will see in §4.3. We may form a new logos by *gluing* $\mathbf{Set}_{\mathbf{X}_U}$ and $\mathbf{Set}_{\mathbf{X}_{\backslash U}}$ along ρ by taking the

following pullback of categories, $\mathcal{E} = \mathbf{Set_{X_{\setminus U}}} \downarrow \rho$:

$$
\begin{array}{ccc}
\mathcal{E} & \longrightarrow & \mathbf{Set_{X_{\setminus U}}}^{\rightarrow} \\
\downarrow & \lrcorner & \downarrow \text{cod} \\
\mathbf{Set_{X_U}} & \xrightarrow{\quad\rho\quad} & \mathbf{Set_{X_{\setminus U}}}
\end{array}
$$

The gluing theorem of SGA 4 [AGV72] states that $\mathcal{E}$ is in fact a logos that is equivalent to $\mathbf{Set_X}$, and under this identification, $\mathcal{E} \longrightarrow \mathbf{Set_{X_U}}$ is the inverse image part of the open immersion $j : \mathbf{X_U} \hookrightarrow \mathbf{X}$.

(2.4∗6) Conversely let $\mathbf{U}$ and $\mathbf{Y}$ be topoi and suppose that $\rho : \mathbf{Set_U} \longrightarrow \mathbf{Set_Y}$ is a left exact morphism. Form the Artin gluing of ρ:

$$
\begin{array}{ccc}
\mathcal{E} & \longrightarrow & \mathbf{Set_Y}^{\rightarrow} \\
\downarrow & \lrcorner & \downarrow \text{cod} \\
\mathbf{Set_U} & \xrightarrow{\quad\rho\quad} & \mathbf{Set_Y}
\end{array}
$$

Letting $\mathbf{X}$ be the topos such that $\mathbf{Set_X} = \mathcal{E}$, we may find an open U of $\mathbf{X}$ such that $\mathbf{U} \simeq \mathbf{X_U}$ and $\mathbf{Y} \simeq \mathbf{X}_{\setminus U}$. Recalling the computation $\mathcal{E} = \mathbf{Set_Y} \downarrow \rho$ from **(2.4∗5)**, we define U by the morphism $\mathbf{0_{Set_Y}} \rightarrowtail \rho(\mathbf{1_{Set_U}})$ which is clearly subterminal in $\mathcal{E}$.

§2.5. TINY OBJECTS IN LOGOI

(2.5∗1) Let $\mathcal{E}$ be a logos and let $E : \mathcal{E}$. The sheaf E is called *tiny* when the exponential functor $(-)^E : \mathcal{E} \longrightarrow \mathcal{E}$ has a right adjoint $(-)_E$ [Yet87]. By the adjoint functor theorem, this is equivalent to $(-)^E$ preserving colimits.

(2.5∗2) *Tininess of representables.* Consider a quasi-free logos $\mathcal{E} = \mathrm{Pr}(\mathcal{C})$, and let $C : \mathcal{C}$ be an object. If $\mathcal{C}$ has products of the form $- \times C$, then $y_{\mathcal{C}}(C)$ is tiny in $\mathcal{E}$.

Proof. We must check that the exponential functor $(y_{\mathcal{C}}(C) \to -)$ preserves colimits; we fix a diagram $E_{\bullet} : I \longrightarrow \mathcal{E}$ and compute the representable points of the exponential $(y_{\mathcal{C}}(C) \to \mathrm{colim}_I E_{\bullet})$, using the fact that colimits of presheaves are computed pointwise:

$$
\begin{aligned}
\mathrm{Hom}_{\mathcal{E}}&\left(y_{\mathcal{C}}(D), y_{\mathcal{C}}(C) \to \mathrm{colim}_I E_{\bullet}\right) \\
&\cong \mathrm{Hom}_{\mathcal{E}}\left(y_{\mathcal{C}}(D) \times y_{\mathcal{C}}(C), \mathrm{colim}_I E_{\bullet}\right) \\
&\cong \mathrm{Hom}_{\mathcal{E}}\left(y_{\mathcal{C}}(D \times C), \mathrm{colim}_I E_{\bullet}\right)
\end{aligned}
$$

$$\cong \mathrm{colim}_I \mathrm{Hom}_{\mathcal{E}}(y_C(D \times C), E_\bullet)$$

$$\cong \mathrm{colim}_I \mathrm{Hom}_{\mathcal{E}}(y_C(D) \times y_C(C), E_\bullet)$$

$$\cong \mathrm{colim}_I \mathrm{Hom}_{\mathcal{E}}(y_C(D), y_C(C) \to E_\bullet)$$

$$\cong \mathrm{Hom}_{\mathcal{E}}(y_C(D), \mathrm{colim}_I y_C(C) \to E_\bullet) \qquad \square$$

(2.5∗3) *Gluing of tiny objects.* Let $f : \mathbf{Y} \longrightarrow \mathbf{U}$ be a morphism of topoi. Write $\mathbf{X}$ for the Artin gluing of the inverse image functor f^*, and write $j : \mathbf{U} \hookrightarrow \mathbf{X}$ and $i : \mathbf{Y} \hookrightarrow \mathbf{X}$ for the respective open and closed immersions of topoi. Suppose that $X : \mathbf{Set_X}$ is a sheaf such that j^*X is a tiny object in $\mathbf{Set_U}$ and i^*X is a tiny object in $\mathbf{Set_Y}$; then X is tiny.

Proof. It suffices to check that the exponential functor $(X \to -)$ preserves colimits. Fixing a diagram $E_\bullet : I \longrightarrow \mathbf{Set_X}$, we may compute the exponential $(X \to \mathrm{colim}_I E_\bullet)$ in the language of $\mathbf{Set_Y}$ as follows; first, the standard computation that glues a function from the open subtopos onto a function from the closed subtopos [Joh02]:

$$
\begin{array}{ccc}
i^*(X \to \mathrm{colim}_I E_\bullet) & \longrightarrow & i^*X \to i^* \mathrm{colim}_I E_\bullet \\
\Big\downarrow & & \Big\downarrow \\
f^*(j^*X \to j^* \mathrm{colim}_I E_\bullet) & \longrightarrow & i^*X \to f^*j^* \mathrm{colim}_I E_\bullet
\end{array}
$$

Commute cocontinuous functors past colimits.

$$
\begin{array}{ccc}
i^*(X \to \mathrm{colim}_I E_\bullet) & \longrightarrow & i^*X \to \mathrm{colim}_I i^*E_\bullet \\
\Big\downarrow & & \Big\downarrow \\
f^*(j^*X \to \mathrm{colim}_I j^*E_\bullet) & \longrightarrow & i^*X \to \mathrm{colim}_I f^*j^*E_\bullet
\end{array}
$$

Use the tininess of i^*X, j^*X and the cocontinuity of f^*.

$$
\begin{array}{ccc}
i^*(X \to \mathrm{colim}_I E_\bullet) & \longrightarrow & \mathrm{colim}_I (i^*X \to i^*E_\bullet) \\
\Big\downarrow & & \Big\downarrow \\
\mathrm{colim}_I f^*(j^*X \to j^*E_\bullet) & \longrightarrow & \mathrm{colim}_I (i^*X \to f^*j^*E_\bullet)
\end{array}
$$

Hence by the universality of colimits we have:

$$
\begin{array}{ccc}
\mathrm{colim}_I i^*(X \to E_\bullet) & \longrightarrow & \mathrm{colim}_I (i^*X \to i^*E_\bullet) \\
\Big\downarrow & & \Big\downarrow \\
\mathrm{colim}_I f^*(j^*X \to j^*E_\bullet) & \longrightarrow & \mathrm{colim}_I (i^*X \to f^*j^*E_\bullet) \qquad \square
\end{array}
$$

CHAPTER 3

THE THEORY OF UNIVERSES

※ **(3.0∗1)** In this chapter we recall the concept of universes, and expose the notion of a *strong universe* which has played an important role in several recent works [Awo21; Bir+16; GSS21; KL21; OP16; Shu15a; Str14a].

§3.1. UNIVERSES IN SET THEORY

※ **(3.1∗1)** Assuming the existence of enough strongly inaccessible cardinals, the category of sets is closed under a transfinite and cumulative hierarchy of universes $\mathcal{U} \subseteq \mathcal{V} \subseteq \ldots$; a universe is a "set of sets" that is closed under all connectives (including dependent products, dependent sums, coproducts, quotients, equality types, *etc.*).

⑂ **(3.1∗2)** Let κ be a cardinal number; we will write $\mathbf{Set}_\kappa$ for the full subcategory of **Set** spanned by sets A such that $|A| < \kappa$. Such sets are called κ-small.

■ **(3.1∗3)** A cardinal number κ is called *strongly inaccessible* when $\mathbf{Set}_\kappa$ is closed under dependent products.

◆ **(3.1∗4)** Most strongly inaccessible cardinals are infinite; but 2 is also a strongly inaccessible cardinal!

⩔ **(3.1∗5)** Explicitly, the force of **(3.1∗3)** is the following: if X has cardinality less than κ and $\{Y_x\}_{x \in X}$ is a family of sets that each have cardinality less than κ, then $\prod_{x \in X} Y_x$ has cardinality less than κ.

■ **(3.1∗6)** Each strongly inaccessible cardinal κ gives rise to a set of sets $\mathcal{U}_\kappa$ called a *Grothendieck universe* [AGV72]; the set $\mathcal{U}_\kappa$ is the set of objects of a category of small sets equivalent (but not isomorphic) to $\mathbf{Set}_\kappa$. Therefore $\mathcal{U}_\kappa$ is closed under all all set/type theoretical notions, including dependent product/sum, κ-small (co)limits, powersets, *etc.*

⧯ **(3.1∗7)** Explicitly, $\mathcal{U}_\kappa$ can be defined to be the set of sets with rank less than κ; then $\mathcal{U}_\kappa$ has both rank and cardinality κ. The *rank* of a set is defined to be the smallest ordinal number strictly bounding the ranks of all the members of that sets; the notion is well-defined because the membership relation is well-founded.

※ **(3.1∗8)** Once we have set up the universe hierarchy, it is better to work with the resulting universes than with the cardinals they are defined from. However, the fact that each universe is a restriction of the collection of sets by a cardinality bound leads to some important properties that we will exploit.

■ **(3.1∗9)** *Cumulativity.* Let $\lambda < \kappa$ be strongly inaccessible cardinals; then $\mathcal{U}_\lambda \subseteq \mathcal{U}_\kappa$ and moreover $\mathcal{U}_\lambda \in \mathcal{U}_\kappa$.

■ **(3.1∗10)** If $\mathcal{U}$ is a universe in **Set**, then consider the family $\dot{\mathcal{U}} \longrightarrow \mathcal{U}$ where we write $\dot{\mathcal{U}}$ for the collection $\sum_{A \in \mathcal{U}} A$. A family of sets $Y \longrightarrow X$ is called $\mathcal{U}$-small if and only if there exists a (not necessarily unique) pullback square in the following configuration:

$$
\begin{array}{ccc}
Y & \dashrightarrow & \dot{\mathcal{U}} \\
\downarrow & \lrcorner & \downarrow \\
X & \dashrightarrow & \mathcal{U}
\end{array}
$$

A set X is called $\mathcal{U}$-small when the terminal family $X \longrightarrow \mathbf{1}_{\mathbf{Set}}$ is $\mathcal{U}$-small. The family $\dot{\mathcal{U}} \longrightarrow \mathcal{U}$ is called the "generic family" of the universe.

§3.2. UNIVERSES IN CATEGORIES

※ **(3.2∗1)** The notion of a universe makes sense in any category $\mathcal{E}$: we might define a universe to be simply an arbitrary morphism $p : \dot{\mathcal{U}} \longrightarrow \mathcal{U}$ such that the following pullback exists for any morphism $A : X \longrightarrow \mathcal{U}$:

$$
\begin{array}{ccc}
[A] & \xdashrightarrow{q_A} & \dot{\mathcal{U}} \\
p_A \downarrow & \lrcorner & \downarrow p \\
X & \xrightarrow{A} & \mathcal{U}
\end{array}
$$

A map that arises as p_A for some $A : X \longrightarrow \mathcal{U}$ is called $\mathcal{U}$-small. Then, it is possible to speak of the universe $\dot{\mathcal{U}} \longrightarrow \mathcal{U}$ being closed under various connectives.

For the sake of simplicity, we will assume that $\mathcal{E}$ itself is locally Cartesian closed.

(3.2∗2) When $A : \mathcal{U}$, we will write $[A]$ for the fiber of $\dot{\mathcal{U}}$ over A:

$$
\begin{array}{ccc}
[A] & \longrightarrow & \dot{\mathcal{U}} \\
\downarrow & \lrcorner & \downarrow \\
X & \longrightarrow & \mathcal{U} \\
 & A &
\end{array}
$$

(3.2∗3) *Dependent sums.* The universe $\dot{\mathcal{U}} \longrightarrow \mathcal{U}$ is closed under dependent sums when the following projection morphism is $\mathcal{U}$-small:

$$\left(\textstyle\sum_{A:\mathcal{U}}\sum_{B:[A]\to\mathcal{U}}\sum_{a:[A]}[B(a)]\right) \longrightarrow \sum_{A:\mathcal{U}}([A] \to \mathcal{U})$$

(3.2∗4) *Dependent products.* The universe $\dot{\mathcal{U}} \longrightarrow \mathcal{U}$ is closed under dependent products when the following projection morphism is $\mathcal{U}$-small:

$$\left(\textstyle\sum_{A:\mathcal{U}}([A] \to \dot{\mathcal{U}})\right) \longrightarrow \sum_{A:\mathcal{U}}([A] \to \mathcal{U})$$

(3.2∗5) What **(3.2∗3)** and **(3.2∗4)** have in common is that the closure of a universe under a connective is stated by requiring that a certain morphism be small; this morphism is the "generic" dependent sum/dependent product/*etc.* situation. From a type theoretic perspective, the domain of the morphism is the data assumed in its *introduction rule* and the codomain is the data assumed in its *formation rule*.

§3.3. STRONG UNIVERSES AND REALIGNMENT

(3.3∗1) *Realignment (externally).* A property of central importance for a universe $p : \dot{\mathcal{U}} \longrightarrow \mathcal{U}$ is to support *realignment* along monomorphisms. We will write $\mathcal{S}$ for the collection of morphisms that arise by pullback from $p : \dot{\mathcal{U}} \longrightarrow \mathcal{U}$. Let $\mathcal{M}$ be a set of monomorphisms in $\mathcal{E}$; we say that $p : \dot{\mathcal{U}} \longrightarrow \mathcal{U}$ supports $\mathcal{M}$-realignment when any lifting problem of the following kind admits a solution in the cartesian arrow category $\mathcal{E}^{\rightarrow}_{cart}$, assuming that $f \in \mathcal{S}$ and $h \longrightarrow f$ lies horizontally over an element of $\mathcal{M}$:

$$
\begin{array}{ccc}
h & \xrightarrow{\text{cart.}} & \pi \\
{\scriptstyle\text{cart.}}\downarrow & \nearrow & \\
f & &
\end{array}
$$

❦ **(3.3∗2)** *History.* The external realignment principle above has played an important role in almost every work on the semantics of homotopy type theory, notably in Voevodsky's simplicial set semantics of homotopy type theory [KL21] as well as that of Streicher [Str14a]. Shulman [Shu15a] stated the principle more abstractly,[1] contributing a construction of universes of presheaves satisfying realignment that relies only on the exactness properties of Grothendieck topoi; this result is replayed in generality for an arbitrary Grothendieck topos by Gratzer, Shulman, and Sterling [GSS21]. The external realignment property also appears in a disguised form in the proof of canonicity for XTT [SAG20], a version of cubical type theory with proof-irrelevant equality types.

An *internal* version of the realignment property, detailed below, has been employed in the semantics of cubical type theory, first suggested by Coquand and then adapted to the setting of guarded cubical sets by Birkedal, Bizjak, Clouston, Grathwohl, Spitters, and Vezzosi [Bir+16], and employed more generally by Orton and Pitts [OP16]. When $\mathcal{M}$ is represented by a subobject $\mathbb{P} \subseteq \Omega$ of the subobject classifier, the presence of internal realignment structure is implied by the external realignment property. Internal realignment is the workhorse lemma of the synthetic Tait computability presented in this dissertation, in essence because it allows one to align a computability structure *strictly* over a given component of the syntactic algebra.

◼ **(3.3∗3)** *Internal realignment.* The internal version of realignment can be stated in the type theoretic language of a logos $\mathcal{E}$. Write $A : \mathcal{U} \vdash \mathsf{Iso}(A)$ for the collection of $\mathcal{U}$-small isomorphs of A defined as follows:

$$\mathsf{Iso}(A) := \textstyle\sum_{B:\mathcal{U}}([B] \cong [A])$$

Let $\mathbb{P} \subseteq \Omega$ be a subobject of the subobject classifier; then a *realignment structure* for $\mathcal{U}$ with respect to $\mathbb{P}$ is an element of the following type:

$$\mathsf{realign}_{\mathcal{U}} : \textstyle\prod_{A:\mathcal{U}}\prod_{\phi:\mathbb{P}}\prod_{B:[\phi]\to\mathsf{Iso}(A)}\{A' : \mathsf{Iso}(A) \mid \forall z : [\phi].A' = B(z)\}$$

◼ **(3.3∗4)** *Strong universe.* A universe $\mathcal{U}$ equipped with such a realignment structure is called $\mathbb{P}$-*strong*; when $\mathbb{P}$ is Ω itself, we simply refer to $\mathcal{U}$ as strong.

◼ **(3.3∗5)** *Internalizing.* If $\mathcal{U}$ satisfies the external realignment property **(3.3∗1)** relative to the collection of monomorphisms classified by $\mathbb{P}$, then $\mathcal{U}$ is $\mathbb{P}$-strong.

Proof. The *generic* internal realignment situation can be expressed as an external realignment problem, as shown in detail by Gratzer, Shulman, and Sterling [GSS21]. □

---

[1] Referred to as *Condition (2')* by Shulman [Shu15a].

(3.3∗6) *Realignment connective.* We will use the following type theoretic notation for applications of the realignment structure of a given strong universe $\mathcal{U}$:

$$\frac{\mathrm{A} : \mathcal{U} \qquad \phi : \mathbb{P} \qquad \mathrm{B} : [\phi] \to \mathsf{Iso}(\mathrm{A})}{\mathsf{realign}\ [\mathrm{A} \mid \phi \hookrightarrow \mathrm{B}] : \mathcal{U} \qquad z : [\phi] \vdash \mathsf{realign}\ [\mathrm{A} \mid \phi \hookrightarrow \mathrm{B}] = \pi_1(\mathrm{B}(z)) : \mathcal{U}}$$

$$\frac{\mathrm{A} : \mathcal{U} \qquad \phi : \mathbb{P} \qquad \mathrm{B} : [\phi] \to \mathsf{Iso}(\mathrm{A}) \qquad a : [\mathrm{A}]}{\lfloor a \rfloor : [\mathsf{realign}\ [\mathrm{A} \mid \phi \hookrightarrow \mathrm{B}]] \qquad z : [\phi] \vdash \lfloor a \rfloor = \pi_2(\mathrm{B}(z))^{-1}(a) : [\pi_1(\mathrm{B}(z))]}$$

$$\frac{\mathrm{A} : \mathcal{U} \qquad \phi : \mathbb{P} \qquad \mathrm{B} : [\phi] \to \mathsf{Iso}(\mathrm{A}) \qquad a : [\mathsf{realign}\ [\mathrm{A} \mid \phi \hookrightarrow \mathrm{B}]]}{\lceil a \rceil : [\mathrm{A}] \qquad z : [\phi] \vdash \lceil a \rceil = \pi_2(\mathrm{B}(z))(a) : [\mathrm{A}]}$$

§3.4. UNIVERSE HIERARCHIES IN LOGOI

(3.4∗1) The hierarchy of strongly inaccessible cardinals in the ambient set theory gives rise to a cumulative hierarchy of universes of sets (§3.1) that has extremely strict properties. In this section, we axiomatize some of the properties of this universe hierarchy for use elsewhere; such hierarchies exist at least in all Grothendieck topoi.

(3.4∗2) In this section, we assume that $\mathcal{E}$ is a logos; let $\mathbf{L}$ be the total order of the natural numbers with an additional top element $\diamond : \mathbf{L}$ adjoined. We intend $\mathbf{L}$ to be the index of a hierarchy of universes; our arguments also work when $\mathbf{L}$ contains additional limit ordinals, but we will not need this.

(3.4∗3) *Universe systems.* An $\mathbf{L}$-shaped *universe system* is given by the following data:

1) A diagram $\mathcal{U}_\bullet : \mathbf{L} \longrightarrow \mathcal{E}$ where all arrows $\langle \uparrow_\alpha^\beta \rangle : \mathcal{U}_\alpha \rightarrowtail \mathcal{U}_\beta$ are monomorphisms, together with a family $\dot{\mathcal{U}} : \mathcal{E}_{/\mathcal{U}_\diamond}$. We will write $\dot{\mathcal{U}}_\alpha$ for the restriction of $\dot{\mathcal{U}}$ along $\langle \uparrow_\alpha^\beta \rangle$:

$$
\begin{array}{ccc}
\dot{\mathcal{U}}_\alpha & \dashrightarrow & \dot{\mathcal{U}} \\
\downarrow & \lrcorner & \downarrow \\
\mathcal{U}_\alpha & \underset{\langle \uparrow_\alpha^\diamond \rangle}{\rightarrowtail} & \mathcal{U}_\diamond
\end{array}
$$

2) For each object $\alpha < \diamond$, an element $\mathsf{U}_\alpha : \mathbf{1}_\mathcal{E} \longrightarrow \mathcal{U}_{\mathsf{s}(\alpha)}$ together with a cartesian square in the following configuration:

$$
\begin{array}{ccccc}
\mathcal{U}_\alpha & \longrightarrow & \dot{\mathcal{U}}_{\mathsf{s}(\alpha)} & \longrightarrow & \dot{\mathcal{U}} \\
\downarrow & & \downarrow & \lrcorner & \downarrow \\
\mathbf{1}_\mathcal{E} & \underset{\mathsf{U}_\alpha}{\longrightarrow} & \mathcal{U}_{\mathsf{s}(\alpha)} & \underset{\langle \uparrow_\alpha^\diamond \rangle}{\rightarrowtail} & \mathcal{U}_\diamond
\end{array}
$$

◼ **(3.4∗4)** *Strong universe hierarchies.* We say that a universe system is strong when each $\mathcal{U}_\alpha$ is strong in the sense of **(3.3∗4)**.

◼ **(3.4∗5)** *Coherent codes for connectives.* Let $(\mathcal{U}_\bullet, \dot{\mathcal{U}}, \mathsf{U}_\bullet)$ be a universe system such that each universe $\mathcal{U}_\alpha$ has dependent product types in the sense of **(3.2∗4)**. It is not *a priori* the case that the Π_α codes at each universe level are coherent with the lifting coercions, *i.e.* we do not necessarily have $\langle\uparrow_\alpha^\beta\rangle\Pi_\alpha(A,B) = \Pi_\beta(\langle\uparrow_\alpha^\beta\rangle A, \lambda x.\langle\uparrow_\alpha^\beta\rangle B(x))$ strictly (though it does hold up to isomorphism by the universal property of the dependent product). If the universe system is *strong*, however, we may choose a new sequence of codes Π_α' such that $\langle\uparrow_\alpha^\beta\rangle\Pi_\alpha'(A,B) = \Pi_\beta'(\langle\uparrow_\alpha^\beta\rangle A, \lambda x.\langle\uparrow_\alpha^\beta\rangle B(x))$.

Proof. This is a result of Shulman [Shu15a], and proceeds by induction on $\alpha : \mathbf{L}$. First we choose $\Pi_0' := \Pi_0$; in the successor and limit cases, we define $\Pi_\alpha'(A,B)$ by realigning the existing code $\Pi_\alpha(A,B)$ against the following proposition $\phi_{A,B}$:

$$\phi_{A,B} := \bigvee_{n<\alpha} \exists A',B'.A = \langle\uparrow_n^\alpha\rangle A' \wedge B = \lambda x.\langle\uparrow_n^\alpha\rangle B'(x)$$

The partial isomorph of $\Pi_\alpha(A,B)$ under $\phi_{A,B}$ is defined piecewise for each $n < \alpha$ in terms of the *unique* A',B' that lie in the image of the lift $\langle\uparrow_n^\alpha\rangle$, writing $\iota x.P(x)$ for the *definite* description operator that is justified in any logos:

$$n \mapsto \langle\uparrow_n^\alpha\rangle\Pi_n'(\iota A'.A = \langle\uparrow_n^\alpha\rangle A', \iota B'.B = \lambda x.\langle\uparrow_n^\alpha\rangle B'(x))$$

It is necessary to check that the family of partial types defined above matches on the overlap between different $n,n' < \alpha$; this follows by induction.   □

※ **(3.4∗6)** An argument analogous to **(3.4∗5)** works for *all* connectives of dependent type theory; hence, a strong universe system can be used as the basis for the cumulative-style notation where level coercions silently commute past connectives.

⬇ **(3.4∗7)** *Cumulative notation.* In *only* the case of a strong universe system, we adopt an only slight abuse of notation in which we omit the coercions $\langle\uparrow_\alpha^\beta\rangle$.

◼ **(3.4∗8)** *Existence of universes.* Assuming enough universes in the ambient set theory, *any* logos has a strong hierarchy of universes as exposed by Gratzer, Shulman, and Sterling [GSS21]; these universes may be constructed using a variant due to Shulman [Shu15a] of the small object argument. Shulman's result refutes a number of conjectures in the literature concerning the lack of strict universes in categories of sheaves [CMR17; Man16; Péd21; Xu15; XE16]. In fact, there seems to have also been confusion in the type theoretic community as to the existence of *weak* universes, a result established by Streicher in 2014.

§3.5. DIRECT IMAGE AND EXTENT TYPES

(3.5∗1) In this section we will fix notations for two important type formers that are available in the internal languages of logoi, *partial types* and *extent types*.

(3.5∗2) *Russellian notation.* When working with universes in a logos we will no longer distinguish between a code $A : \mathcal{U}$ and its decoding $[A]$. While these are formally different, leaving the decoding implicit causes no ambiguity. Given a proposition $\phi : \Omega$, we will likewise abuse notation by writing ϕ for the decoding $[\phi]$.

(3.5∗3) *Direct image types.* Let $\phi : \Omega$ be a proposition, and let $A : \phi \to \mathcal{U}$ be a partial type defined on ϕ. We may define the "direct image" of A in $\mathcal{U}$ to be the dependent product type $\prod_{z:\phi} A(z) : \mathcal{U}$. Because these direct images will play such an important role in our developments, we will impose an "implicit dependent function" notation in the style of Agda:

$$
\begin{aligned}
\prod\nolimits_{z:\phi} A(z) \quad &\text{written} \quad \{\phi\}\, A \\
\lambda z : \phi.a(z) \quad &\text{written} \quad a \\
a(z) \quad &\text{written} \quad a
\end{aligned}
$$

The above causes no ambiguity because any two proofs $u, v : \phi$ are equal.

(3.5∗4) *Partial element types.* A special case of **(3.5∗3)** is the *partial element* type $\{\phi\}\, A$ for a total type $A : \mathcal{U}$.

(3.5∗5) *Extent types.* Let $A : \mathcal{U}$ be a type, and let $a : \{\phi\}\, A$ be a partial element defined on $\phi : \Omega$. We may define the *extent* of a to be the subset of A spanned by elements that restrict under ϕ to a:

$$
\frac{A : \mathcal{U} \qquad \phi : \Omega \qquad a : \{\phi\}\, A}{\{A \mid \phi \hookrightarrow a\} := \{a' : A \mid \forall z : \phi.a' = a\}}
$$

As in **(3.5∗3)** we will render the coercion $\{A \mid \phi \hookrightarrow a\} \longrightarrow A$ implicit in our notation.

(3.5∗6) The extent types are based on the work of Riehl and Shulman [RS17], where they appeared under the name *extension types*, and on the work of Cohen, Coquand, Huber, and Mörtberg [Coh+17] and Orton and Pitts [OP16] where they appeared alongside partial element types as a notational device in the service of the syntax and semantics of cubical type theory.

§3.6. MODAL UNIVERSES OF MODAL TYPES

(3.6∗1) Fix a proposition $\phi : \Omega$; we also assume strong universes $\mathcal{U} \subseteq \mathcal{V}$.

(3.6∗2) From within $\mathbf{Set_X}$, it is useful to think of the point / terminal object $\mathbf{1}_{\mathbf{Set_X}}$ as an internal representation of the entire space $\mathbf{X}$; from this perspective, we might write $\mathbb{1}_{\mathbf{X}}$ for the corresponding "internal topos" in $\mathbf{Set_X}$. Then a universe $\mathcal{U} : \mathbf{Set_X}$ can be thought of in two ways:

1) $\mathcal{U}$ is an internal category of (small) sheaves on the point $\mathbb{1}_{\mathbf{X}}$.

2) $\mathcal{U}$ is an internal representation of the (external) category of (small) sheaves on $\mathbf{X}$.

Using these intuitions, it is geometrically profitable to view a proposition $\phi : \Omega$ as an "internal open subtopos" $\phi \hookrightarrow \mathbb{1}_{\mathbf{X}}$; it is therefore appropriate to ponder the internal closed subtopos $\backslash\phi \hookrightarrow \mathbb{1}_{\mathbf{X}}$. In the internal language of $\mathbf{Set_X}$, we may approximate both internal subtopoi by defining universes $\mathcal{U}_\phi$ and $\mathcal{U}_{\backslash\phi}$ that serve as their internal categories of small sheaves in the same sense that $\mathcal{U}$ is the internal category of small sheaves on $\mathbb{1}_{\mathbf{X}}$.

(3.6∗3) Open and closed subuniverses are developed more thoroughly in the context of homotopy type theory by Rijke, Shulman, and Spitters [RSS20]; our contribution will be to give a specific construction of these universes that is compatible with the needs of strict & non-homotopical type theory in which the univalence principle is not available.

(3.6∗4) *Open subuniverse.* We define the open subuniverse $\mathcal{U}_\phi$ to be the following realignment of $\{\phi\}\mathcal{U}$:

$$\mathcal{U}_\phi : \{\mathcal{V} \mid \phi \hookrightarrow \mathcal{U}\}$$
$$\mathcal{U}_\phi := \mathsf{realign}\ [\{\phi\}\mathcal{U} \mid \phi \hookrightarrow (\mathcal{U}, \lambda A : \mathcal{U}.\lambda\{z : \phi\}.A)]$$

The inverse image function $\phi^* : \mathcal{U} \longrightarrow \mathcal{U}_\phi$ is deduced from the weakening $\mathcal{U} \longrightarrow \{\phi\}\mathcal{U}$ via the realignment isomorphism:

$$\phi^* : \mathcal{U} \longrightarrow \mathcal{U}_\phi$$
$$\phi^*A = \lfloor \lambda\{z : \phi\}.A \rfloor$$

We could naïvely define the direct image function to take $A : \{\phi\}\mathcal{U}$ to $\{\phi\}A$, but this would not be an injection; the problem is easily solved by realignment:

$$\phi_* : \mathcal{U}_\phi \rightarrowtail \mathcal{U}$$
$$\phi_*(A) = \mathsf{realign}\ [\{\phi\}\lceil A \rceil \mid \phi \hookrightarrow (\lceil A \rceil\{\star\}, \lambda a : \lceil A \rceil\{\star\}.\lambda\{z : \phi\}.a)]$$

(3.6∗5) It is a surprising reminder of the power of realignment that is possible to define a version of the direct image function that is an injection! It is worth verifying for yourself that the direct image function $\phi_* : \mathcal{U}_\phi \rightarrowtail \mathcal{U}$ is in fact an injection.

(3.6∗6) *Closed subuniverse.* The closed subuniverse $\mathcal{U}_{\backslash\phi}$ is defined as a realignment of the extent type $\{\mathcal{U} \mid \phi \hookrightarrow \mathbf{1}\}$:

$$\mathcal{U}_{\backslash\phi} : \{\mathcal{V} \mid \phi \hookrightarrow \mathbf{1}\}$$

$$\mathcal{U}_{\backslash\phi} = \mathsf{realign}\ [\{\mathcal{U} \mid \phi \hookrightarrow \mathbf{1}\} \mid \phi \to (\mathbf{1}, \lambda\_ : \mathbf{1}.\star)]$$

The direct image function is the easy part this time:

$$\backslash\phi_* : \mathcal{U}_{\backslash\phi} \rightarrowtail \mathcal{U}$$
$$\backslash\phi_*(\mathrm{A}) = \lceil \mathrm{A} \rceil$$

The inverse image function $\backslash\phi^* : \mathcal{U} \longrightarrow \mathcal{U}_{\backslash\phi}$ takes a type $\mathrm{A} : \mathcal{U}$ to a realignment of the *join* of A with ϕ:

$$\backslash\phi^* : \mathcal{U} \longrightarrow \mathcal{U}_{\backslash\phi}$$
$$\backslash\phi^*\mathrm{A} := \lfloor \mathsf{realign}\ [\phi \sqcup_{\phi \times \mathrm{A}} \mathrm{A} \mid \phi \hookrightarrow (\mathbf{1}, \lambda\_ : \mathbf{1}.\mathsf{inl}(\_ : \phi))] \rfloor$$

Above we have written $\phi \sqcup_{\phi \times \mathrm{A}} \mathrm{A}$ for the following internal pushout, which can be formed using quotient and coproduct types:

$$
\begin{array}{ccc}
\phi \times \mathrm{A} & \xrightarrow{\ \pi_2\ } & \mathrm{A} \\
{\scriptstyle \pi_1} \downarrow & & \vdots\ {\scriptstyle \mathsf{inr}} \\
\phi & \dashrightarrow & \phi \sqcup_{\phi \times \mathrm{A}} \mathrm{A} \\
 & \mathsf{inl} &
\end{array}
$$

(3.6*7) *Modalities.* Composing direct image with inverse image, one obtains an internal lex idempotent monad on the universe $\mathcal{U}$. We define notations for these monads below:

$$\bigcirc_\phi, \bullet_\phi : \mathcal{U} \longrightarrow \mathcal{U}$$
$$\bigcirc_\phi \mathrm{A} = \phi_* \phi^* \mathrm{A}$$
$$\bullet_\phi \mathrm{A} = \backslash\phi_* \backslash\phi^* \mathrm{A}$$

Of course, it is not strictly necessary for us to even write down the direct images — it is conventional to write canonical injections as subtype inclusions. But there will be times when clarity is served by being explicit.

(3.6*8) *Notation for the open subuniverse.* In **(3.6*4)** we used realignment twice: first to define the universe $\mathcal{U}_\phi$, and then to define the direct image $\phi^*\mathrm{A}$. We will harmlessly treat the corresponding realignment isomorphisms as implicit coercions in our notation:

$$
\begin{array}{rcll}
\mathrm{A} : \mathcal{U}_\phi & \vdash & \lceil \mathrm{A} \rceil : \{\phi\}\mathcal{U} & \text{written} \quad \mathrm{A} \\
\mathrm{A} : \{\phi\}\mathcal{U} & \vdash & \lfloor \mathrm{A} \rfloor : \mathcal{U}_\phi & \text{written} \quad \mathrm{A} \\
\mathrm{A} : \mathcal{U}_\phi, a : \bigcirc_\phi \mathrm{A} & \vdash & \lceil a \rceil : \{\phi\}\mathrm{A} & \text{written} \quad a \\
\mathrm{A} : \mathcal{U}_\phi, a : \{\phi\}\mathrm{A} & \vdash & \lfloor a \rfloor : \bigcirc_\phi \mathrm{A} & \text{written} \quad a
\end{array}
$$

(3.6*9) *Notation for the closed subuniverse.* In **(3.6*6)** we made two uses of realignment; first we realigned the extent $\{\mathcal{U} \mid \phi \hookrightarrow \mathbf{1}\}$ to define the closed subuniverse $\mathcal{U}_{\backslash\phi}$. In our

notations, we treat the corresponding realignment isomorphism as an implicit coercion:

$$A : \mathcal{U}_{\backslash\phi} \;\; \vdash \;\; \lceil A \rceil : \{\mathcal{U} \mid \phi \hookrightarrow \mathbf{1}\} \quad \text{written} \quad A$$
$$A : \{\mathcal{U} \mid \phi \hookrightarrow \mathbf{1}\} \;\; \vdash \;\; \lfloor A \rfloor : \mathcal{U}_{\backslash\phi} \qquad \text{written} \quad A$$

Second we realigned a pushout to obtain the inverse image $\backslash\phi^* A : \mathcal{U}_{\backslash\phi}$; of course, universal properties are stable under isomorphism, hence $\bullet_\phi A$ is also the pushout of the product projections of $\phi \times A$. We impose notations below for its introduction and elimination forms:

$$\frac{a : A}{\eta_{\backslash\phi}(a) : \bullet_\phi A} \qquad \frac{u : \phi}{\star : \bullet_\phi A} \qquad \frac{a : \bullet_\phi A \qquad c_\phi : \{\phi\}\, C \qquad c_A : A \to \{C \mid \phi \hookrightarrow c_\phi\}}{\mathsf{try}\; a \; [c_A \mid \phi \hookrightarrow c_\phi] : \{C \mid \phi \hookrightarrow c_\phi\}}$$

These notations are substantiated below, writing h for $\mathsf{try} - [c_A \mid \phi \hookrightarrow c_\phi]$:

(3.6∗10) The open and closed subuniverses $\mathcal{U}_\phi, \mathcal{U}_{\backslash\phi}$ are both strong; the condition for the open subuniverse is trivial (but tedious) to verify; for the closed subuniverse, realignment at $\psi : \Omega$ is achieved using realignment at $\psi \vee \phi$.[2]

§3.7. ALGEBRAS FOR A SIGNATURE IN A UNIVERSE

(3.7∗1) Let $\mathbf{X}$ be a topos, and let $\mathbb{S}$ be a signature in the logical framework of Chapter 1; an algebra for $\mathbb{S}$ in $\mathbf{Set_X}$ is nothing more than a locally Cartesian closed functor $\mathcal{T}_\mathbb{S} \longrightarrow \mathbf{Set_X}$ where $\mathcal{T}_\mathbb{S}$ is the category of judgments defined in **(1.4∗8)**. When all the sorts of $\mathbb{S}$ are interpreted by small objects for a given universe $\mathcal{U}$ of $\mathbf{Set_X}$, we may give an alternative internal characterization of $\mathbb{S}$-algebras.

(3.7∗2) Every universe in $\mathbf{Set_X}$ gives rise to a model of the theory of LF signatures in the following sense. Let $\mathcal{U}$ be a universe in $\mathbf{Set_X}$; we have a locally Cartesian closed functor $\llbracket - \rrbracket_\mathcal{U} : \mathbf{SIG} \longrightarrow \mathbf{Set_X}$ sending $\square$ to $\mathcal{U}$, and $\boxdot : \mathbf{SIG}_{/\square}$ to $\dot{\mathcal{U}} : (\mathbf{Set_X})_{/\mathcal{U}}$. The functor $\llbracket - \rrbracket_\mathcal{U}$ sends each signature $\mathbb{S}$ to the sheaf of $\mathcal{U}$-small $\mathbb{S}$-algebras over $\mathbf{X}$.

---

[2] I have checked these conditions carefully in Coq, because it is intricate enough that an informal proof may not be convincing.

Open subuniverse:

$$\mathcal{U}_\phi : \{\mathcal{V} \mid \phi \hookrightarrow \mathcal{U}\}$$
$$\mathcal{U}_\phi := \mathsf{realign}\ [\{\phi\}\mathcal{U} \mid \phi \hookrightarrow (\mathcal{U}, \lambda \mathrm{A} : \mathcal{U}.\lambda\{z : \phi\}.\mathrm{A})]$$

$$\phi^* : \mathcal{U} \longrightarrow \mathcal{U}_\phi$$
$$\phi^*\mathrm{A} = \lfloor \lambda\{z : \phi\}.\mathrm{A} \rfloor$$

$$\phi_* : \mathcal{U}_\phi \rightarrowtail \mathcal{U}$$
$$\phi_*(\mathrm{A}) = \mathsf{realign}\ [\{\phi\}\ \lceil \mathrm{A} \rceil \mid \phi \hookrightarrow (\lceil \mathrm{A} \rceil\{\star\}, \lambda a : \lceil \mathrm{A} \rceil\{\star\}.\lambda\{z : \phi\}.a)]$$

Closed subuniverse:

$$\mathcal{U}_{\backslash\phi} : \{\mathcal{V} \mid \phi \hookrightarrow \mathbf{1}\}$$
$$\mathcal{U}_{\backslash\phi} = \mathsf{realign}\ [\{\mathcal{U} \mid \phi \hookrightarrow \mathbf{1}\} \mid \phi \rightarrow (\mathbf{1}, \lambda\_ : \mathbf{1}.\star)]$$

$$\backslash\phi^* : \mathcal{U} \longrightarrow \mathcal{U}_{\backslash\phi}$$
$$\backslash\phi^*\mathrm{A} := \lfloor \mathsf{realign}\ [\phi \sqcup_{\phi\times\mathrm{A}} \mathrm{A} \mid \phi \hookrightarrow (\mathbf{1}, \lambda\_ : \mathbf{1}.\mathsf{inl}(\_ : \phi))] \rfloor$$

$$\backslash\phi_* : \mathcal{U}_{\backslash\phi} \rightarrowtail \mathcal{U}$$
$$\backslash\phi_*(\mathrm{A}) = \lceil \mathrm{A} \rceil$$

Modalities:

$$\bigcirc_\phi, \bullet_\phi : \mathcal{U} \longrightarrow \mathcal{U}$$
$$\bigcirc_\phi\mathrm{A} = \phi_*\phi^*\mathrm{A}$$
$$\bullet_\phi\mathrm{A} = \backslash\phi_*\backslash\phi^*\mathrm{A}$$

Figure 3.1: A summary of the construction of open and closed subuniverses for a proposition $\phi : \Omega$ and a universe $\mathcal{U} : \mathcal{V}$.

◆ **(3.7∗3)** Consider the signature $\mathbb{ML}_{\varnothing}$ of the "walking type theory" **(1.6∗3)**; then the interpretation $[\![\mathbb{ML}_{\varnothing}]\!]_{\mathcal{U}}$ is the following object of $\mathbf{Set_X}$:

$$[\![\mathbb{ML}_{\varnothing}]\!]_{\mathcal{U}} \cong \sum\nolimits_{\mathsf{tp}:\mathcal{U}}(\mathsf{tp} \to \mathcal{U})$$

If $\mathcal{U} \leq \mathcal{V}$, we have the canonical coercion $[\![\mathbb{ML}_{\varnothing}]\!]_{\mathcal{U}} \longrightarrow [\![\mathbb{ML}_{\varnothing}]\!]_{\mathcal{V}}$ that we will leave implicit in the presence of cumulative universes. In fact these coercions exist more generally for any signature $\mathbb{S}$, in essence because $\square$ is restricted in a signature to appear in strictly positive positions, recalling **(1.2.1∗4)**.

▨ **(3.7∗4)** Let ϕ be an open of $\mathbf{Set_X}$; because the open modality **(3.6∗7)** commutes with dependent product/sum types and equality types, we always have a canonical isomorphism $\bigcirc_{\phi}[\![\mathbb{S}]\!]_{\mathcal{U}} \cong [\![\mathbb{S}]\!]_{\mathcal{U}_{\phi}}$ for any signature $\mathbb{S} : \mathbf{SIG}$.

Part III

Synthetic Tait Computability

CHAPTER 4

TAIT'S METHOD OF COMPUTABILITY

This chapter contains joint work with Robert Harper [SH21] and Daniel Gratzer.

§4.1. CONTEXTS, FIGURE SHAPES, AND TAIT COMPUTABILITY

※ **(4.1∗1)** Experience has shown that the important theorems about the syntax of type theory (canonicity, parametricity, normalization, *etc.*) do *not* hold unconditionally, but only with respect to a restricted form of context. For instance, because *equality* is a judgment it may appear in either the domain or in the codomain of a morphism in the category $\mathcal{T}$ of judgments, but unrestricted assumptions of equality are well-known to make the type checking problem undecidable [CCD17].

Hence, metatheorems such as the existence of canonical forms are stated not unconditionally for a form of judgment like $\mathsf{tp} : \mathcal{T}$, but are instead formulated relative to the *nerve* functor $\mathrm{N}_\rho : \mathcal{T} \longrightarrow \mathrm{Pr}(\mathcal{C})$ sending $\mathrm{X} : \mathcal{T}$ to the "functor of points" $\mathrm{Hom}_\mathcal{T}(\rho - , \mathrm{X})$ for some carefully chosen *figure shape* $\rho : \mathcal{C} \longrightarrow \mathcal{T}$.

※ **(4.1∗2)** The figure shape $\rho : \mathcal{C} \longrightarrow \mathcal{T}$ plays essentially the same role as *worlds* in the Edinburgh tradition of logical frameworks [HL07]. In some cases, the figure shape is a full subcategory of the category of judgments spanned by certain judgments distinguished as contexts — such canonical figure shapes may be used to obtain a traditional "gammas and turnstiles" presentation from the category of judgments of a type theory.

▪ **(4.1∗3)** Each figure shape $\rho : \mathcal{C} \longrightarrow \mathcal{T}$ induces a (generalized) notion of Kripke logical relations or *Tait computability* on $\mathcal{T}$: a ρ-computability structure over a sort $X : \mathcal{T}$ is a family of presheaves $X' : \mathrm{Pr}(\mathcal{C})_{/\mathrm{N}_\rho(X)}$ — the fibers of X' over a given piece of syntax are the "proofs" of computability/reducibility of that piece of syntax; the indexing in $\mathcal{C}$ generalizes the variation over Kripke worlds in Kripke logical relations.

Such computability structures organize themselves into a category $\mathcal{G} = \mathrm{Pr}(\mathcal{C}) \downarrow \mathrm{N}_\rho$ called the *Artin gluing* of N_ρ; a morphism in $\mathcal{G}$ is (roughly) a morphism in $\mathcal{T}$ together with a *proof* that it takes computable terms to computable terms. The "fundamental theorem" of ρ-computability is then to exhibit a suitable structure preserving functor $\mathcal{T} \longrightarrow \mathcal{G}$ that is a section of the projection $\mathsf{gl} : \mathcal{G} \longrightarrow \mathcal{T}$.

◆ **(4.1∗4)** Consider a type theory $\mathcal{T}$ that contains an answer type ans and two constants yes, no : ans; the property that every global element a : ans is either equal to yes or no is called **closed canonicity**. The closed canonicity theorem is stated (and in many cases proved) relative to the trivial figure shape $\ast : \mathbf{1}_{\mathbf{Cat}} \longrightarrow \mathcal{T}$ determined by the terminal judgment $\mathbf{1}_\mathcal{T}$; the corresponding nerve functor is the global sections functor. This trivial figure shape corresponds to the fact that canonicity is a statement about terms without any free variables.

The Tait computability arising from the trivial figure shape assigns to each sort $X : \mathcal{T}$ a family of sets X' indexed in the *closed* elements of X; this is only a mild generalization of the conventional logical predicates argument known to type theorists. The main substantive difference is that the families of sets are restricted in the conventional argument (to little advantage, we will see) to families of truth values, rendering X' a predicate on the closed elements of X. We will return to this example in §4.5.

◆ **§4.1.1. Cubical canonicity**

▪ **(4.1.1∗1)** Let $\mathcal{T}_\square$ be the category of judgments of *cubical type theory*. $\mathcal{T}_\square$ is similar to $\mathcal{T}$ from **(4.1∗4)** except that it contains some additional objects and morphisms corresponding to an *interval* (along with a number of other primitives that we need not detail now):

$$\mathbb{I} : \mathcal{T}_\square \qquad\qquad 0,1 : \mathbf{1}_{\mathcal{T}_\square} \longrightarrow \mathbb{I}$$

※ **(4.1.1∗2)** While it remains possible to *state* a closed canonicity result for $\mathcal{T}_\square$ in the spirit of **(4.1∗4)**, it does not appear to be possible to prove such a result in terms of the trivial figure shape $\ast : \mathbf{1}_{\mathbf{Cat}} \longrightarrow \mathcal{T}_\square$. Instead one must apparently prove a stronger canonicity result

characterizing terms not only in the empty context, but in any context of the form $\mathbb{I}^n$ — in other words, the "purely cubical" contexts. The generalized canonicity result is achieved by means of a more complicated (but geometrically natural) figure shape.

(4.1.1∗3) Write $\Box$ for the *Cartesian cube category*, which is the same as the free category with finite products generated by an object with two global points. The universal property of $\Box$ places "finite product categories $\mathcal{E}$ equipped with an interval object" into unique correspondence with finite product preserving functors $\Box \longrightarrow \mathcal{E}$. Hence we have a finite product preserving functor $\rho : \Box \longrightarrow \mathcal{T}_\Box$, the *cubical figure shape*, taking each formal finite product $[n] : \Box$ to the n-fold product of the interval $\rho[n] = \mathbb{I}^n : \mathcal{T}$.

(4.1.1∗4) The nerve $N_\rho : \mathcal{T}_\Box \longrightarrow \mathrm{Pr}(\Box)$ takes, for instance, the sort $\mathsf{tp} : \mathcal{T}_\Box$ of types to the *functor of cubical points* $\mathrm{Hom}_{\mathcal{T}_\Box}(\rho - , \mathsf{tp})$, consisting of the collection of all types with free variables ranging only over the interval. The use of such a nerve was suggested by Awodey in 2015 as a way to rationalize the nuts-and-bolts canonicity argument of Huber [Hub18], and employed in unpublished work of Awodey and Fiore in 2018, and by Sterling, Angiuli, and Gratzer [SAG19] in 2019; in light of the analysis of Kripke logical relations by Jung and Tiuryn [JT93] as early as 1993, it is not surprising in hindsight that this cubical nerve plays an important role.

(4.1.1∗5) The Tait computability theory induced by ρ associates to each sort $X : \mathcal{T}_\Box$ a *cubical computability structure*, which is a family of cubical sets $X' : \mathrm{Pr}(\Box)_{/N_\rho X}$ indexed in the *cubical points* of X. The cubical Tait computability sketched here generalizes and immensely simplifies the argument of Huber [Hub18] for the canonicity of cubical type theory, to the point that cubical canonicity can be established without more difficulty than ordinary canonicity in a purely conceptual way involving no technical lemmas whatsoever.

§4.2. CANONICITY FOR TYPED λ-CALCULUS, EXPLICITLY

(4.2∗1) We warm up by considering a familiar example, canonicity of the λ-calculus. Although conceptual arguments for λ-canonicity are well known enough to appear in undergraduate-level textbooks [Cro93], regrettably intricate proofs employing operational semantics and "hand-coded" logical relations still remain dominant in both the scholarly and pedagogical literature.

(4.2∗2) By *typed λ-calculus* we mean a simple type theory with finite products $A \times B \times \ldots$, exponentials $A \to B$, and an answer type ans with two constants $\mathsf{yes}, \mathsf{no} : \mathsf{ans}$. The purpose of the answer type, as argued by Harper [Har16, §48.2], is to provide an *observation* with respect to which one may state and prove a canonicity result. The statement of the canonicity theorem is as follows:

> *Canonicity.* Any closed term $\cdot \vdash a : \mathsf{ans}$ is equal to either yes or no.

(**4.2∗3**) Rather than defining the syntax of the λ-calculus by means of a grammar, dealing with the complexities of α-renaming and substitution, we may simply define the syntax of λ-calculus to be the *free Cartesian closed category* $\mathcal{T}$ generated by one object **ans** and two constants yes,no : $\mathbf{1} \longrightarrow$ ans. The motivation of such a definition is that the syntax of a given theory may be presented in many different ways, but all these presentations give rise to the same *category*, considered up to equivalence of categories.

An important observation on which this dissertation rests is that the theorems of type theory can be stated in a way that is invariant under equivalence of categories; hence, there is no need to concern ourselves with the specifics of tree representations or variable binding conventions, which in hindsight can be seen to be predilections of essentially the same ill-advised nature as the unfortunate emphasis of "parsing" in undergraduate computer science pedagogy, or counting of parentheses in undergraduate logic pedagogy.

(**4.2∗4**) As previewed in (**4.1∗4**) we consider the trivial figure shape $\{\mathbf{1}_{\mathcal{T}}\}$: $\mathbf{1}_{\mathbf{Cat}} \longrightarrow \mathcal{T}$ determined by the terminal object $\mathbf{1}_{\mathcal{T}}$. Hence the nerve $N_{\{\mathbf{1}_{\mathcal{T}}\}}$ takes each A : $\mathcal{T}$ to its set of closed terms $\mathrm{Hom}_{\mathcal{T}}(\mathbf{1}_{\mathcal{T}},A)$ under the identification $\mathrm{Pr}(\mathbf{1}_{\mathbf{Cat}}) = \mathbf{Set}$; in other words, $N_{\{\mathbf{1}_{\mathcal{T}}\}}$ is the global sections functor $\mathbf{\Gamma} : \mathcal{T} \longrightarrow \mathbf{Set}$.

(**4.2∗5**) To develop a theory of *closed computability* for $\mathcal{T}$ we will form the Artin gluing of the global sections functor $\mathbf{\Gamma}$, namely the comma category $\mathcal{G} = \mathbf{Set} \downarrow \mathbf{\Gamma}$; it will be helpful in the longer term to think of $\mathcal{G}$ as the fiber of the codomain fibration $\mathbf{Set}^{\rightarrow} \longrightarrow \mathbf{Set}$ over $\mathcal{T}$ as depicted below:

$$
\begin{array}{ccc}
\mathcal{G} & \dashrightarrow & \mathbf{Set}^{\rightarrow} \\
{\scriptstyle \mathrm{gl}} \Big\downarrow & \lrcorner & \Big\downarrow {\scriptstyle \mathrm{cod}} \\
\mathcal{T} & \xrightarrow[\mathbf{\Gamma}]{} & \mathbf{Set}
\end{array}
$$

(**4.2∗6**) An object of $\mathcal{G}$ is called a *computability structure*, and is defined by specifying a pair of an object A : $\mathcal{T}$ together with a morphism $A' \longrightarrow \mathbf{\Gamma}(A)$: $\mathbf{Set}$, *i.e.* a family of sets A'_a indexed in closed terms $a : \mathbf{1}_{\mathcal{T}} \longrightarrow A$. Let $(A,A' \longrightarrow \mathbf{\Gamma}(A))$ and $(B,B' \longrightarrow \mathbf{\Gamma}(B))$ be two objects of $\mathcal{G}$; a morphism between them is given by a morphism $f : A \longrightarrow B : \mathcal{T}$ together with a commuting square in $\mathbf{Set}$ configured like so:

$$
\begin{array}{ccc}
A' & \xdashrightarrow{\ f'\ } & B' \\
\Big\downarrow & & \Big\downarrow \\
\mathbf{\Gamma}(A) & \xrightarrow[\mathbf{\Gamma}(f)]{} & \mathbf{\Gamma}(A)
\end{array}
$$

(**4.2∗7**) As alluded to in (**4.1∗3**), we will prove the canonicity theorem for $\mathcal{T}$ by exhibiting a suitable model of the typed λ-calculus in $\mathcal{G}$ in which the interpretation of the answer

type **ans** associates to each closed term of answer type a proof that it is either **yes** or **no**. Technically speaking, this means to exhibit a diagram of the following shape such that the dotted functor preserves Cartesian closed structure:

$$
\begin{array}{ccc}
\mathcal{T} & \overset{s}{\dashrightarrow} & \mathcal{G} \\
 & \searrow{\scriptstyle id_\mathcal{T}} & \downarrow{\scriptstyle gl} \\
 & & \mathcal{T}
\end{array}
\tag{4.2*7*1}
$$

Diagram 4.2*7*1 is the mathematical version of the *fundamental theorem of logical relations*: one is exhibiting a computability structure over every construct from $\mathcal{T}$ in a way that is compatible with substitution (this is the content of functoriality). However, it is *always* considerably less technical to exhibit Diagram 4.2*7*1 than to manually prove the FTLR — not least because one may develop a wealth of general theorems about gluings of different kinds of categories that can be reused, unlike the case for conventional non-mathematical applications of logical relations. It is also less technical even in the cases where general results are not available beforehand, because the categorical setting provides certain significant simplifications and reductions, a strange state of affairs that will come into relief when we develop the topos theoretic perspective on Tait computability later on.

▪ **(4.2*8)** Using the universal property of $\mathcal{T}$ as the free Cartesian closed category generated by an answer type, we may split **(4.2*7)** into two easy steps:

1) Show that $\mathcal{G}$ has finite products and exponentials, and that **gl** preserves them.
2) Choose a suitable interpretation of the answer type in $\mathcal{G}$, aligned over the answer type from $\mathcal{T}$ in the following configuration:

$$
\begin{array}{ccccccc}
1_\mathcal{G} & \overset{\text{yes}*}{\dashrightarrow} & \text{ans}* & \overset{\text{no}*}{\dashleftarrow} & 1_\mathcal{G} & \qquad & \mathcal{G} \\
\downarrow & & \downarrow & & \downarrow & & \mid{\scriptstyle gl} \\
 & & & & & & \downarrow \\
1_\mathcal{T} & \underset{\text{yes}}{\longrightarrow} & \text{ans} & \underset{\text{no}}{\longleftarrow} & 1_\mathcal{T} & & \mathcal{T}
\end{array}
\tag{4.2*8*1}
$$

▪ **(4.2*9)** *Closure of $\mathcal{G}$ under finite products.* This is simple to establish because finite products may be computed pointwise in $\mathcal{G}$; for pedagogical reasons, however, we give the construction explicitly. We will use the (unconditional) fact that the global sections functor $\mathbf{\Gamma}$ for any category preserves finite products. Given a finite family of objects $X_{(i \in I)} : \mathcal{G}$ we define the product $\prod_{i \in I} X_i : \mathcal{G}$ by the following family of sets, writing $X_i^{\downarrow} : X_i^{\bullet} \to \mathbf{\Gamma}(X_i^{\circ})$

for the corresponding families of sets:

$$
\begin{array}{c}
\prod_{i\in I}X_i^\bullet \\
\Big\downarrow \prod_{i\in I}X_i^\downarrow \\
(\prod_{i\in I}X_i)^\downarrow \qquad \prod_{i\in I}\mathbf{\Gamma}(X_i^\circ) \\
\Big\downarrow \cong \\
\mathbf{\Gamma}(\prod_{i\in I}X_i^\circ)
\end{array}
$$

It remains to check that $\prod_{i\in I}X_i : \mathcal{G}$ so defined is *in fact* the cartesian product of the finitary family $X_{(i\in I)}$. We first exhibit projections $\pi_k : \prod_{i\in I}X_i \longrightarrow X_k$ for each $k \in I$:

$$
\begin{array}{ccc}
\prod_{i\in I}X_i^\bullet & \xdashrightarrow{\ \pi_k\ } & X_k^\bullet \\
(\prod_{i\in I}X_i)^\downarrow \Big\downarrow & & \Big\downarrow X_k^\downarrow \\
\mathbf{\Gamma}(\prod_{i\in I}X_i^\circ) & \xdashrightarrow[\mathbf{\Gamma}(\pi_k)]{} & \mathbf{\Gamma}(X_k^\circ)
\end{array}
$$

Then it remains to show that the (discrete) cone $\{\pi_k : \prod_{i\in I}X_i \longrightarrow X_k\}$ given by all the projections is *universal*. Fixing any other cone $\{p_k : Z \longrightarrow X_k\}$, we must exhibit a *unique* morphism $Z \longrightarrow \prod_{i\in I}X_i$ with the property that all the triangles below commute:

$$
\begin{array}{ccc}
Z & \dashrightarrow & \prod_{i\in I}X_i \\
& \searrow_{p_k} & \Big\downarrow \pi_k \\
& & X_k
\end{array}
$$

Rephrasing into the language of **Set** using **(4.2∗6)**, the existence and uniqueness of the universal map $Z \longrightarrow \prod_{i\in I}X_i$ follows immediately from the corresponding universal properties of the cartesian products $\prod_{i\in I}X_i^\bullet : \mathbf{Set}$ and $\prod_{i\in I}X_i^\circ : \mathcal{T}$.

(4.2∗10) *Closure of $\mathcal{G}$ under exponentials.* The construction of exponentials in $\mathcal{G}$ is more technical than that of finite products, but it can be seen to mirror *exactly* the conventional method for defining logical relations at function type: one must attach a "semantic function" between sets of computable closed terms to a syntactic function. Fixing $X, Y : \mathcal{G}$

we construct their exponential $W := (X \to Y)$ as a pullback in **Set** involving both the exponentials of **Set** and the exponentials of $\mathcal{T}$:

$$
\begin{array}{ccc}
W^{\bullet} & \dashrightarrow & (X^{\bullet} \to Y^{\bullet}) \\
\downarrow W^{\downarrow} & \lrcorner & \downarrow Y^{\downarrow} \circ - \\
\mathbf{\Gamma}(X^{\circ} \to Y^{\circ}) & \xrightarrow{f \mapsto \lambda x.\mathsf{app}(f, X^{\downarrow}x)} & (X^{\bullet} \to \mathbf{\Gamma}(Y^{\circ})) \\
\end{array}
$$

with app from $\mathbf{\Gamma}(X^{\circ} \to Y^{\circ})$ and $(- \circ X^{\downarrow})$ into $(\mathbf{\Gamma}(X^{\circ}) \to \mathbf{\Gamma}(Y^{\circ}))$.

The map app is implemented by the counit $\epsilon : ((X^{\circ} \to Y^{\circ}) \times X^{\circ}) \to Y^{\circ}$ of the exponential after commuting some finite products past $\mathbf{\Gamma}$:

$$
\mathrm{Hom}(\mathbf{\Gamma}(X^{\circ} \to Y^{\circ}), \mathbf{\Gamma}(X^{\circ}) \to \mathbf{\Gamma}(Y^{\circ}))
$$
$$
\cong \mathrm{Hom}(\mathbf{\Gamma}(X^{\circ} \to Y^{\circ}) \times \mathbf{\Gamma}(X^{\circ}), \mathbf{\Gamma}(Y^{\circ}))
$$
$$
\cong \mathrm{Hom}(\mathbf{\Gamma}((X^{\circ} \to Y^{\circ}) \times X^{\circ}), \mathbf{\Gamma}(Y^{\circ}))
$$

To check that $W : \mathcal{G}$ so-defined is actually the exponential $(X \to Y)$, we check that we have the family of isomorphisms below natural in a computability structure Z taken by gl to the corresponding isomorphism in $\mathcal{T}$:

$$
\text{(natural in Z)} \quad \alpha_Z : \mathrm{Hom}_{\mathcal{G}}(Z, W) \cong \mathrm{Hom}_{\mathcal{G}}(Z \times X, Y) \tag{4.2*10*1}
$$

(4.2*11) Identity 4.2*10*1 above is a more compact way to package the existence of λ-abstraction and application operators for W that (1) commute with substitution, and (2) satisfy the β and η laws of the function type. Verifying this condition is left to the reader — not only because it is a worthwhile and elementary exercise, but also because we will later on see how to totally dispense with it using a much stronger general theorem.

(4.2*12) Any construction of a connective that has both a β-law and an η-law is automatically unique up to unique isomorphism! This means that there can be no creativity or careful thought involved in exhibiting its computability structure: either it exists or it doesn't, but there is no question of the "good" way to define it. For this reason, the usual emphasis on careful constructions of logical relations for (*e.g.*) product types or function types that pervades the literature must be regarded as essentially misguided, an artifact of the now-eclipsed era in which the universal properties of type theoretic connectives were either omitted or hidden underneath inessential syntactical matters such as raw terms.

(4.2*13) The second step of **(4.2*8)**, the interpretation of the answer type, is the meat of the canonicity argument; the rest of it is bureaucracy that we will later on see how to

completely dispense with in all cases. First, we note that however we define $\mathsf{ans}* : \mathcal{G}$, we must have $\mathsf{gl}(\mathsf{ans}*) = \mathsf{ans}$, since we are trying to make a computability structure over the answer type, not some other type!

We will *choose* to define $\mathsf{ans}* : \mathcal{G}$ by the family of sets $\{\mathsf{yes},\mathsf{no}\} \hookrightarrow \mathbf{\Gamma}(\mathsf{ans})$ that isolates the *canonical* closed terms of answer type. Recalling the computation of morphisms in $\mathcal{G}$ in terms of squares in **Set** of **(4.2∗6)**, we substantiate Diagram 4.2∗8∗1 in the language of **Set** as follows:

$$
\begin{array}{ccccc}
\mathbf{1}_{\mathbf{Set}} & \xrightarrow{\;\mathsf{yes}\;} & \{\mathsf{yes},\mathsf{no}\} & \xleftarrow{\;\mathsf{no}\;} & \mathbf{1}_{\mathbf{Set}} \\
\downarrow & & \updownarrow & & \downarrow \\
\mathbf{\Gamma}(\mathbf{1}_{\mathcal{T}}) & \xrightarrow[\mathbf{\Gamma}(\mathsf{yes})]{} & \mathbf{\Gamma}(\mathsf{ans}) & \xleftarrow[\mathbf{\Gamma}(\mathsf{no})]{} & \mathbf{\Gamma}(\mathbf{1}_{\mathcal{T}})
\end{array}
$$

Considering the division of labor of **(4.2∗8)**, we have defined by means of **(4.2∗9)**, **(4.2∗10)** and **(4.2∗13)** a Cartesian closed section $\mathsf{s} : \mathcal{T} \longrightarrow \mathcal{G}$ of the projection $\mathsf{gl} : \mathcal{G} \dashrightarrow \mathcal{T}$ such that $\mathsf{s}(\mathsf{ans}) = \mathsf{ans}*$, *etc.*

(4.2∗14) *Canonicity.* The canonicity result is then read off directly off our construction. Fixing a closed term $a : \mathbf{1}_{\mathcal{T}} \longrightarrow \mathsf{ans}$, the upstairs functor $\mathsf{s} : \mathcal{T} \longrightarrow \mathcal{G}$ of Diagram 4.2∗7∗1 gives us a morphism $\mathsf{s}(a) : \mathbf{1}_{\mathcal{G}} \cong \mathsf{s}(\mathbf{1}_{\mathcal{T}}) \longrightarrow \mathsf{ans}*$ such that $\mathsf{gl}(\mathsf{s}(a)) = a$. Rephrasing into the language of **Set**, we have a commuting square in the following configuration:

$$
\begin{array}{ccc}
\mathbf{1}_{\mathbf{Set}} & \xdashrightarrow{\;a^{\bullet}\;} & \{\mathsf{yes},\mathsf{no}\} \\
\downarrow & & \downarrow \\
\mathbf{\Gamma}(\mathbf{1}_{\mathcal{T}}) & \xrightarrow[\mathbf{\Gamma}(a)]{} & \mathbf{\Gamma}(\mathsf{ans})
\end{array}
\qquad (4.2\!*\!14\!*\!1)
$$

The commutativity of Diagram 4.2∗14∗1 states that we have an element $a^{\bullet} \in \{\mathsf{yes},\mathsf{no}\}$ such that $a = a^{\bullet}$ under the identification $\mathrm{Hom}_{\mathbf{Set}}(\mathbf{\Gamma}(\mathbf{1}_{\mathcal{T}}),\mathbf{\Gamma}(\mathsf{ans})) \cong \mathbf{\Gamma}(\mathsf{ans})$. In other words, either $a = \mathsf{yes}$ or $a = \mathsf{no}$.

§4.3. GLUING WITH FIGURE SHAPES

(4.3∗1) In §4.1 we considered ρ-computability for a figure shape $\rho : \mathcal{C} \longrightarrow \mathcal{T}$ where $\mathcal{T}$ is the category of judgments for a given type theory. Here we explore a more conceptual and geometric construction that generalizes the category of ρ-computability structures — as a particularly well-behaved special case of the Artin gluing detailed in §2.4.

⁂ **(4.3∗2)** Let $\mathcal{T}$ be a type theory and fix a figure shape $\rho : \mathcal{C} \longrightarrow \mathcal{T}$; in §4.2 we constructed the Artin gluing $\mathcal{G} = \mathrm{Pr}(\mathcal{C}) \downarrow \mathrm{N}_\rho$ of the nerve functor $\mathrm{N}_\rho : \mathcal{T} \longrightarrow \mathrm{Pr}(\mathcal{C})$ to form the category of ρ-computability structures. A disadvantage of the perspective detailed there is that the category $\mathcal{G}$ does not *a priori* have any useful structure: we must manually show that it possesses products, exponentials, *etc.*, but as our type theoretic structure becomes more complex (consider for example dependent products), the corresponding explicit constructions start to become intractable and unintuitive.

We have discovered a very simple way to work around this problem and render any explicit construction of type theoretic structure just as redundant as it is ill-advised. Rather than considering the category $\mathcal{G} = \mathrm{Pr}(\mathcal{C}) \downarrow \mathrm{N}_\rho$ of ρ-computability structures over $\mathcal{T}$, we will instead work with a topos $\mathbf{G}$ such that $\mathcal{G}$ embeds fully faithfully into $\mathbf{Set_G}$. Hence, all type theoretic structure will already exist (without any special effort on our part) in $\mathbf{Set_G}$ and the substantiation of a type theoretic metatheorem is reduced to elementary considerations of representation by universes.

▨ **(4.3∗3)** From the figure shape $\rho : \mathcal{C} \longrightarrow \mathcal{T}$, we obtain a morphism between presheaf topoi $\rho : \widehat{\mathcal{C}} \longrightarrow \widehat{\mathcal{T}}$ (no ambiguity will be caused by sharing the name); the inverse image part of ρ simply restricts a presheaf on $\mathcal{T}$ to a presheaf on $\mathcal{C}$. We intend to create a topos $\mathbf{G}$ that includes $\widehat{\mathcal{T}}$ via an open immersion and $\widehat{\mathcal{C}}$ via a complementary closed immersion. We may achieve this geometrically by forming the cylinder $\widehat{\mathcal{C}} \times \mathbb{S}$ and gluing the *open* copy of $\widehat{\mathcal{C}}$ along ρ to the topos $\widehat{\mathcal{T}}$ by means of a pushout in **Topos**:

$$
\begin{array}{ccc}
\widehat{\mathcal{C}} & \xrightarrow{\ \rho\ } & \widehat{\mathcal{T}} \\
{\scriptstyle \circ_{\widehat{\mathcal{C}}}}\downarrow & & \downarrow{\scriptstyle j} \\
\widehat{\mathcal{C}} \times \mathbb{S} & \longrightarrow & \mathbf{G}
\end{array}
\qquad (4.3{*}3{*}1)
$$

$$
{\scriptstyle \bullet_{\widehat{\mathcal{C}}}}\uparrow \qquad \nearrow{\scriptstyle i}
$$

$$
\widehat{\mathcal{C}}
$$

⩔ **(4.3∗4)** We will examine the logos $\mathbf{Set_G}$ induced by **(4.3∗3)** from the algebraic perspective. The first step is to dualize Diagram 4.3∗3∗1 into the language of categories.

$$
\begin{array}{ccc}
\mathbf{Set_G} & \longrightarrow & \mathbf{Set}_{\widehat{\mathcal{C}} \times \mathbb{S}} \\
{\scriptstyle j^*}\downarrow & & \downarrow{\scriptstyle \langle \mathrm{id}_{\widehat{\mathcal{C}}}, \circ \rangle^*} \\
\mathbf{Set}_{\widehat{\mathcal{T}}} & \xrightarrow{\ \rho^*\ } & \mathbf{Set}_{\widehat{\mathcal{C}}}
\end{array}
\qquad (4.3{*}4{*}1)
$$

Following **(2.2.4∗4)** we may rewrite the right hand side of Diagram 4.3∗4∗1 as follows:

$$
\begin{array}{ccc}
\mathbf{Set_G} & \longrightarrow & \mathrm{Pr}(\mathcal{C})^{\rightarrow} \\
{\scriptstyle j^*}\downarrow & \lrcorner & \downarrow{\scriptstyle \mathrm{cod}} \\
\mathrm{Pr}(\mathcal{T}) & \xrightarrow[\rho^*]{} & \mathrm{Pr}(\mathcal{C})
\end{array}
\qquad\qquad (4.3\ast4\ast2)
$$

Hence $\mathbf{Set_G}$ is just the Artin gluing of the inverse image functor $\rho^* : \mathrm{Pr}(\mathcal{T}) \rightarrow \mathrm{Pr}(\mathcal{C})$ in the sense of **(2.4∗5)**.

▪ **(4.3∗5)** The existence of a left adjoint $\rho_! \dashv \rho^*$ characterizes the property of $\rho : \widehat{\mathcal{C}} \rightarrow \widehat{\mathcal{T}}$ being an *essential geometric morphism*. In this case the closed immersion $\widehat{\mathcal{C}} \hookrightarrow \mathbf{G}$ is also essential, with $i_! \dashv i^*$ taking each $\mathrm{E} : \mathrm{Pr}(\mathcal{C})$ to the family determined by the unit of the monad $\rho^*\rho_!$:

$$
i_! : \mathrm{Pr}(\mathcal{C}) \rightarrow \mathbf{Set_G}
$$
$$
i_! : \mathrm{E} \longmapsto \left(\rho_!\mathrm{E}, \eta_{\mathrm{E}} : \mathrm{E} \rightarrow \rho^*\rho_!\mathrm{E}\right)
$$

In this case, the monad $\bullet := i_*i^* : \mathbf{Set_G} \rightarrow \mathbf{Set_G}$ that isolates the part of a sheaf lying over the closed subtopos has a left adjoint $\blacksquare := i_!i^*$:

$$
\begin{aligned}
\mathrm{Hom}_{\mathbf{Set_G}}(\mathrm{E},\bullet\mathrm{F}) &= \mathrm{Hom}_{\mathbf{Set_G}}(\mathrm{E},i_*i^*\mathrm{F}) \\
&\cong \mathrm{Hom}_{\mathrm{Pr}(\mathcal{C})}(i^*\mathrm{E},i^*\mathrm{F}) \\
&\cong \mathrm{Hom}_{\mathbf{Set_G}}(i_!i^*\mathrm{E},\mathrm{F}) \\
&= \mathrm{Hom}_{\mathbf{Set_G}}(\blacksquare\mathrm{E},\mathrm{F})
\end{aligned}
$$

If $i_!$ is additionally left exact, then so is $\blacksquare$.

§4.4. SYNTHETIC TAIT COMPUTABILITY

⊽ **(4.4∗1)** In this section, let ¶ be an open of a topos $\mathbf{G}$; hence in the internal language of $\mathbf{Set_G}$ we have a proposition $\P : \Omega$; we fix a strict universe hierarchy $\mathcal{U} \leq \mathcal{V} < \mathcal{W}$; accordingly we will freely employ the notation **(3.4∗7)** in which $\langle\uparrow^{\mathcal{V}}_{\mathcal{U}}\rangle$ are omitted. We will write $\bigcirc,\bullet$ for the modalities $\bigcirc_\P,\bullet_\P$ respectively.

※ **(4.4∗2)** The open and closed modalities corresponding to $\P : \Omega$ provide the abstract setting in which to *synthetically* develop the main components of proofs by Tait computability.

Rather than explicitly constructing logical relations or computability predicates, we take the perspective that *everything* is a computability structure (relative to $\P : \Omega$), and use the modalities to fracture an object into its syntactic and semantic aspects as needed.

First we build up a large library of small results in this "synthetic Tait computability"; then, we instantiate these constructions at a specific glued topos and extract a metatheorem for a specific type theory, as in §4.5. Our approach to computability arguments / logical relations is analogous to other topos theoretic reconstructions of differential geometry [Koc06; Koc09], differential topology [BGS18], algebraic geometry [Ble17], domain theory [Bir+11a; Hyl91], and computability theory [Bau06].

■ **(4.4∗3)** We will refer to an element of $[\![\mathbb{ML}_\varnothing]\!]_{\mathcal{U}_\P}$ as a *syntactic universe*; recall from **(3.7∗3)** that this is a syntactic type $\mathsf{tp} : \mathcal{U}_\P$ together with a family of syntactic types $\mathsf{tm} : \mathsf{tp} \to \mathcal{U}_\P$.

■ **(4.4∗4)** *Universe of computability structures.* Let $(\mathsf{tp},\mathsf{tm}) : [\![\mathbb{ML}_\varnothing]\!]_{\mathcal{U}_\P}$ be a syntactic universe; for each strong universe $\mathcal{V} < \mathcal{W}$ we may define an element of $[\![\mathbb{ML}_\varnothing]\!]_{\mathcal{W}}$ aligned over $(\mathsf{tp},\mathsf{tm})$ as follows:

$$\mathsf{tp}^\star_\mathcal{V} : \{\mathcal{W} \mid \P \hookrightarrow \mathsf{tp}\}$$
$$\mathsf{tp}^\star_\mathcal{V} = \mathsf{realign}\ \left[\textstyle\sum_{A:\mathsf{tp}} \{\mathcal{V} \mid \P \hookrightarrow \mathsf{tm}(A)\} \mid \P \hookrightarrow (\mathsf{tp}, \lambda A : \mathsf{tp}.(A, \mathsf{tm}(A))) \right]$$
$$\mathsf{tm}^\star_\mathcal{V}(A) = \lceil A \rceil.2$$

The realignment above is possible because the extent type $\{\mathcal{V} \mid \P \hookrightarrow \mathsf{tm}(A)\}$ restricts to a singleton under $\P$.

⊽ **(4.4∗5)** In the future, we will not explicitly write the realignment; instead we will write definitions like **(4.4∗4)** in the following style:

> **record** $\mathsf{tp}^\star_\mathcal{V} : \{\mathcal{W} \mid \P \hookrightarrow \mathsf{tp}\}$ **where**
> $\mathsf{syn} : \mathsf{tp}$
> $\mathsf{ext} : \{\mathcal{V} \mid \P \hookrightarrow \mathsf{tm}(\mathsf{syn})\}$

> $\mathsf{tm}^\star_\mathcal{V} : \{\mathsf{tp}^\star_\mathcal{V} \to \mathcal{W} \mid \P \hookrightarrow \mathsf{tm}\}$
> $\mathsf{tm}^\star_\mathcal{V}(A) = A.\mathsf{ext}$

Given $A : \mathsf{tp}^\star_\mathcal{V}$, we will write A to mean $\mathsf{tm}^\star_\mathcal{V}(A)$.

⊽ **(4.4∗6)** We may write $\Omega_{\backslash\P}$ for the classifier of ●-modal propositions, *i.e.* the ones of the form $●\phi := \phi \vee \P$.

■ **(4.4∗7)** *Semantic realignment.* The computability universe $\mathsf{tp}^\star_\mathcal{V}$ is $\Omega_{\backslash\P}$-strong, *i.e.* supports realignment for any proposition of the form $●\phi$.

Proof. Fix a type $B : \mathsf{tp}^\star_\mathcal{V}$ and a partial isomorph $A : \{●\phi\}\,\mathsf{tp}^\star_\mathcal{V}$ of B. We may define the realignment $\mathsf{realign}\ [B \mid ●\phi \hookrightarrow A] : \mathsf{tp}^\star_\mathcal{V}$ as follows:

$$\mathsf{realign}\ [B \mid ●\phi \hookrightarrow A].\mathsf{syn} = A$$
$$\mathsf{realign}\ [B \mid ●\phi \hookrightarrow A].\mathsf{ext} = \mathsf{realign}\ [B.\mathsf{ext} \mid ●\phi \hookrightarrow A.\mathsf{ext}] \qquad \square$$

§4.4.1. Lifting negative types

▨ **(4.4.1∗1)** *Lifting dependent products.* Suppose that the syntactic universe $(\mathsf{tp},\mathsf{tm})$ is closed under dependent products in the sense that there exist constants of the following types:

$$\Pi : \left(\textstyle\sum_{\mathsf{A:tp}}(\mathsf{tm}(\mathsf{A}) \to \mathsf{tp})\right) \to \mathsf{tp}$$
$$\lambda : \{\mathsf{A},\mathsf{B}\} \left(\textstyle\prod_{x:\mathsf{tm}(\mathsf{A})}\mathsf{tm}(\mathsf{B}(x))\right) \cong \mathsf{tm}(\Pi(\mathsf{A},\mathsf{B}))$$

Then we may define a dependent product structure aligned *strictly* over (Π,λ):

$$\Pi_{\mathcal{V}}^{\star} : \left\{\left(\textstyle\sum_{\mathsf{A:tp}_{\mathcal{V}}^{\star}}(\mathsf{A} \to \mathsf{tp}_{\mathcal{V}}^{\star})\right) \to \mathsf{tp}_{\mathcal{V}}^{\star} \mid \P \hookrightarrow \Pi\right\}$$
$$\Pi_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}).\mathsf{syn} = \Pi(\mathsf{A},\mathsf{B})$$
$$\Pi_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}).\mathsf{ext} = \mathsf{realign}\left[\textstyle\prod_{x:\mathsf{A}}\mathsf{B}(x) \mid \P \hookrightarrow (\mathsf{tm}(\Pi(\mathsf{A},\mathsf{B})), \lambda\{\mathsf{A},\mathsf{B}\})\right]$$

The constant $\lambda_{\mathcal{V}}^{\star} : \{\{\mathsf{A},\mathsf{B}\} \left(\textstyle\prod_{x:\mathsf{A}}\mathsf{B}(x)\right) \cong \Pi_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}) \mid \P \hookrightarrow \lambda\}$ is *precisely* the realignment isomorphism induced by the clause $\Pi_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}).\mathsf{ext}$.

⤓ **(4.4.1∗2)** A code $\mathsf{A}^{\star} : \{\mathsf{tp}_{\mathcal{V}}^{\star} \mid \P \hookrightarrow \mathsf{A}\}$ is completely determined by its collection of elements $\mathsf{A} : \{\mathcal{V} \mid \P \hookrightarrow \mathsf{tm}(\mathsf{A})\}$; furthermore, in keeping with **(4.4∗5)**, we may suppress the realignment by a canonical isomorphism. Hence, we can impose the following notation for the definition of $\Pi_{\mathcal{V}}^{\star}$ without sacrificing any precision:

$$\Pi_{\mathcal{V}}^{\star} : \left\{\left(\textstyle\sum_{\mathsf{A:tp}_{\mathcal{V}}^{\star}}(\mathsf{A} \to \mathsf{tp}_{\mathcal{V}}^{\star})\right) \to \mathsf{tp}_{\mathcal{V}}^{\star} \mid \P \hookrightarrow \Pi\right\}$$
$$\Pi_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}).\mathsf{syn} = \Pi(\mathsf{A},\mathsf{B})$$
$$\Pi_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}).\mathsf{ext} \cong \textstyle\prod_{x:\mathsf{A}}\mathsf{B}(x)$$

▨ **(4.4.1∗3)** *Lifting dependent sums.* Let the syntactic universe $(\mathsf{tp},\mathsf{tm})$ be closed under dependent sums in the sense that there exist constants of the following types:

$$\Sigma : \left(\textstyle\sum_{\mathsf{A:tp}}(\mathsf{tm}(\mathsf{A}) \to \mathsf{tp})\right) \to \mathsf{tp}$$
$$\mathsf{pair} : \{\mathsf{A},\mathsf{B}\} \left(\textstyle\sum_{x:\mathsf{tm}(\mathsf{A})}\mathsf{tm}(\mathsf{B}(x))\right) \cong \mathsf{tm}(\Sigma(\mathsf{A},\mathsf{B}))$$

We may define a dependent sum structure on $\mathsf{tp}_{\mathcal{V}}^{\star}$ strictly aligned over (Σ,pair):

$$\Sigma_{\mathcal{V}}^{\star} : \left\{\left(\textstyle\sum_{\mathsf{A:tp}_{\mathcal{V}}^{\star}}(\mathsf{A} \to \mathsf{tp}_{\mathcal{V}}^{\star})\right) \to \mathsf{tp}_{\mathcal{V}}^{\star} \mid \P \hookrightarrow \Sigma\right\}$$
$$\Sigma_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}).\mathsf{syn} = \Sigma(\mathsf{A},\mathsf{B})$$
$$\Sigma_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}).\mathsf{ext} \cong \textstyle\sum_{x:\mathsf{A}}\mathsf{B}(x)$$

The realignment isomorphism implicitly introduced above above induces the correctly aligned pairing constant:

$$\mathsf{pair}_{\mathcal{V}}^{\star} : \{\{\mathsf{A},\mathsf{B}\} \left(\textstyle\sum_{x:\mathsf{A}}\mathsf{B}(x)\right) \cong \Sigma_{\mathcal{V}}^{\star}(\mathsf{A},\mathsf{B}) \mid \P \hookrightarrow \mathsf{pair}\}$$

▨ **(4.4.1∗4)** *Lifting equality types.* Let the syntactic universe (tp,tm) be closed under equality types in the sense that there exist constants of the following types:

$$\mathsf{eq} : \left(\textstyle\sum_{A:\mathsf{tp}} \mathsf{tm}(A) \times \mathsf{tm}(A) \right) \to \mathsf{tp}$$
$$\mathsf{refl} : \{A, a_0, a_1\} \; (a_0 =_{\mathsf{tm}(A)} a_1) \cong \mathsf{tm}(\mathsf{eq}(A, a_0, a_1))$$

We may define an equality type structure on $\mathsf{tp}_{\mathcal{V}}^{\star}$ aligned over (eq,refl):

$$\mathsf{eq}^{\star} : \left(\textstyle\sum_{A:\mathsf{tp}^{\star}} A \times A \right) \to \mathsf{tp}^{\star}$$
$$\mathsf{eq}^{\star}(A, a_0, a_1).\mathsf{syn} = \mathsf{eq}(A, a_0, a_1)$$
$$\mathsf{eq}^{\star}(A, a_0, a_1).\mathsf{ext} \cong (a_0 =_A a_1)$$

The constant $\mathsf{refl}_{\mathcal{V}}^{\star}$ is induced by the realignment isomorphism above.

§4.4.2. Lifting positive types

※ **(4.4.2∗1)** While the lifting of "negative" types is always trivial, creativity is required when lifting a positive type to computability structures. This is because positive types in dependent type theory rarely have strict universal properties — hence their liftings are not unique up to unique isomorphism; instead one chooses a lifting that will facilitate the proof of the desired metatheorem. In this section, we provide a number of tools for lifting positive types that can be pulled off the shelf later on.

▨ **(4.4.2∗2)** *Lifting an answer type.* Let $\mathsf{ans} : \mathsf{tp}$ be a syntactic type, and let $\mathsf{yes}, \mathsf{no} : \mathsf{tm}(\mathsf{ans})$ be two constants; we may define a computability structure $\mathsf{ans}_{\mathcal{V}}^{\star}$ over ans that can be used to prove a *canonicity* property for the answer type.

$$\textbf{record } \mathbf{ans}_{\mathcal{V}} : \{\mathcal{V} \mid \P \hookrightarrow \mathsf{tm}(\mathsf{ans})\} \textbf{ where}$$
$$\mathsf{syn} : \mathsf{tm}(\mathsf{ans})$$
$$\mathsf{sem} : \bullet((\mathsf{syn} = \mathsf{yes}) + (\mathsf{syn} = \mathsf{no}))$$

$$\mathsf{ans}_{\mathcal{V}}^{\star} : \{\mathsf{tp}_{\mathcal{V}}^{\star} \mid \P \hookrightarrow \mathsf{ans}\}$$
$$\mathsf{ans}_{\mathcal{V}}^{\star}.\mathsf{syn} = \mathsf{ans}$$
$$\mathsf{ans}_{\mathcal{V}}^{\star}.\mathsf{ext} = \mathbf{ans}_{\mathcal{V}}$$

$$\mathsf{yes}_{\mathcal{V}}^{\star} : \{\mathsf{ans}_{\mathcal{V}}^{\star} \mid \P \hookrightarrow \mathsf{yes}\}$$
$$\mathsf{yes}_{\mathcal{V}}^{\star}.\mathsf{syn} = \mathsf{yes}$$
$$\mathsf{yes}_{\mathcal{V}}^{\star}.\mathsf{sem} = \eta_{\backslash\P}(\mathsf{inl}(\star))$$

$$\mathsf{no}_{\mathcal{V}}^{\star} : \{\mathsf{ans}_{\mathcal{V}}^{\star} \mid \P \hookrightarrow \mathsf{no}\}$$
$$\mathsf{no}_{\mathcal{V}}^{\star}.\mathsf{syn} = \mathsf{no}$$
$$\mathsf{no}_{\mathcal{V}}^{\star}.\mathsf{sem} = \eta_{\backslash\P}(\mathsf{inr}(\star))$$

※ **(4.4.2∗3)** The computability structure of an answer type **(4.4.2∗2)** is the first time we have used the *closed* modality **(3.6∗7)**; without wrapping the computability data in $\bullet$, the realignment above would not be possible! Under the assumption of $z : \P$, however, the second component of the record becomes a singleton.

▨ **(4.4.2∗4)** *Lifting an inductive type.* We will show that the lifting of an answer type **(4.4.2∗2)** extends to the case that the answer type includes an induction principle. Fix a syntactic

induction form of the following type:

$$\text{ind} : \prod_{C:\text{tm}(\text{ans})\to\text{tp}}\prod_{c_0:\text{tm}(C(\text{yes}))}\prod_{c_1:\text{tm}(C(\text{no}))}\prod_{x:\text{tm}(\text{ans})}\text{tm}(C(x))$$

We can program an induction form for the lifted answer type aligned directly over ind:

$$\text{ind}^\star_{\mathcal{V}} : \{\prod_{C:\text{ans}^\star_{\mathcal{V}}\to\text{tp}^\star_{\mathcal{V}}}\prod_{c_0:C(\text{yes}^\star_{\mathcal{V}})}\prod_{c_1:C(\text{no}^\star_{\mathcal{V}})}\prod_{x:\text{ans}^\star_{\mathcal{V}}}C(x) \mid \P \hookrightarrow \text{ind}\}$$

$$\text{ind}^\star_{\mathcal{V}}(C,c_0,c_1,x) = \text{try } x.\text{sem } [\text{case}\{\text{inl}(\_) \hookrightarrow c_0, \text{inr}(\_) \hookrightarrow c_1\} \mid \P \hookrightarrow \text{ind}(C,c_0,c_1,x)]$$

If the syntactic induction form ind satisfies its β-laws, then so does the lifted induction form $\text{ind}^\star_{\mathcal{V}}$; likewise, if ind satisfies a unicity principle (η-law), then so does the lifted induction form.

§4.4.3. Lifting hierarchies of strict universes

(4.4.3∗1) Let **L** be the partial order $\mathbb{N} \cup \{\diamond\}$ where $\diamond > n$ for all $n \in \mathbb{N}$. In this section, we will fix an **L**-indexed hierarchy of syntactic universes assuming a $\mathcal{U}_{\P}$-valued algebra M for the following signature:

$$\text{tp}_\alpha : \square$$
$$\text{tm}_\alpha : \text{tp}_\alpha \to \square$$
$$\langle\uparrow^\beta_\alpha\rangle : \text{tp}_\alpha \to \text{tp}_\beta \quad \text{for each } \alpha \leq \beta : \mathbf{L}$$
$$\_ : \{A\} \; \langle\uparrow^\alpha_\alpha\rangle A =_{\text{tp}_\alpha} A$$
$$\_ : \{A\} \; \langle\uparrow^\gamma_\beta\rangle\langle\uparrow^\beta_\alpha\rangle A =_{\text{tp}_\gamma} \langle\uparrow^\gamma_\alpha\rangle A$$
$$\_ : \{A\} \; \text{tm}_\alpha(A) =_\square \text{tm}(\langle\uparrow^\diamond_\alpha\rangle A)$$
$$\_ : \{A,B\} \; \langle\uparrow^\beta_\alpha\rangle A =_{\text{tp}_\beta} \langle\uparrow^\beta_\alpha\rangle B \to A =_{\text{tp}_\alpha} B$$

$$\mathsf{U}_n : \text{tp}_{n+1}$$
$$\_ : \text{tm}_{n+1}(\mathsf{U}_n) =_\square \text{tp}_n$$

We will define computability structures aligned over the above, assuming strong universes $\mathcal{V}_\alpha$ for each $\alpha : \mathbf{L}$ such that $\mathcal{U} \leq \mathcal{V}_\alpha \leq \mathcal{V}_\beta < \mathcal{W}$ for all $\alpha \leq \beta$. For convenience, we will extend the $\mathcal{V}_\bullet$ hierarchy by defining $\mathcal{V}_{\mathsf{s}(\diamond)} := \mathcal{W}$.

(4.4.3∗2) We define the computability structure for each level of the typehood judgment à la **(4.4∗4)**; in particular, we have:

record $\text{tp}^\star_\alpha : \{\mathcal{V}_{\mathsf{s}(\alpha)} \mid \P \hookrightarrow \text{tp}_\alpha\}$ **where** $\text{tm}^\star_\alpha : \{\text{tp}^\star_\alpha \to \mathcal{V}_\alpha \mid \P \hookrightarrow \text{tm}_\alpha\}$
   syn : tp_α $\text{tm}^\star_\alpha(A) = A.\text{ext}$
   ext : $\{\mathcal{V}_\alpha \mid \P \hookrightarrow \text{tm}_\alpha(A)\}$

As a notational convenience, we will write A for $\text{tm}^\star_\alpha(A) = A.\text{ext}$.

◻ **(4.4.3∗3)** We may define the computability structure of the coherent lifting coercions, using the corresponding (implicit) coercions for the universes $\mathcal{V}_\alpha$:

$$\langle \uparrow_\alpha^\beta \rangle^\star : \{ \mathsf{tp}_\alpha^\star \to \mathsf{tp}_\beta^\star \mid \P \hookrightarrow \langle \uparrow_\alpha^\beta \rangle \}$$

$$\langle \uparrow_\alpha^\beta \rangle^\star \mathsf{A.syn} = \langle \uparrow_\alpha^\beta \rangle \mathsf{A}$$

$$\langle \uparrow_\alpha^\beta \rangle^\star \mathsf{A.ext} = \mathsf{A}$$

◻ **(4.4.3∗4)** *Codes for universes.* We define the computability structure of universes:

$$\mathsf{U}_n^\star : \{ \mathsf{tp}_{n+1}^\star \mid \P \hookrightarrow \mathsf{U}_n \}$$

$$\mathsf{U}_n^\star.\mathsf{syn} = \mathsf{U}_n$$

$$\mathsf{U}_n^\star.\mathsf{ext} = \mathsf{tp}_n^\star$$

The correct alignment of the above depends on the fact that $\mathsf{tp}_n = \mathsf{tm}_{n+1}(\mathsf{U}_n)$.

◻ **(4.4.3∗5)** *Coherent connectives.* Just as in **(3.4∗5)**, we may close each universe $\mathsf{tp}_\alpha^\star$ under codes for connectives that commute strictly with the lifting coercions $\langle \uparrow_\alpha^\beta \rangle^\star$, lying strictly over the syntactic codes that we assume in tp_α. By **(4.4.1∗1)** each universe is closed (incoherently) under dependent product types by some code $\Pi_\alpha^\star$; by the semantic realignment lemma **(4.4∗7)**, we may realign $\Pi_\alpha^\star(\mathsf{A},\mathsf{B})$ against the following ●-modal proposition:

$$\bullet \bigvee_{n<\alpha} \exists \mathsf{A}',\mathsf{B}'.\mathsf{A} = \langle \uparrow_n^\alpha \rangle^\star \mathsf{A}' \wedge \mathsf{B} = \lambda x.\langle \uparrow_n^\alpha \rangle^\star \mathsf{B}'(x)$$

The partial isomorph of $\Pi_\alpha^\star(\mathsf{A},\mathsf{B})$ against which we realign is then defined piecewise in two cases: underneath $\P$ we choose the syntactic code $\Pi_\alpha(\mathsf{A},\mathsf{B})$, and underneath each $n < \alpha$ we apply the lift coercion $\langle \uparrow_n^\alpha \rangle$ to the inductively determined dependent product code applied to the uniquely determined A',B' such that $\mathsf{A} = \langle \uparrow_n^\alpha \rangle \mathsf{A}'$ and $\mathsf{B} = \lambda x.\langle \uparrow_n^\alpha \rangle \mathsf{B}'(x)$. The realignment can be seen to be well-defined using our assumption that the syntactic codes commute with the syntactic lift coercions.

◆ **§4.5. CANONICITY FOR MARTIN-LÖF TYPE THEORY**

(4.5∗1) We will now develop the canonicity proof for Martin-Löf's type theory, as defined in Fig. 1.2; if it was somewhat technical to close the category $\mathcal{G}$ of computability structures under exponentials **(4.2∗10)**, it would be *significantly* worse to manually define the computability structure of the judgment $\mathsf{tp} : \mathcal{T}$ as well as its closure under dependent products and dependent sums, to the point of being nearly intractable if a suitable level of precision is maintained.

❦ **(4.5∗2)** Although Coquand [Coq19] has sketched an explicit construction of the computability structure of the judgments of type theory as well as their closure under connectives at

the level of raw sets, we will promote a significantly simpler and more abstract approach that relies only on the closure of Grothendieck topoi under Artin gluing [AGV72] as well as certain results about universes in logoi detailed in Chapter 3 — all of which can be proved more easily than any of their type theoretic corollaries. Our approach, which we will later abstract into *synthetic Tait computability*, was pioneered by the author in several recent papers [SA21; SH21].

While Coquand works directly at the level of sets, thus incurring an avalanche of mostly unchecked but apparently trivial naturality and (dependent) functoriality obligations, the essence of our approach will be to exploit the internal language of a category formed by Artin gluing — reducing the canonicity result to a number of reusable constructions on universes over a topos that can be developed modularly in naïve type theoretic language.

§4.5.1. Exploring the computability topos

(4.5.1∗1) Let $\mathcal{T}$ be the category of judgments of Martin-Löf's type theory; we are trying to prove *closed* canonicity as in **(4.1∗4)**, so we consider the trivial figure shape $\mathbf{1_{Cat}} \longrightarrow \mathcal{T}$ determined by the empty context $\mathbf{1}_{\mathcal{T}}$. Noting that $\widehat{\mathbf{1}_{\mathcal{C}}} \simeq \mathbb{1}$, we see that the morphism of topoi corresponding to the figure shape $\mathbf{1_{Cat}} \longrightarrow \mathcal{T}$ is a topos theoretic point $\mathbf{0}_{\widehat{\mathcal{T}}} : \mathbb{1} \longrightarrow \widehat{\mathcal{T}}$.

(4.5.1∗2) The point $\mathbf{0}_{\widehat{\mathcal{T}}} : \mathbb{1} \longrightarrow \widehat{\mathcal{T}}$ can be seen under inverse image to correspond to the global sections functor, *i.e.* $\mathbf{0}^*_{\widehat{\mathcal{T}}}(E) = \mathrm{Hom}_{\widehat{\mathcal{T}}}(\mathbf{1}, E)$. On the other hand, the direct image $(\mathbf{0}_{\widehat{\mathcal{T}}})_* : \mathbf{Set} \longrightarrow \mathrm{Pr}(\mathcal{T})$ may be computed by adjointness at each set S:

$$(\mathbf{0}_{\widehat{\mathcal{T}}})_* S(X) \cong \mathrm{Hom}_{\mathrm{Pr}(\mathcal{T})}\big(y(X), (\mathbf{0}_{\widehat{\mathcal{T}}})_* S\big) \cong \mathrm{Hom}_{\mathbf{Set}}\big(\mathbf{0}^*_{\widehat{\mathcal{T}}} y(X), S\big) \cong \mathrm{Hom}_{\mathbf{Set}}\big(\mathrm{Hom}_{\mathcal{T}}(\mathbf{1}, X), S\big)$$

We have already seen the global sections functor in the guise of the *direct image* part of the terminal map $\widehat{\mathcal{T}} \longrightarrow \mathbb{1}$; the existence of a point whose inverse image corresponds with the direct image of a different geometric morphism will provide us with an alternative (and perhaps more familiar!) computation of the computability topos in **(4.5.1∗5)**.

(4.5.1∗3) The point $\mathbf{0}_{\widehat{\mathcal{T}}} : \mathbb{1} \longrightarrow \widehat{\mathcal{T}}$ has a universal property: it is the *initial* object in the category of points $\mathrm{Hom}_{\mathbf{Topos}}(\mathbb{1}, \widehat{\mathcal{T}})$. Recalling the perspective of classifying topoi and presheaves on left exact categories **(2.3.1∗5)**, the initial point $\mathbf{0}_{\widehat{\mathcal{T}}}$ is then the *initial model* of $\mathcal{T}$ regarded as a finite limit theory. But what does this mean?

Recall that as a category of judgments, $\mathcal{T}$ is a locally Cartesian closed category; when the structure of dependent products is forgotten, one obtains a finite limit theory whose models are "nonstandard" models of Martin-Löf type theory in which hypothetical judgments may not be interpreted by dependent products of sets. It is historically appropriate to refer to such a nonstandard model as a *Henkin model* of $\mathcal{T}$; hence, $\widehat{\mathcal{T}}$ is the classifying topos of Henkin models of Martin-Löf's type theory, and $\mathbf{0}_{\widehat{\mathcal{T}}}$ corresponds under inverse image to the Henkin model obtained by taking closed terms at every sort.

record $\mathbb{ML}_{base}$: $\mathfrak{Sig}$ **where**
$\quad \mathsf{M}_\alpha : \mathbb{ML}_\varnothing \quad$ for each $\alpha : \mathbf{L}$
$\quad \mathsf{tp}_\alpha,\mathsf{tm}_\alpha := \mathsf{M}_\alpha.\mathsf{tp},\mathsf{M}_\alpha.\mathsf{tm}$
$\quad \mathsf{tp},\mathsf{tm} := \mathsf{tp}_\diamond,\mathsf{tm}_\diamond$

$\quad \langle\uparrow_\alpha^\beta\rangle : \mathsf{tp}_\alpha \to \mathsf{tp}_\beta \quad$ for each $\alpha \le \beta$
$\quad \_\_ : \{A\} \ \langle\uparrow_\alpha^\alpha\rangle A =_{\mathsf{tp}_\alpha} A$
$\quad \_\_ : \{A\} \ \langle\uparrow_\beta^\gamma\rangle\langle\uparrow_\alpha^\beta\rangle A =_{\mathsf{tp}_\gamma} \langle\uparrow_\alpha^\gamma\rangle A$
$\quad \_\_ : \{A\} \ \mathsf{tm}_\alpha(A) =_\square \mathsf{tm}(\langle\uparrow_\alpha^\diamond\rangle A)$
$\quad \_\_ : \{A,B\} \ \langle\uparrow_\alpha^\beta\rangle A =_{\mathsf{tp}_\beta} \langle\uparrow_\alpha^\beta\rangle B \to A =_{\mathsf{tp}_\alpha} B$

$\quad$ **include** $\mathbb{J}_{\mathsf{bool}}(\mathsf{M}_\diamond)$
$\quad$ **include** $\prod_{F:\mathsf{fam}(\mathsf{M}_\diamond)}\mathbb{J}_\Pi(\mathsf{M}_\diamond,F)$
$\quad$ **include** $\prod_{F:\mathsf{fam}(\mathsf{M}_\diamond)}\mathbb{J}_\Sigma(\mathsf{M}_\diamond,F)$
$\quad \mathsf{bool}_\diamond,\Pi_\diamond,\Sigma_\diamond := \mathsf{bool},\Pi,\Sigma$

$\quad \mathsf{U}_n : \mathsf{tp}_{n+1} \quad$ for each $n < \diamond$
$\quad \_\_ : \mathsf{tm}_{n+1}(\mathsf{U}_m) =_\square \mathsf{tp}_m$

$\quad \Pi_n,\Sigma_n : \mathsf{fam}(\mathsf{tp}_n,\mathsf{tm}_n) \to \mathsf{tp}_n \quad$ for each $n < \diamond$
$\quad \mathsf{bool}_0 : \mathsf{tp}_0$

$\quad \_\_ : \{A,B\} \ \langle\uparrow_\alpha^\beta\rangle\Pi_n(A,B) =_{\mathsf{tp}_\beta} \Pi_\beta(\langle\uparrow_\alpha^\beta\rangle A,\langle\uparrow_\alpha^\beta\rangle \circ B)$
$\quad \_\_ : \{A,B\} \ \langle\uparrow_\alpha^\beta\rangle\Sigma_n(A,B) =_{\mathsf{tp}_\beta} \Sigma_\beta(\langle\uparrow_\alpha^\beta\rangle A,\langle\uparrow_\alpha^\beta\rangle \circ B)$

record $\mathbb{ML}_{ext}$: $\mathfrak{Sig}$ **where**
$\quad$ **include** $\mathbb{ML}_{base}$
$\quad$ **include** $\prod_{A:\mathsf{tp};a_0,a_1:\mathsf{tm}(A)}\mathbb{J}_{\mathsf{eq}}(\mathsf{M}_\diamond,A,a_0,a_1)$
$\quad \mathsf{eq}_\diamond := \mathsf{eq}$
$\quad \mathsf{eq}_n : \left(\sum_{A:\mathsf{tp}_n}\mathsf{tm}_nA \times \mathsf{tm}_nA\right) \to \mathsf{tp}_n \quad$ for each $n < \diamond$
$\quad \_\_ : \{A,a_0,a_1\} \ \langle\uparrow_\alpha^\beta\rangle\mathsf{eq}_\alpha(A,a_0,a_1) =_{\mathsf{tp}_\beta} \mathsf{eq}_\beta(\langle\uparrow_\alpha^\beta\rangle A,a_0,a_1)$

Each universe is closed under the booleans as well as all the smaller universes:

$$\mathsf{bool}_\alpha : \mathsf{tp}_\alpha \qquad\qquad \mathsf{U}_{n,\alpha} : \mathsf{tp}_\alpha \quad \text{for each } n < \alpha$$
$$\mathsf{bool}_\alpha = \langle\uparrow_0^\alpha\rangle\mathsf{bool}_0 \qquad \mathsf{U}_{n,\alpha} = \langle\uparrow_{n+1}^\alpha\rangle\mathsf{U}_n$$

Figure 4.1: (For convenience, we repeat Fig. 1.2 from Chapter 1.)

■ **(4.5.1∗4)** The gluing for the trivial figure shape **(4.5.1∗1)** may be constructed à la **(4.3∗3)**:

$$
\begin{array}{ccc}
\mathbb{1} & \xrightarrow{\ \mathbf{0}_{\widehat{\mathcal{T}}}\ } & \widehat{\mathcal{T}} \\
{\scriptstyle \circ}\downarrow & & \downarrow{\scriptstyle j} \\
\mathbb{S} & \xrightarrow{\qquad} & \mathbf{G}
\end{array}
\tag{4.5.1∗4∗1}
$$

⩔ **(4.5.1∗5)** Using our observation that $\mathbf{0}_{\widehat{\mathcal{T}}}^{*} = \widehat{\mathcal{T}}_{*}$ from **(4.5.1∗2)**, we may give an alternative construction of the computability topos that involves gluing along the terminal map $\widehat{\mathcal{T}} \to \mathbb{1}$ rather than the initial point $\mathbf{0}_{\widehat{\mathcal{T}}} : \mathbb{1} \to \widehat{\mathcal{T}}$

$$
\begin{array}{ccc}
\widehat{\mathcal{T}} & \xrightarrow{\qquad} & \mathbb{1} \\
{\scriptstyle \bullet_{\widehat{\mathcal{T}}}}\downarrow & & \downarrow{\scriptstyle i} \\
\widehat{\mathcal{T}} \times \mathbb{S} & \xrightarrow{\qquad} & \mathbf{G}
\end{array}
\tag{4.5.1∗5∗1}
$$

To see that these constructions of **G** coincide, we consider what a sheaf is on the two pushout topoi from Diagrams 4.5.1∗4∗1 and 4.5.1∗5∗1 respectively. In the case of Diagram 4.5.1∗4∗1, a sheaf on **G** is a presheaf $E : \mathrm{Pr}(\mathcal{T})$ together with a family of sets $S \to \mathbf{0}_{\widehat{\mathcal{T}}}^{*} E$. On the other hand, in the case of Diagram 4.5.1∗5∗1, a sheaf on **G** is a set S together with a family of presheaves $\widehat{\mathcal{T}}^{*} S \to E : \mathrm{Pr}(\mathcal{T})$. These can be seen to coincide by a simple computation:

$$
\mathrm{Hom}_{\mathrm{Pr}(\mathcal{T})}(\widehat{\mathcal{T}}^{*} S, E) \cong \mathrm{Hom}_{\mathbf{Set}}(S, \widehat{\mathcal{T}}_{*} E) \cong \mathrm{Hom}_{\mathbf{Set}}(S, \mathbf{0}_{\widehat{\mathcal{T}}}^{*} E)
$$

◔ **(4.5.1∗6)** Diagrams 4.5.1∗4∗1 and 4.5.1∗5∗1 are two perspectives on the same object: the first is specified in terms of open immersions and the second is specified in terms of closed immersions. Diagram 4.5.1∗4∗1 is paradigmatic of type theoretic gluing situations (Artin gluing along an inverse image functor), but Diagram 4.5.1∗5∗1 has topos theoretic significance as the *Sierpiński cone* or *Freyd cover* [Fre78] of $\widehat{\mathcal{T}}$. Most type theoretic gluing situations cannot be described in terms of pushouts of closed immersions; closed canonicity and parametricity, as well as phase separated variants of these [SH21], are notable exceptions.

§4.5.2. The computability algebra

■ **(4.5.2∗1)** The abstract syntax of Martin-Löf type theory embeds into $\mathbf{Set}_{\mathbf{G}}$ along the following locally Cartesian closed functor:

$$
\mathcal{T} \lhook\joinrel\longrightarrow \mathrm{Pr}(\mathcal{T}) \lhook\joinrel\longrightarrow \mathbf{Set}_{\mathbf{G}}
$$

From an internal perspective, there is a universe $\mathcal{U}$ such that the interpretation above corresponds to an algebra $\mathbf{M} : [\![\mathbb{ML}_{ext}]\!]_{\mathcal{U}_{\P}}$. Our goal will be to construct a computability algebra $\mathbf{M}^{\star} : \{[\![\mathbb{ML}_{ext}]\!]_{\mathcal{W}} \mid \P \hookrightarrow \mathbf{M}\}$ for some sufficiently large $\mathcal{W}$.

▣ **(4.5.2∗2)** In $\mathbf{Set_G}$ we have a strict universe hierarchy $\mathcal{U} < \mathcal{V}_0 < \ldots < \mathcal{V}_n \ldots < \mathcal{V}_{\diamond} < \mathcal{W}$. These universes exist because we have assumed sufficiently many strongly inaccessible cardinals in the background set theory.

▣ **(4.5.2∗3)** We define the components of the computability algebra $\mathbf{M}^{\star}$ one-by-one:
1) The hierarchy of type structures M_{α} and universe codes U_m are given by **(4.4.3∗2)** through **(4.4.3∗4)** from §4.4.3.
2) The dependent product structure is given by **(4.4.1∗1)**.
3) The dependent sum structure is given by **(4.4.1∗3)**.
4) The equality type structure is given by **(4.4.1∗4)**.
5) The boolean structure is given by **(4.4.2∗2)** and **(4.4.2∗4)**.
   a) The formation and introduction structure is implemented by **(4.4.2∗2)**, setting $\mathsf{ans} := \mathsf{bool}$, $\mathsf{yes} := \mathsf{tt}$, and $\mathsf{no} := \mathsf{ff}$.
   b) The elimination structure is implemented by **(4.4.2∗4)**.
6) The strict closure of each universe level under connectives is given by **(4.4.3∗5)**.

§4.5.3. The canonicity result

▣ **(4.5.3∗1)** *Canonicity.* Let $b : \mathbb{1} \longrightarrow \mathsf{tm}(\mathsf{bool}) : \mathcal{T}$ be a closed term of boolean type; then either $b = \mathsf{tt}$ or $b = \mathsf{ff}$, meant as a statement about global elements in $\mathbf{Set}$.

Proof. The assumed closed term may be interpreted into the computability algebra **(4.5.2∗3)** as a global element $b^{\star} : \mathbf{1}_{\mathbf{Set_G}} \longrightarrow \{\mathsf{tm}_{\diamond}^{\star}(\mathsf{bool}^{\star}) \mid \P \hookrightarrow b\}$. Unfolding the computability structure of the booleans from **(4.4.2∗2)**, we have a global element of the modal type $\bullet((b = \mathsf{tt}) + (b = \mathsf{ff}))$. We compute, using the fact that i^{*} is left exact and cocontinuous:

$$\mathrm{Hom}_{\mathbf{Set_G}}(\mathbf{1}_{\mathbf{Set_G}}, \bullet((b = \mathsf{tt}) + (b = \mathsf{ff})))$$
$$= \mathrm{Hom}_{\mathbf{Set_G}}(\mathbf{1}_{\mathbf{Set_G}}, i_{*}i^{*}((b = \mathsf{tt}) + (b = \mathsf{ff}))) \qquad \text{by def.}$$
$$\cong \mathrm{Hom}_{\mathbf{Set}}(\mathbf{1}_{\mathbf{Set}}, i^{*}((b = \mathsf{tt}) + (b = \mathsf{ff}))) \qquad i^{*} \text{ left exact}$$
$$\cong \mathrm{Hom}_{\mathbf{Set}}(\mathbf{1}_{\mathbf{Set}}, i^{*}(b = \mathsf{tt}) + i^{*}(b = \mathsf{ff})) \qquad i^{*} \text{ cocontinuous}$$
$$\cong \mathrm{Hom}_{\mathbf{Set}}(\mathbf{1}_{\mathbf{Set}}, (b = \mathsf{tt}) + (b = \mathsf{ff})) \qquad i^{*} \text{ left exact} \qquad \square$$

CHAPTER 5

SYNTHETIC NORMALIZATION BY EVALUATION

This chapter contains joint work with Daniel Gratzer.

🍁 **(5.0∗1)** This chapter is a more abstract and more detailed version of the normalization argument of Coquand [Coq19], compressed into the language of synthetic Tait computability (§4.4); the construction of Coquand is itself a dependently typed generalization of the arguments of Altenkirch, Hofmann, and Streicher [AHS95] and Fiore [Fio02]. The purpose of this exposition is to lay the groundwork for developing the more complex normalization argument for cubical type theory in Chapter 6.

🜄 **(5.0∗2)** In this chapter we work abstractly over a given topos **G** equipped with an open ¶ : $\mathcal{O}_{\mathbf{G}}$, as well as a transfinite and strict hierarchy of universes $\mathcal{U} \le \mathcal{V}_\alpha < \mathcal{W}$ for $\alpha : \mathbf{L}$,

defining $\mathcal{V}_{s(\diamond)} := \mathcal{W}$. In §5.5, we will explicitly construct a topos whose logos models the synthetic Tait computability of normal forms developed in §§5.2 and 5.3.

§5.1. WHAT IS NORMALIZATION?

※ **(5.1∗1)** Normalization means many things to many people; one common view of normalization is that it is a property of a rewriting system, in which case it is often qualified as either "weak" or "strong". Strong normalization *qua* rewriting is the property that every sequence of rewrites eventually results in an term that can be reduced no further. Of course, normalization in this sense is not a property of a *type theory*, but of a presentation of a type theory by a rewriting system. The purpose of presenting a type theory by a confluent and strongly normalizing rewriting system is to achieve the following desirable properties:
1) The decidability of judgmental equality.
2) The admissible injectivity of type constructors: for example, to deduce $\Gamma \vdash A = A'$ from $\Gamma \vdash (A \to B) = (A' \to B')$.

Modern type theory has unfortunately resisted the application of rewriting theoretic techniques; in part, this is because rewriting appears to be poorly adapted to handling extensionality and unicity laws. For this reason, *reduction-free* approaches to normalization have been increasingly used by type theorists such as normalization by evaluation [Abe09; Abe13; AAD07; AVW17; AMB13; Alt+01; GSB19a; WB18].

Because strongly normalizing confluent rewriting systems that are both sound and complete for modern type theories are somewhat rare and exceptional, we take a broad view of normalization that does not constrain the presentation artificially. We take a "normalization result" to be any result that equips a type theory with a presentation for which it is *trivial* to establish the desired properties (including decidability of the word problem, as well as various inversion principles). In practice, the resulting "normal forms" may not be fully syntactic: for instance, normal forms for type theory with coproducts are decision trees quotiented under certain permutations [Alt+01].

❀ **(5.1∗2)** There is even precedent to consider notions of normal form that are *not* decidable, such as the infinitary normal forms employed by Zeilberger [Zei08; Zei09] in his higher-order analysis of focusing. Zeilberger's generalization of Schütte's ω-rule [Sch50] is spiritually very similar to our account of binders for both the interval and for witnesses of cofibrations, except that the latter can be seen to be finitary from an external perspective.

※ **(5.1∗3)** Our method will be to axiomatize in synthetic Tait computability a notion of normal form, and then prove that under the assumption of various normal form constants, that the generic $\mathcal{T}$-algebra admits a sound and complete translation into these normal forms. Then when substantiating the axioms of this synthetic Tait computability with a topos model, we ensure that the assumed normal form constants can be constructed

in such a way that they have the appropriate properties (inversion laws and decidable equality) *externally*, *i.e.* when viewed from **Set**.

§5.2. SYNTHETIC NORMAL FORMS

(5.2∗1) Let $\mathbf{L}$ be the partial order $\mathbb{N} \cup \{\diamond\}$ with $\diamond > n$ as in §4.4.3. In this section, we assume a $\mathcal{U}_\P$-valued algebra for the following signature:

$$\mathsf{tp}_\alpha : \square$$
$$\mathsf{tm}_\alpha : \mathsf{tp}_\alpha \to \square$$
$$\langle\uparrow_\alpha^\beta\rangle : \mathsf{tp}_\alpha \to \mathsf{tp}_\beta \quad \text{for each } \alpha \leq \beta$$
$$\_ : \{A\} \; \langle\uparrow_\alpha^\alpha\rangle A =_{\mathsf{tp}_\alpha} A$$
$$\_ : \{A\} \; \langle\uparrow_\beta^\gamma\rangle\langle\uparrow_\alpha^\beta\rangle A =_{\mathsf{tp}_\gamma} \langle\uparrow_\alpha^\gamma\rangle A$$
$$\_ : \{A\} \; \mathsf{tm}_\alpha(A) =_\square \mathsf{tm}(\langle\uparrow_\alpha^\diamond\rangle A)$$
$$\_ : \{A,B\} \; \langle\uparrow_\alpha^\beta\rangle A =_{\mathsf{tp}_\beta} \langle\uparrow_\alpha^\beta\rangle B \to A =_{\mathsf{tp}_\alpha} B$$

$$\mathsf{U}_m : \mathsf{tp}_{m+1}$$
$$\_ : \mathsf{tm}_{m+1}(\mathsf{U}_m) =_\square \mathsf{tp}_m$$

Further assumptions (such as closure under connectives) will be made locally.

(5.2∗2) Next we assume collections of normal forms for the basic sorts of the type theory.

$$\mathsf{nftp} : \{\mathcal{U} \mid \P \hookrightarrow \mathsf{tp}\}$$
$$\mathsf{var},\mathsf{ne},\mathsf{nf} : \textstyle\prod_{A:\mathsf{tp}}\{\mathcal{U} \mid \P \hookrightarrow \mathsf{tm}(A)\}$$

We define derived collections $\mathsf{nftp}_\alpha, \mathsf{netp}_\alpha$:

$$\mathsf{nftp}_\diamond = \mathsf{nftp}$$
$$\mathsf{nftp}_n = \mathsf{nf}(\langle\uparrow_{n+1}^\diamond\rangle\mathsf{U}_n)$$
$$\mathsf{netp}_n = \mathsf{ne}(\langle\uparrow_{n+1}^\diamond\rangle\mathsf{U}_n)$$

(5.2∗3) To populate the normal form presentation, we assume the following constants:

$$\mathbf{var} : \{A\} \; \{\mathsf{var}(A) \to \mathsf{ne}(A) \mid \P \hookrightarrow \lambda x.x\}$$
$$\mathbf{U}_n^\alpha : \{\mathsf{nftp}_\alpha \mid \P \hookrightarrow \langle\uparrow_{n+1}^\alpha\rangle\mathsf{U}_n\} \qquad\qquad \text{for } n < \alpha$$
$$\mathbf{up}_n^\alpha : \{\mathsf{netp}_n \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \langle\uparrow_n^\alpha\rangle\} \qquad\qquad \text{for } n < \alpha$$
$$\widetilde{\mathbf{up}}_n : \textstyle\prod_{A:\mathsf{netp}_n}\{\mathsf{ne}(\langle\uparrow_n^\diamond\rangle A) \to \mathsf{nf}(\langle\uparrow_n^\diamond\rangle A) \mid \P \hookrightarrow \lambda x.x\}$$

Every variable is a neutral form; the presentation of the normal forms of universes embodied in $\mathbf{U}_n^\alpha$ and $\mathbf{up}_n^\alpha$ reflects our general strategy to collate the level coercions as close to the leaves of the syntax tree as possible.

§5.3. NORMALIZATION STRUCTURES AND TAIT'S YOGA

(5.3∗1) Relative to each syntactic type system $(\mathsf{tp}_\alpha, \mathsf{tm}_\alpha)$ we have defined in **(4.4.3∗2)** a universe $\mathsf{tp}_\alpha^\star$ of *computability structures* that glues a type $A^\star : \mathcal{V}_\alpha$ together with a syntactic type $A : \mathsf{tp}_\alpha$ assuming that $A^\star$ restricts to $\mathsf{tm}_\alpha(A)$ under ¶. Inspired by Altenkirch, Hofmann, and Streicher [AHS95], Coquand [Coq19], and Tait [Tai67] we will define a universe of *normalization structures* that incorporates into the above a normal form for each type, and functions that reflect neutrals as computable elements and reify computable elements as normals.

$$
\begin{aligned}
&\textbf{record } \mathsf{tp}_\alpha^\circledast : \{\mathcal{V}_{\mathsf{s}(\alpha)} \mid \P \hookrightarrow \mathsf{tp}_\alpha\} \textbf{ where} \\
&\quad \textbf{include } \mathsf{tp}_\alpha^\star \textbf{ as } \mathsf{C} \\
&\quad \mathsf{N} : \textstyle\prod_{\beta \geq \alpha}\{\mathsf{nftp}_\beta \mid \P \hookrightarrow \langle \uparrow_\alpha^\beta \rangle \mathsf{C}\} \\
&\quad \mathord{\Uparrow} : \{\mathsf{ne}(\mathsf{C}) \to \mathsf{C} \mid \P \hookrightarrow \lambda x.x\} \\
&\quad \mathord{\Downarrow} : \{\mathsf{C} \to \mathsf{nf}(\mathsf{C}) \mid \P \hookrightarrow \lambda x.x\}
\end{aligned}
$$

(5.3∗2) The phrase "**include $\mathsf{tp}_\alpha^\star$ as C**" in the signature above is meant to expand to the more verbose signature:

$$
\begin{aligned}
&\textbf{record } \mathsf{tp}_\alpha^\circledast : \{\mathcal{V}_{\mathsf{s}(\alpha)} \mid \P \hookrightarrow \mathsf{tp}_\alpha\} \textbf{ where} \\
&\quad \textbf{include } \mathsf{tp}_\alpha^\star \\
&\quad \mathsf{C} : \{\mathsf{tp}_\alpha^\star \mid \top \hookrightarrow (\mathsf{syn},\mathsf{ext})\} \\
&\quad \mathsf{N} : \textstyle\prod_{\beta \geq \alpha}\{\mathsf{nftp}_\beta \mid \P \hookrightarrow \langle \uparrow_\alpha^\beta \rangle \mathsf{C}\} \\
&\quad \mathord{\Uparrow} : \{\mathsf{ne}(\mathsf{C}) \to \mathsf{C} \mid \P \hookrightarrow \lambda x.x\} \\
&\quad \mathord{\Downarrow} : \{\mathsf{C} \to \mathsf{nf}(\mathsf{C}) \mid \P \hookrightarrow \lambda x.x\}
\end{aligned}
$$

We will also impose the following notations for projecting the components of a normalization structure:

$$
\begin{aligned}
\mathord{\Downarrow}_{\mathsf{tp}}^\beta A &:= A.\mathsf{N}_\beta \\
\mathord{\Uparrow}_A &:= A.\mathord{\Uparrow} \\
\mathord{\Downarrow}_A &:= A.\mathord{\Downarrow}
\end{aligned}
$$

We will treat the obvious projection $\mathsf{tp}_\mathcal{V}^\circledast \to \mathsf{tp}_\mathcal{V}^\star$ implicitly in our notation when it causes no ambiguity. Symmetrically, when defining an instance of $\mathsf{tp}_\alpha^\circledast$ by "co-pattern matching", we provide a notation to extend an existing element of $\mathsf{tp}_\alpha^\star$ by the additional fields of a normalization structure. Therefore the declaration on the left should be read as

expanding to the declaration on the right, given some $X^\star : tp_\alpha^\star$:

$$X^\circledast \Leftarrow \textbf{extend } X^\star \qquad\qquad X^\circledast.\mathsf{syn} = X^\star.\mathsf{syn}$$

$$\downarrow^\beta_{\mathsf{tp}} X^\circledast = \mathbf{X}_\beta \qquad\qquad X^\circledast.\mathsf{ext} = X^\star.\mathsf{ext}$$

$$\mathord{\uparrow}_{X^\circledast} a = f(a) \qquad\qquad X^\circledast.C = X^\star$$

$$\mathord{\downarrow}_{X^\circledast} a = g(a) \qquad\qquad X^\circledast.N(\beta) = \mathbf{X}_\beta$$

$$X^\circledast.\mathord{\uparrow}(a) = f(a)$$

$$X^\circledast.\mathord{\downarrow}(a) = g(a)$$

■ **(5.3∗3)** *Coherent lifts.* The normalization structure of universe level lifts is inherited nearly directly from **(4.4.3∗3)**.

$$\boxed{\langle\mathord{\uparrow}^\beta_\alpha\rangle^\circledast : \{tp^\circledast_\alpha \to tp^\circledast_\beta \mid \P \hookrightarrow \langle\mathord{\uparrow}^\beta_\alpha\rangle\}}$$

$$\langle\mathord{\uparrow}^\beta_\alpha\rangle^\circledast A \Leftarrow \textbf{extend } \langle\mathord{\uparrow}^\beta_\alpha\rangle^\star A$$

$$\downarrow^{\gamma\geq\beta}_{\mathsf{tp}}(\langle\mathord{\uparrow}^\beta_\alpha\rangle^\circledast A) = \downarrow^{\gamma\geq\alpha}_{\mathsf{tp}} A$$

$$\mathord{\uparrow}_{\langle\mathord{\uparrow}^\beta_\alpha\rangle^\circledast A} a = \mathord{\uparrow}_A a$$

$$\mathord{\downarrow}_{\langle\mathord{\uparrow}^\beta_\alpha\rangle^\circledast A} a = \mathord{\downarrow}_A a$$

■ **(5.3∗4)** *Normalization structure of neutral types.* Let $A : \mathsf{netp}_n$ be a neutral type; we may define a normalization structure consisting of the neutral elements of A.

$$\boxed{\mathsf{elim}^\circledast_n : \{\mathsf{netp}_n \to tp^\circledast_n \mid \P \hookrightarrow \lambda A.A\}}$$

$$\mathsf{elim}^\circledast_n(A).\mathsf{syn} = A$$

$$\mathsf{elim}^\circledast_n(A).\mathsf{ext} = \mathsf{ne}(\langle\mathord{\uparrow}^\diamond_n\rangle A)$$

$$\downarrow^{\beta\geq n}_{\mathsf{tp}} \mathsf{elim}^\circledast_n(A) = \mathbf{up}^\beta_n(A)$$

$$\mathord{\uparrow}_{\mathsf{elim}^\circledast_n(A)} a = a$$

$$\mathord{\downarrow}_{\mathsf{elim}^\circledast_n(A)} a = \widetilde{\mathbf{up}}_n(A,a)$$

■ **(5.3∗5)** *Normalization structure of universes.* For each finite universe level n we may define the normalization structure of the universe U_n, making critical use of **(5.3∗4)**:

$$\boxed{U^\circledast_n : \{tp^\circledast_{n+1} \mid \P \hookrightarrow U_n\}}$$

$$U^\circledast_n.\mathsf{syn} = U_n$$

$$U^\circledast_n.\mathsf{ext} = tp^\circledast_n$$

$$\downarrow^{\beta\geq n+1}_{\mathsf{tp}} U^\circledast_n = \mathbf{U}^n_\beta$$

$$\mathord{\uparrow}_{U^\circledast_n} A = \mathsf{elim}^\circledast_n(A)$$

$$\mathord{\downarrow}_{U^\circledast_n} A = \downarrow^n_{\mathsf{tp}} A$$

■ **(5.3∗6)** *Normalization structure of dependent products.* Suppose that our type theory is closed under dependent product types and the the language of normal forms contains the following constants:

$$\mathbf{pi}_\alpha : \{(\textstyle\sum_{A:\mathsf{nftp}_\alpha}(\mathsf{var}(\langle\uparrow_\alpha^\diamond\rangle A) \to \mathsf{nftp}_\alpha)) \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \Pi_\alpha\}$$

$$\mathbf{app} : \{A,B\}\ \{\mathsf{ne}(\Pi(A,B)) \to \textstyle\prod_{x:\mathsf{nf}(A)}\mathsf{ne}(B(x)) \mid \P \hookrightarrow \lambda f.\lambda x.f(x)\}$$

$$\mathbf{lam} : \{A,B\}\ \{(\textstyle\prod_{x:\mathsf{var}(A)}\mathsf{nf}(B(x))) \to \mathsf{nf}(\Pi(A,B)) \mid \P \hookrightarrow \lambda f.\lambda x.f(x)\}$$

Then we may extend the (coherent) computability structure of dependent products **(4.4.3∗5)** to a normalization structure:

$$\boxed{\Pi_\alpha^\circledast : \{(\textstyle\sum_{A:\mathsf{tp}_\alpha^\circledast}(A \to \mathsf{tp}_\alpha^\circledast)) \to \mathsf{tp}_\alpha^\circledast \mid \P \hookrightarrow \Pi_\alpha\}}$$

$$\Pi_\alpha^\circledast(A,B) \Leftarrow \mathbf{extend}\ \Pi_\alpha^\star(A,B)$$

$$\downarrow_{\mathsf{tp}}^{\beta\geq\alpha}\Pi_\alpha^\circledast(A,B) = \mathbf{pi}_\beta(\downarrow_{\mathsf{tp}}^\beta A, \lambda x.\downarrow_{\mathsf{tp}}^\beta B(\uparrow_A\mathbf{var}(x)))$$

$$\uparrow_{\Pi_\alpha^\circledast(A,B)}f = \lambda x.\uparrow_{B(A(x))}\mathbf{app}(f,\downarrow_A x)$$

$$\downarrow_{\Pi_\mathcal{V}^\circledast(A,B)}f = \mathbf{lam}(\lambda x.\downarrow_{B(\uparrow_A\mathbf{var}(x))}f(\uparrow_A\mathbf{var}(x)))$$

■ **(5.3∗7)** *Normalization structure of dependent sums.* Suppose that our type theory is closed under dependent sum types and we have the following constants implementing their normal forms:

$$\mathbf{sg}_\alpha : \{(\textstyle\sum_{A:\mathsf{nftp}_\alpha}(\mathsf{var}(\langle\uparrow_\alpha^\diamond\rangle A) \to \mathsf{nftp}_\alpha)) \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \Sigma_\alpha\}$$

$$\mathbf{split} : \{A,B\}\ \{\mathsf{ne}(\Sigma(A,B)) \to \textstyle\sum_{x:\mathsf{ne}(A)}\mathsf{ne}(B(x)) \mid \P \hookrightarrow \lambda p.(p.1,p.2)\}$$

$$\mathbf{pair} : \{A,B\}\ \{(\textstyle\sum_{x:\mathsf{nf}(A)}\mathsf{nf}(B(x))) \to \mathsf{nf}(\Sigma(A,B)) \mid \P \hookrightarrow \lambda p.(p.1,p.2)\}$$

We may extend the (coherent) computability structure of dependent sums **(4.4.3∗5)** to a normalization structure:

$$\boxed{\Sigma_\alpha^\circledast : \{(\textstyle\sum_{A:\mathsf{tp}_\alpha^\circledast}(A \to \mathsf{tp}_\alpha^\circledast)) \to \mathsf{tp}_\alpha^\circledast \mid \P \hookrightarrow \Sigma_\alpha\}}$$

$$\Sigma_\alpha^\circledast(A,B) \Leftarrow \mathbf{extend}\ \Sigma_\alpha^\star(A,B)$$

$$\downarrow_{\mathsf{tp}}^{\beta\geq\alpha}\Sigma_\alpha^\circledast(A,B) = \mathbf{sg}_\beta(\downarrow_{\mathsf{tp}}^\beta A, \lambda x.\downarrow_{\mathsf{tp}}^\beta B(\uparrow_A\mathbf{var}(x)))$$

$$(\uparrow_{\Sigma_\alpha^\circledast(A,B)}p).1 = \uparrow_A\mathbf{split}(p).1$$

$$(\uparrow_{\Sigma_\alpha^\circledast(A,B)}p).2 = \uparrow_{B(\uparrow_A\mathbf{split}(p).1)}\mathbf{split}(p).2$$

$$\downarrow_{\Sigma_\mathcal{V}^\circledast(A,B)}p = \mathbf{pair}(\downarrow_A p.1, \downarrow_{B(p.1)}p.2)$$

■ **(5.3∗8)** *Computability structure of the open booleans.* Base types (such as abstract types, booleans, or other inductive types) do not inherit their underlying computability structures

from the canonicity argument **(4.4.2∗2)**: this is the base case for the Tait normalization argument. We must therefore define a computability structure for the booleans that incorporates the lifting from neutrals to normals.

Assuming closure of the type theory under a boolean type, we define **bool** : $\{\mathcal{U}\ |\ \P \hookrightarrow \mathsf{bool}\}$ to be the smallest type aligned over the syntactic booleans closed under the following constructors:

$$\mathbf{tt} : \{\mathbf{bool}\ |\ \P \hookrightarrow \mathsf{tt}\}$$
$$\mathbf{ff} : \{\mathbf{bool}\ |\ \P \hookrightarrow \mathsf{ff}\}$$
$$\mathbf{up}_{\mathsf{bool}}[\phi] : \{\mathsf{ne}(\mathsf{bool}) \to \mathbf{bool}\ |\ \P \hookrightarrow \lambda x.x\}$$

We then define the computability structure of the open booleans as follows:

$$\boxed{\mathsf{bool}_\alpha^\star : \{\mathsf{tp}_\alpha^\star\ |\ \P \hookrightarrow \mathsf{bool}_\alpha\}}$$
$$\mathsf{bool}_\alpha^\star.\mathsf{syn} = \mathsf{bool}_\alpha$$
$$\mathsf{bool}_\alpha^\star.\mathsf{ext} = \mathbf{bool}$$

(5.3∗9) To execute the inductive definition described in **(5.3∗8)**, we actually define an indexed inductive family $x : \mathsf{bool} \vdash \mathbf{bool}[x] : \mathcal{U}_{\backslash\P}$ of ●-modal types, and then realign the dependent sum $\sum_{x:\mathsf{bool}} \mathbf{bool}[x]$ as in **(4.4.2∗2)**. Inductive definitions of ●-modal types are *quotient*-inductive definitions of ordinary types in which a constructor is added that collapses the type to a point underneath ¶.

(5.3∗10) *Normalization structure of booleans.* We extend the computability structure **(5.3∗8)** to a normalization structure; first we assume the following constants in the language of normal forms:

$$\mathbf{bool}_\alpha : \{\mathsf{nftp}_\alpha\ |\ \P \hookrightarrow \mathsf{bool}_\alpha\}$$
$$\mathbf{tt} : \{\mathsf{nf}(\mathsf{bool})\ |\ \P \hookrightarrow \mathsf{tt}\}$$
$$\mathbf{ff} : \{\mathsf{nf}(\mathsf{bool})\ |\ \P \hookrightarrow \mathsf{ff}\}$$
$$\mathbf{up}_{\mathsf{bool}} : \{\mathsf{ne}(\mathsf{bool}) \to \mathsf{nf}(\mathsf{bool})\ |\ \P \hookrightarrow \lambda x.x\}$$
$$\mathbf{ind}_{\mathsf{bool}} : \{\textstyle\prod_{C:\mathsf{var}(\mathsf{bool})\to\mathsf{nftp}}\prod_{c_0:\mathsf{nf}(C(\mathsf{tt}))}\prod_{c_1:\mathsf{nf}(C(\mathsf{ff}))}\prod_{x:\mathsf{ne}(\mathsf{bool})}\mathsf{ne}(C(x))\ |\ \P \hookrightarrow \mathsf{ind}_{\mathsf{bool}}\}$$

The remainder of the normalization structure is then defined as follows:

$$\boxed{\mathsf{bool}_\alpha^{\circledast} : \{\mathsf{tp}_\alpha^{\circledast} \mid \P \hookrightarrow \mathsf{bool}_\alpha\}}$$

$$\mathsf{bool}_\alpha^{\circledast} \Leftarrow \mathbf{extend}\ \mathsf{bool}_\alpha^{\star}\ \langle\text{defined in }(\mathbf{5.3*8})\rangle$$

$$\downarrow_{\mathsf{tp}}^{\beta \geq \alpha}\mathsf{bool}_\alpha^{\circledast} = \mathbf{bool}_\beta$$

$$\hat{\uparrow}_{\mathsf{bool}_\alpha^{\circledast}} b = \mathbf{up}_{\mathsf{bool}}(b)$$

$$\downarrow_{\mathsf{bool}_\alpha^{\circledast}}\mathbf{tt} = \mathbf{tt}$$

$$\downarrow_{\mathsf{bool}_\alpha^{\circledast}}\mathbf{ff} = \mathbf{ff}$$

$$\downarrow_{\mathsf{bool}_\alpha^{\circledast}}\mathbf{up}_{\mathsf{bool}}(x) = \mathbf{up}_{\mathsf{bool}}(x)$$

$$\_ : \P \vdash \downarrow_{\mathsf{bool}_\alpha^{\circledast}} x = x$$

Above, the reification map goes by induction on elements of $\mathsf{bool}^{\star}$, or to be more precise, by induction on the proofs $\mathbf{bool}[b]$ for some syntactic boolean $b : \mathsf{bool}$. We recall from $(\mathbf{5.3*9})$ that $\mathbf{bool}[b]$ is defined as the smallest fiberwise $\P$-connected type closed under the rules $(\mathbf{5.3*8})$; to eliminate into a type that is not $\P$-connected we must therefore provide an additional clause to state what the map does under $\P$ that matches all the other clauses. We must also implement the induction principle:

$$\boxed{\{\mathsf{ind}_{\mathsf{bool}}^{\circledast} : \textstyle\prod_{C:\mathsf{bool}^{\circledast} \to \mathsf{tp}^{\circledast}} \prod_{c_0:C(\mathsf{tt}^{\circledast})} \prod_{c_1:C(\mathsf{ff}^{\circledast})} \prod_{x:\mathsf{bool}^{\circledast}} C(x) \mid \P \hookrightarrow \mathsf{ind}_{\mathsf{bool}}\}}$$

$$\mathsf{ind}_{\mathsf{bool}}^{\circledast}(C, c_0, c_1, \mathbf{tt}) = c_0$$

$$\mathsf{ind}_{\mathsf{bool}}^{\circledast}(C, c_0, c_1, \mathbf{ff}) = c_1$$

$$\mathsf{ind}_{\mathsf{bool}}^{\circledast}(C, c_0, c_1, \mathbf{up}_{\mathsf{bool}}(x)) = \hat{\uparrow}_{C(\hat{\uparrow}_{\mathsf{bool}^{\circledast}}x)}\mathbf{ind}_{\mathsf{bool}}\left(\downarrow_{\mathsf{tp}}^{\diamond}C(\hat{\uparrow}_{\mathsf{bool}^{\circledast}}x), \downarrow_{C(\mathbf{tt})}c_0, \downarrow_{C(\mathbf{ff})}c_1, x\right)$$

$$\_ : \P \vdash \mathsf{ind}_{\mathsf{bool}}^{\circledast}(C, c_0, c_1, x) = \mathsf{ind}_{\mathsf{bool}}(C, c_0, c_1, x)$$

§5.4. A NORMALIZATION ALGEBRA FOR MARTIN-LÖF'S TYPE THEORY

※ $(\mathbf{5.4*1})$ Suppose we have a *syntactic* algebra for Martin-Löf's type theory with universes, valued in $\mathcal{U}_\P : \mathbf{Set}_\mathsf{G}$; as in §4.5.2, our goal is to construct a normalization algebra $\mathbf{M}^{\circledast} : \{[\![\mathbb{ML}_{base}]\!]_\mathcal{W} \mid \P \hookrightarrow \mathbf{M}\}$ for sufficiently large $\mathcal{W}$. In §5.4.1 we give an inductive definition that explicitly substantiates the normal form constants assumed earlier in this chapter, and in §5.4.2 we construct the normalization algebra.

§5.4.1. Indexed inductive definition of normal forms

※ $(\mathbf{5.4.1*1})$ *An indexed inductive definition.* To substantiate the neutral/normal form constants enumerated in Fig. 5.1, we will manually define an indexed inductive definition

whose fibers are taken in the *closed subuniverse* $\mathcal{U}_{\backslash\P} \hookrightarrow \mathcal{U}$:

$$[a \in_{\mathsf{ne}} A] : \mathcal{U}_{\backslash\P} \qquad (A : \mathsf{tp}, a : \mathsf{tm}(A))$$

$$[A \ni_{\mathsf{nf}} a] : \mathcal{U}_{\backslash\P} \qquad (A : \mathsf{tp}, a : \mathsf{tm}(A))$$

$$[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A] : \mathcal{U}_{\backslash\P} \qquad (A : \mathsf{tp})$$

We then define $[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} A]$ to be $[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A]$ when $\alpha = \diamond$ and $[\mathsf{U}_n \ni_{\mathsf{nf}} A]$ when $\alpha = n$.

Viewed from the perspective of $\mathcal{U}$, we take this to mean that each inductive definition has an implicit clause that collapses each fiber to a point under $\P$; for example:

$$\frac{A : \mathsf{tp} \qquad z : \P}{\mathbf{pt}(A,z) : [\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A]} \qquad\qquad \frac{A : \mathsf{tp} \qquad \widetilde{A} : [\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A] \qquad z : \P}{\widetilde{A} =_{[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A]} \mathbf{pt}(A,z)}$$

The resulting indexed inductive types have an induction principle for motives valued in $\mathcal{U}_{\backslash\P}$ that *omits* the **pt** case; to eliminate into $\mathcal{U}$, one must provide a case for **pt** and show that under $\P$, all other cases agree with it.

(5.4.1∗2) *Basic judgmental structure.* We add inductive clauses to substantiate the basic judgmental structure of Martin-Löf's type theory with universes:

$$\frac{(n < \alpha)}{\mathsf{U}_n^\alpha : [\mathsf{tp}_\alpha \ni_{\mathsf{nf}} \langle\uparrow_{n+1}^\alpha\rangle\mathsf{U}_n]} \qquad\qquad \frac{A : \mathsf{tp}_n \qquad (n \leq \alpha) \qquad \widetilde{A} : [A \in_{\mathsf{ne}} \langle\uparrow_{n+1}^\diamond\rangle\mathsf{U}_n]}{\mathsf{up}_n^\alpha\{A\}(\widetilde{A}) : [\mathsf{tp}_\alpha \ni_{\mathsf{nf}} \langle\uparrow_n^\alpha\rangle A]}$$

$$\frac{\begin{array}{c} A : \mathsf{tp}_n \qquad a : \mathsf{tm}_n(A) \\ \widetilde{A} : [A \in_{\mathsf{ne}} \langle\uparrow_{n+1}^\diamond\rangle\mathsf{U}_n] \qquad \widetilde{a} : [a \in_{\mathsf{ne}} \langle\uparrow_n^\diamond\rangle A] \end{array}}{\widetilde{\mathsf{up}}_n\{A,a\}(\widetilde{A},\widetilde{a}) : [\langle\uparrow_n^\diamond\rangle A \ni_{\mathsf{nf}} a]} \qquad\qquad \frac{A : \mathsf{tp}_n \qquad \widetilde{a} : \mathsf{var}(A)}{\mathsf{var}\{A\}(\widetilde{a}) : [\widetilde{a} \in_{\mathsf{ne}} A]}$$

(5.4.1∗3) *Normal and neutral forms for connectives.* We add the inductive clauses exhibiting normal and neutral forms involving dependent products and sums:

$$\frac{\begin{array}{c} A : \mathsf{tp}_\alpha \qquad B : \mathsf{tm}_\alpha(A) \to \mathsf{tp}_\alpha \\ \widetilde{A} : [\mathsf{tp}_\alpha \ni_{\mathsf{nf}} A] \qquad \widetilde{B} : \prod_{x:\mathsf{var}(\langle\uparrow_\alpha^\diamond\rangle A)}[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} B(x)] \end{array}}{\mathbf{pi}_\alpha\{A,B\}(\widetilde{A},\widetilde{B}) : [\mathsf{tp}_\alpha \ni_{\mathsf{nf}} \Pi_\alpha(A,B)] \qquad \mathbf{sg}_\alpha\{A,B\}(\widetilde{A},\widetilde{B}) : [\mathsf{tp}_\alpha \ni_{\mathsf{nf}} \Sigma_\alpha(A,B)]}$$

$$\frac{\begin{array}{c} A : \mathsf{tp} \qquad B : \mathsf{tm}(A) \to \mathsf{tp} \qquad f : \mathsf{tm}(\Pi(A,B)) \\ \widetilde{f} : \prod_{x:\mathsf{var}(A)}[B(x) \ni_{\mathsf{nf}} f(x)] \end{array}}{\mathbf{lam}\{A,B,f\}(\widetilde{f}) : [\Pi(A,B) \ni_{\mathsf{nf}} f]}$$

$$\frac{\begin{array}{c} A : \mathsf{tp} \qquad B : \mathsf{tm}(A) \to \mathsf{tp} \qquad f : \mathsf{tm}(\Pi(A,B)) \qquad a : \mathsf{tm}(A) \\ \widetilde{f} : [f \in_{\mathsf{ne}} \Pi(A,B)] \qquad \widetilde{a} : [A \ni_{\mathsf{nf}} a] \end{array}}{\mathbf{app}\{A,B,f,a\}(\widetilde{f},\widetilde{a}) : [f(a) \in_{\mathsf{ne}} B(a)]}$$

$$\frac{A : \mathsf{tp} \qquad B : \mathsf{tm}(A) \to \mathsf{tp} \qquad p : \mathsf{tm}(\Sigma(A,B)) \qquad \widetilde{a} : [A \ni_{\mathsf{nf}} p.1] \qquad \widetilde{b} : [B(a) \ni_{\mathsf{nf}} p.2]}{\mathbf{pair}\{A,b,p\}(\widetilde{a},\widetilde{b}) : [\Sigma(A,B) \ni_{\mathsf{nf}} p]}$$

$$\frac{A : \mathsf{tp} \qquad B : \mathsf{tm}(A) \to \mathsf{tp} \qquad p : \mathsf{tm}(\Sigma(A,B)) \qquad \widetilde{p} : [p \in_{\mathsf{ne}} \Sigma(A,B)]}{\mathbf{fst}\{A,B,p\}(\widetilde{p}) : [p.1 \in_{\mathsf{ne}} A] \qquad \mathbf{snd}\{A,B,p\}(\widetilde{p}) : [p.2 \in_{\mathsf{ne}} B(a)]}$$

◻ **(5.4.1∗4)** *Normal and neutral forms for the booleans.* The normal and neutral forms for the boolean type are presented below:

$$\overline{\mathbf{bool}_\alpha : [\mathsf{tp}_\alpha \ni_{\mathsf{nf}} \mathsf{bool}_\alpha]} \qquad \overline{\mathbf{tt} : [\mathsf{bool} \ni_{\mathsf{nf}} \mathsf{tt}]} \qquad \overline{\mathbf{ff} : [\mathsf{bool} \ni_{\mathsf{nf}} \mathsf{tt}]}$$

$$\frac{a : \mathsf{tm}(\mathsf{bool}) \qquad \widetilde{a} : [a \in_{\mathsf{ne}} \mathsf{bool}]}{\mathbf{up}_{\mathsf{bool}}\{a\}(\widetilde{a}) : [\mathsf{bool} \ni_{\mathsf{nf}} a]}$$

$$\frac{C : \mathsf{tm}(\mathsf{bool}) \to \mathsf{tp} \quad c_0 : \mathsf{tm}(C(\mathsf{tt})) \quad c_1 : \mathsf{tm}(C(\mathsf{ff})) \quad a : \mathsf{tm}(\mathsf{bool}) \quad \widetilde{C} : \prod_{x:\mathsf{var}(\mathsf{bool})}[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} C(x)] \quad \widetilde{c}_0 : [C(\mathsf{tt}) \ni_{\mathsf{nf}} c_0] \quad \widetilde{c}_1 : [C(\mathsf{ff}) \ni_{\mathsf{nf}} c_1] \quad \widetilde{a} : [a \in_{\mathsf{ne}} \mathsf{bool}]}{\mathbf{ind}_{\mathsf{bool}}\{C,c_0,c_1,a\}(\widetilde{C},\widetilde{c}_0,\widetilde{c}_1,\widetilde{a}) : [\mathsf{ind}_{\mathsf{bool}}(C,c_0,c_1,a) \in_{\mathsf{ne}} C(a)]}$$

◻ **(5.4.1∗5)** *The collections of neutral and normal forms.* We then take the total spaces of the indexed inductive definitions **(5.4.1∗1)** to define the collections of neutral and normal forms of terms and types by realignment:

$$\mathsf{nftp}_\alpha : \{\mathcal{U} \mid \P \hookrightarrow \mathsf{tp}_\alpha\} \qquad\qquad \mathsf{nf} : \prod_{A:\mathsf{tp}}\{\mathcal{U} \mid \P \hookrightarrow \mathsf{tm}(A)\}$$
$$\mathsf{nftp}_\alpha \cong \sum_{A:\mathsf{tp}_\alpha}[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} A] \qquad \mathsf{nf}(A) \cong \sum_{a:\mathsf{tm}(A)}[A \ni_{\mathsf{nf}} a]$$

$$\mathsf{ne} : \prod_{A:\mathsf{tp}}\{\mathcal{U} \mid \P \hookrightarrow \mathsf{tm}(A)\}$$
$$\mathsf{ne}(A) \cong \sum_{a:\mathsf{tm}(A)}[a \in_{\mathsf{ne}} A]$$

The realignments above are possible because each fiber $[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} A], [A \ni_{\mathsf{nf}} a], [a \in_{\mathsf{ne}} A]$ was valued in the closed subuniverse $\mathcal{U}_{\backslash\P} \hookrightarrow \mathcal{U}$, and hence is $\P$-connected.

◻ **(5.4.1∗6)** The constants specified in Fig. 5.1 are obtained directly from the indexed inductive definitions under the realignment from **(5.4.1∗5)**. For instance, consider the constructor for the normal form of the dependent product type:

$$\mathbf{pi}_\alpha : \left\{ \left(\sum_{A:\mathsf{nftp}_\alpha}(\mathsf{var}(\langle\uparrow_\alpha^\diamond\rangle A) \to \mathsf{nftp}_\alpha) \right) \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \Pi_\alpha \right\}$$

Given a normal type $A : \mathsf{nftp}_\alpha$ and a normal family of types $B : \mathsf{var}(\langle\uparrow_\alpha^\diamond\rangle A) \to \mathsf{nftp}_\alpha$, we have by unfolding **(5.4.1∗5)** some $\widetilde{A} : [\alpha \ni_{\mathsf{nf}} A]$ and $\widetilde{B} : \prod_{x:\mathsf{var}(\langle\uparrow_\alpha^\diamond\rangle A)}[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} B(x)]$. We

then choose the element of nftp_α determined by the pair $(\Pi_\alpha(A,B), \mathbf{pi}_\alpha\{A,B\}(\tilde{A},\tilde{B}))$; the required boundary $\P \hookrightarrow \Pi_\alpha(A,B)$ follows immediately by definition of realignment. All the remaining constants work in an identical fashion.

◼ **(5.4.1∗7)** *Modal injectivity of normal form constructors.* It is not the case that the constructor for the normal form of the dependent product type

$$\mathbf{pi}_\diamond : \{(\textstyle\sum_{A:\mathsf{nftp}_\diamond}(\mathsf{var}(A) \to \mathsf{nftp})) \to \mathsf{nftp} \mid \P \hookrightarrow \Pi\}$$

is injective: indeed, this would imply that the syntactic Π constructor is injective, which we would not expect to hold internally to the syntactic category because of the existence of non-injective models of dependent products.

What we *can* show is a modal version of injectivity that captures the sense in which injectivity rules should be admissible but not derivable. Letting nffam be the type of pairs of a normal type $A : \mathsf{nftp}$ and a normal family $B : \mathsf{var}(A) \to \mathsf{nftp}$, we will verify the folowing modal statement:

$$\forall F,F' : \mathsf{nffam}. \ (\mathbf{pi}_\diamond(F) = \mathbf{pi}_\diamond(F')) \implies \bullet(F = F')$$

Proof. By induction for normal forms, we may define a predicate $\mathsf{isPi} : \mathsf{nftp} \to \Omega_{\backslash\P}$ valued in purely semantic propositions, satisfying the following universal property:

$$\forall X : \mathsf{nftp}.\mathsf{isPi}(X) \Leftrightarrow \bullet\exists A,B.X = \mathbf{pi}_\diamond(A,B) \qquad\qquad (5.4.1∗7∗1)$$

We define $\mathsf{isPi}(X)$ by induction as follows:

$$\mathsf{isPi}(\mathbf{U}_n^\diamond) = \bullet\bot \qquad \mathsf{isPi}(\mathbf{up}_n^\diamond) = \bullet\bot \qquad \mathsf{isPi}(\mathbf{pi}_\diamond(A,B)) = \top \qquad \mathsf{isPi}(\mathbf{sg}_\diamond(A,B)) = \bullet\bot$$

$$\mathsf{isPi}(\mathbf{bool}_\diamond) = \bullet\bot$$

The universal property depicted in Eq. (5.4.1∗7∗1) holds by induction. It is then possible to define another function to invert the $\mathbf{pi}_\diamond$ constructor within the extent of isPi:

$$\mathsf{unPi} : \{X : \mathsf{nftp} \mid \mathsf{isPi}(X)\} \to \bullet\mathsf{nffam}$$

$$\mathsf{unPi}(\mathbf{pi}_\diamond(A,B)) = \eta_{\backslash\P}(A,B)$$

No further cases are needed because $\mathsf{isPi}(X)$ computes to $\bullet\bot$ in all other cases. Now suppose that $\mathbf{pi}_\diamond(A,B) = \mathbf{pi}_\alpha(A',B')$; by applying unPi to both sides of this equation, we obtain $\eta_{\backslash\P}(A,B) = \eta_{\backslash\P}(A',B')$, which is equivalent to $\bullet((A,B) = (A',B'))$.   □

◔ **(5.4.1∗8)** The modal injectivity result of **(5.4.1∗7)** verifies the sense in which our definition of normal forms is *in fact* concrete enough to be useful — since one could have degenerately chosen $\mathsf{nftp}_\alpha := \mathsf{tp}_\alpha$ otherwise. Now that we have established the appropriateness of our notion of normal form, we will the instantiate synthetic constructions in the foregoing sections onto our explicit construction of the topos **G** in order to argue that every type and term can be represented by one of these normal forms in a sound and complete way.

§5.4.2. The normalization algebra

■ **(5.4.2∗1)** We define the components of the normalization algebra $\mathbf{M}^{*}$ one-by-one:

1) The hierarchy of type structures $\mathbf{M}^{*}.M_{\alpha}$ and universe codes $\mathbf{M}^{*}.U_{n}$ are given by **(5.3∗1)**, **(5.3∗3)** and **(5.3∗5)** from §5.3.
2) The dependent product structure at each universe level is given by **(5.3∗6)**.
3) The dependent sum structure at each universe level is given by **(5.3∗7)**.
4) The boolean structure is given by **(5.3∗8)** and **(5.3∗10)**.

§5.5. A TOPOS FOR NORMALIZATION

※ **(5.5∗1)** In this section we construct a topos $\mathbf{G}$ that substantiates the assumptions made by the foregoing synthetic constructions, to yield a concrete normalization result. As always, we will employ the yoga of *figure shapes* (§4.1); whereas in §4.5 we used the trivial figure shape **(4.5.1∗1)**, we will develop a more complex figure shape based on the notion of an *atomic term*; in ordinary dependent type theory, the atomic terms are just the variables, but they may have additional structure in more complex theories such as cubical type theory (Chapter 6).

$$
\begin{array}{llll}
\textit{(atomic context)} & \Gamma, \Delta & ::= & \mathbb{1} \mid \Gamma.A \\
\textit{(atomic term)} & \mathfrak{a}, \mathfrak{b} & ::= & \mathbf{q}_A \mid \mathbf{p}_A(\mathfrak{b}) \\
\textit{(atomic substitution)} & \gamma, \delta & ::= & \cdot \mid \gamma.\mathfrak{a}
\end{array}
$$

Figure 5.3: The grammar of atomic contexts, terms, and substitutions.

§5.5.1. Atomic terms: variables and structural renamings

■ **(5.5.1∗1)** We begin by defining a notion of *atomic context*; intuitively, such a context is the same as an ordinary context in dependent type theory[1], but the morphisms between atomic contexts will correspond only to renamings of variables and not to arbitrary terms. We define the atomic contexts together with the judgments they correspond to inductively

---

[1] Recall, however, that our axiomatization of dependent type theory Fig. 1.2 does not distinguish contexts *a priori* from other judgments.

by assertions $\Gamma\ ctx\ \rightsquigarrow X$ with $X : \mathcal{T}$, meaning "Γ is an atomic context-form of X".

$$\boxed{\Gamma\ ctx\ \rightsquigarrow X}\ presupposing\ X : \mathcal{T}$$

$$\frac{}{\mathbb{1}\ ctx\ \rightsquigarrow \mathbf{1}_{\mathcal{T}}} \qquad \frac{\Gamma\ ctx\ \rightsquigarrow X \qquad A : X \longrightarrow \mathsf{tp}}{\Gamma.A\ ctx\ \rightsquigarrow \sum_{x:X}\mathsf{tm}(A(x))}$$

(5.5.1∗2) *Atomic terms.* For Martin-Löf type theory, an "atomic term" is just a variable; we give an inductive De Bruijn encoding of variables below, simultaneously with their decodings into actual semantic terms.

$$\boxed{\Gamma\Vdash \mathfrak{a} : A \rightsquigarrow a}\ presupposing\ \begin{cases} \Gamma\ ctx\ \rightsquigarrow X \\ A : X \to \mathsf{tp} \\ a : \prod_{x:X}\mathsf{tm}(A(x)) \end{cases}$$

TOP VARIABLE
$$\frac{\Gamma\ ctx\ \rightsquigarrow X \qquad A : X \to \mathsf{tp}}{\Gamma.A \Vdash \mathbf{q}_A : A \circ \pi_1 \rightsquigarrow \pi_2}$$

POP VARIABLE
$$\frac{\Gamma\ ctx\ \rightsquigarrow X \qquad A,B : X \to \mathsf{tp} \qquad a : \prod_{x:X}\mathsf{tm}(A)(x) \qquad \Gamma\Vdash \mathfrak{a} : A \rightsquigarrow a}{\Gamma.B \Vdash \mathbf{p}_A(\mathfrak{a}) : A \circ \pi_1 \rightsquigarrow a \circ \pi_1}$$

(5.5.1∗3) *Atomic substitutions.* The notion of an atomic term **(5.5.1∗2)** is extended point-wise to give a notion of simultaneous atomic substitutions:

$$\boxed{\Delta\Vdash \gamma : \Gamma \rightsquigarrow y}\ presupposing\ \begin{cases} \Delta\ ctx\ \rightsquigarrow X \\ \Gamma\ ctx\ \rightsquigarrow Y \\ y : X \to Y \end{cases}$$

NIL
$$\frac{\Delta\ ctx\ \rightsquigarrow X}{\Delta\Vdash \cdot : \mathbb{1} \rightsquigarrow !_X}$$

SNOC
$$\frac{\begin{array}{cc} \Delta\ ctx\ \rightsquigarrow X & \Gamma\ ctx\ \rightsquigarrow Y \\ A : Y \to \mathsf{tp} & y : X \to Y \qquad a : \prod_{x:X}\mathsf{tm}(A(y(x))) \\ \Delta\Vdash \gamma : \Gamma \rightsquigarrow y & \Delta\Vdash \mathfrak{a} : A \circ y \rightsquigarrow a \end{array}}{\Delta\Vdash \gamma.\mathfrak{a} : \Gamma.A \rightsquigarrow (y,a)}$$

(5.5.1∗4) *Identity substitution.* For any atomic context $\Gamma\ ctx\ \rightsquigarrow X$, there is an atomic substitution $\Gamma\Vdash \mathbf{id}_\Gamma : \Gamma \rightsquigarrow \mathbf{id}_X$. We may form $\mathbf{id}_\Gamma$ by recursion on the structure of Γ.

$$\mathbf{id}_{\mathbb{1}} = \cdot$$
$$\mathbf{id}_{\Gamma.A} = \mathbf{id}_\Gamma.\mathbf{q}_A$$

▪ **(5.5.1∗5)** *Atomic substitution action.* Let $\Delta\ ctx \rightsquigarrow X$ and $\Gamma\ ctx \rightsquigarrow Y$ be atomic contexts, and let $\Delta \Vdash \gamma : \Gamma \rightsquigarrow y$ by an atomic substitution. For any atomic term atomic term $\Gamma \Vdash \mathfrak{a} : A \rightsquigarrow a$, we may define a substituted atomic term $\Gamma \Vdash \gamma^*\mathfrak{a} : A \rightsquigarrow a$ by recursion on $\gamma,\mathfrak{a}$ using inversion.

$$(\gamma.\mathfrak{a})^*\mathbf{q}_A = \mathfrak{a}$$
$$(\gamma.\mathfrak{b})^*\mathbf{p}_B(\mathfrak{a}) = \gamma^*\mathfrak{a}$$

▪ **(5.5.1∗6)** *Composition of atomic substitutions.* Let $\Delta\ ctx \rightsquigarrow X$, $\Gamma\ ctx \rightsquigarrow Y$, and $\Xi\ ctx \rightsquigarrow Z$ be atomic contexts, and let $\Delta \Vdash \gamma : \Gamma \rightsquigarrow y$ and $\Gamma \Vdash \xi : \Xi \rightsquigarrow z$ be atomic substitutions. We may define a composite substitution $\Delta \Vdash \xi \circ \gamma : \Xi \rightsquigarrow z \circ y$ using the substitution action on atomic terms **(5.5.1∗5)** by recursion on ξ.

$$\cdot \circ \gamma = \cdot$$
$$(\xi.\mathfrak{a}) \circ \gamma = (\xi \circ \gamma).\gamma^*\mathfrak{a}$$

We observe that the composition of substitutions is associative and unital, *i.e.* we have the following laws:

$$\mathbf{id}_\Gamma \circ \gamma = \gamma$$
$$\gamma \circ \mathbf{id}_\Gamma = \gamma$$
$$(\gamma \circ \delta) \circ \xi = \gamma \circ (\delta \circ \xi)$$

▪ **(5.5.1∗7)** *The atomic figure shape.* We may therefore form a category $\mathcal{A}$ whose objects are atomic contexts $\Gamma\ ctx \rightsquigarrow X$ and whose morphisms are atomic substitutions $\Delta \Vdash \gamma : \Gamma \rightsquigarrow y$, together with a functor $\boldsymbol{\alpha} : \mathcal{A} \longrightarrow \mathcal{T}$ taking each atomic context to its underlying judgment:

$$\boxed{\boldsymbol{\alpha} : \mathcal{A} \longrightarrow \mathcal{T}}$$
$$\boldsymbol{\alpha}(\Gamma\ ctx \rightsquigarrow X) = X$$
$$\boldsymbol{\alpha}(\Delta \Vdash \gamma : \Gamma \rightsquigarrow y) = y$$

We will refer to the functor $\boldsymbol{\alpha} : \mathcal{A} \longrightarrow \mathcal{T}$ as the *atomic figure shape* in the sense of §4.1; the purpose of $\boldsymbol{\alpha}$ is to determine a suitable essential geometric morphism **(4.3∗5)** to glue along.

§5.5.2. The syntactic topos and its universal property

▪ **(5.5.2∗1)** *The syntactic topos and the Yoneda model.* We can turn the category of judgments $\mathcal{T}$ into a topos $\mathbf{T} := \widehat{\mathcal{T}}$, defined by the identification $\mathbf{Set}_\mathbf{T} = \mathrm{Pr}(\mathcal{T})$. The Yoneda embedding $y_\mathcal{T} : \mathcal{T} \hookrightarrow \mathrm{Pr}(\mathcal{T})$ is locally Cartesian closed, hence it is a *model* of Martin-Löf type theory over $\mathbf{T}$.

■ **(5.5.2∗2)** *The generic model.* The Yoneda model **(5.5.2∗1)** is *universal* in the sense that any other topos model M : $\mathcal{T} \longrightarrow \mathbf{Set_Y}$ arises from $y_\mathcal{T}$ by restriction along an essentially unique morphism of topoi $\chi_M : \mathbf{Y} \longrightarrow \mathbf{T}$ in the sense that $\chi_M^* \circ y_\mathcal{T} \cong M$:

$$
\begin{array}{ccc}
M & \dashrightarrow & y_\mathcal{T} \\[-2pt]
\downarrow & \lrcorner & \downarrow \\[-2pt]
\mathbf{Y} & \xrightarrow[\chi_M]{} & \mathbf{T}
\end{array}
\qquad (5.5.2\ast2\ast1)
$$

The characteristic map $\chi_M : \mathbf{Y} \longrightarrow \mathbf{T}$ is obtained in the following way from the special case of Diaconescu's theorem [Dia75] that characterizes geometric morphisms into presheaf topoi. We consider the cocontinuous *Yoneda extension* of the model M : $\mathcal{T} \longrightarrow \mathbf{Set_Y}$ induced by the universal property of $\Pr(\mathcal{T})$ as the free cocompletion of $\mathcal{T}$:

$$
\begin{array}{ccc}
\mathcal{T} & \xrightarrow{\ M\ } & \mathbf{Set_Y} \\[-2pt]
{\scriptstyle y_\mathcal{T}}\downarrow & \nearrow {\scriptstyle \widetilde{M}} & \\[-2pt]
\Pr(\mathcal{T}) & &
\end{array}
\qquad (5.5.2\ast2\ast2)
$$

By definition $\widetilde{M}$ is cocontinuous; because M is left exact and $\mathcal{T}$ is finitely complete, we conclude from Diaconescu's theorem that $\widetilde{M}$ is also left exact. Hence $\widetilde{M}$ is the inverse image part of a morphism of topoi that we shall denote $\chi_M : \mathbf{Y} \longrightarrow \mathbf{T}$; we immediately have $\chi_M^* \circ y_\mathcal{T} \cong M$ considering Diagram 5.5.2∗2∗2, because we have defined $\chi_M^* = \widetilde{M}$. That χ_M is essentially unique with this property is deduced again from Diaconescu's theorem.

◔ **(5.5.2∗3)** *Henkin models.* We have seen that each model $\mathcal{T} \longrightarrow \mathbf{Set_Y}$ arises in a canonical way from a morphism of topoi $\mathbf{Y} \longrightarrow \mathbf{T}$; conversely, it is reasonable to ask which morphisms $\mathbf{Y} \longrightarrow \mathbf{T}$ correspond to models of $\mathcal{T}$ in $\mathbf{Set_Y}$. If $f : \mathbf{Y} \longrightarrow \mathbf{T}$ is an arbitrary morphism of topoi, the restriction $f^* \circ y_\mathcal{T} : \mathcal{T} \longrightarrow \mathbf{Set_Y}$ is clearly left exact, but there is no reason for it to preserve dependent products.

In logic and model theory, an "almost model" that may interpret relations or function types in a non-standard way is called a *Henkin model*[2] — in particular, every actual model is a Henkin model, but important metatheorems (such as completeness) tend to involve "proper" Henkin models with non-standard interpretations of relations; the study of type theory is no exception, as can be seen already from the importance of "weak morphisms of cwfs" in the existing literature [KHS19; New18].

■ **(5.5.2∗4)** It is perhaps appropriate to adapt the same terminology to our situation, defining a Henkin model of the type theory $\mathcal{T}$ to be a left exact functor $\mathcal{T} \longrightarrow \mathbf{Set_Y}$ without

---

[2] Originally considered by Henkin [Hen50]; relevantly, these were studied in a topos-theoretic setting by Awodey and Butz [AB00].

any additional conditions on the interpretation of dependent products (hypothetical judgments). The interpretation of a dependent product in such a Henkin model necessarily behaves like a dependent product with respect to things in the image of the interpretation, but it may not in fact be the dependent product in $\mathbf{Set_Y}$.

■ **(5.5.2∗5)** Therefore the pair $(\mathbf{T}, y_{\mathcal{T}})$ satisfies an additional universal property: $\mathbf{T}$ is also the classifying topos of Henkin models of $\mathcal{T}$, and $y_{\mathcal{T}}$ is the universal Henkin model of $\mathcal{T}$.

■ **(5.5.2∗6)** When it is convenient, we will often refer to a morphism of topoi $f : \mathbf{Y} \longrightarrow \mathbf{T}$ as a (Henkin) $\mathcal{T}$-model, blurring the lines between a functorial model and its characteristic map. Under this identification, we may then attach further adjectives to the model that describe classes of morphisms of topoi. For example:

1) An essential (Henkin) $\mathcal{T}$-model is one whose characteristic map $f : \mathbf{Y} \longrightarrow \mathbf{T}$ is essential, *i.e.* the inverse image $f^* : \mathrm{Pr}(\mathcal{T}) \longrightarrow \mathbf{Set_Y}$ has an additional left adjoint $f_! \dashv f^*$.

2) A locally connected (Henkin) $\mathcal{T}$-model is one whose characteristic map $f : \mathbf{Y} \longrightarrow \mathbf{T}$ is locally connected, *i.e.* the inverse image $f^* : \mathrm{Pr}(\mathcal{T}) \longrightarrow \mathbf{Set_Y}$ preserves dependent products.[3]

■ **(5.5.2∗7)** *Essential Henkin models from figure shapes.* Any figure shape $\rho : \mathcal{C} \longrightarrow \mathcal{T}$ gives rise to a Henkin $\mathcal{T}$-model over the presheaf topos $\widehat{\mathcal{C}}$. We define the (abusively named) morphism of topoi $\rho : \widehat{\mathcal{C}} \longrightarrow \mathbf{T}$ in terms of its inverse image using precomposition:

$$\rho^* : \mathrm{Pr}(\mathcal{T}) \longrightarrow \mathrm{Pr}(\mathcal{C})$$
$$(\rho^* X)(C) = X(\rho(C))$$

The precomposition functor ρ^* has both left and right adjoints $\rho_! \dashv \rho^* \dashv \rho_*$ by Kan extension, so it is the inverse image part of an essential geometric morphism. We therefore refer to $\rho : \widehat{\mathcal{C}} \longrightarrow \mathbf{T}$ as the *essential Henkin model* corresponding to the figure shape $\rho : \mathcal{C} \longrightarrow \mathcal{T}$.

■ **(5.5.2∗8)** We will write $\mathbf{A} := \widehat{\mathcal{A}}$ for the presheaf topos on atomic contexts and substitutions, which we might refer to as the "topos of atomic terms". Then, we will write $\alpha : \mathbf{A} \longrightarrow \mathbf{T}$ for the essential Henkin model corresponding to the figure shape $\alpha : \mathcal{A} \longrightarrow \mathcal{T}$. This notation accords with our intuition that $\mathbf{A}, \mathbf{T}$ are geometrical entities that correspond to "blown up" versions of $\mathcal{A}, \mathcal{T}$.

§5.5.3. The computability topos

■ **(5.5.3∗1)** In this section, we define a topos $\mathbf{G}$ that satisfies all the assumptions made earlier in the chapter. In particular, we take $\mathbf{G}$ to be the Artin gluing of the atomic figure

---

[3] Obviously any locally connected Henkin model is an actual model; the converse does not hold.

shape $\alpha : \mathbf{A} \longrightarrow \mathbf{T}$:
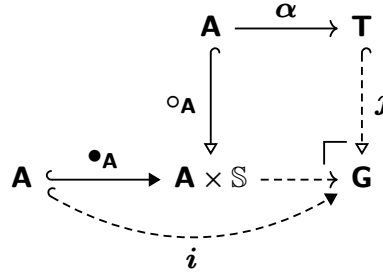
The syntactic open $\P : \mathcal{O}_\mathbf{G}$ is defined to be the subterminal satisfying $\mathbf{T} \simeq \mathbf{G}_\P$ in the sense of **(2.1∗6)**; this can be computed explicitly by setting $\P := j_! 1_{\mathbf{Set}_\mathbf{T}} = (1_{\mathbf{Set}_\mathbf{T}}, 0_{\mathbf{Set}_\mathbf{A}} \longrightarrow \alpha^* 1_{\mathbf{Set}_\mathbf{T}})$.

(5.5.3∗2) We recall from **(4.3∗5)** that the closed immersion $i : \mathbf{A} \hookrightarrow \mathbf{G}$ is an essential map, because the figure shape $\alpha : \mathbf{A} \longrightarrow \mathbf{T}$ is essential. Hence we have an additional left adjoint $i_! \dashv i^* \dashv i_*$, computed as follows:

$$i_! : \mathbf{Set}_\mathbf{A} \longrightarrow \mathbf{Set}_\mathbf{G}$$
$$i_! : \mathrm{E} \longmapsto (\alpha_! \mathrm{E}, \eta_\mathrm{E} : \mathrm{E} \longrightarrow \alpha^* \alpha_! \mathrm{E})$$

(5.5.3∗3) *The collection of variables.* We first define a presheaf of typed variables in $\mathbf{Set}_\mathbf{A} = \mathrm{Pr}(\mathcal{A})$; to be precise, we start by defining a presheaf V equipped with a representable natural transformation $\mathsf{V} \longrightarrow \alpha^* \mathsf{tm} : \mathrm{Pr}(\mathcal{A})$, where by tm we mean $\sum_{\mathrm{A:tp}} \mathsf{tm}(\mathrm{A})$. By adjointness, each representable point of $\alpha^* \mathsf{tm} : \mathrm{Pr}(\mathcal{A})$ is determined by an "atomic point" of $\mathsf{tm} : \mathrm{Pr}(\mathcal{T})$, *i.e.* a morphism $\alpha(\Gamma) \longrightarrow \mathsf{tm}$ for some atomic context Γ. Hence we define V by setting its fiber over each $(\mathrm{A}, a) : \alpha(\Gamma) \longrightarrow \mathsf{tm}$ to be the collection of variables $\mathfrak{a}$ such that $\Gamma \Vdash \mathfrak{a} : \mathrm{A} \rightsquigarrow a$. Hence we have a glued object $\widetilde{\mathsf{var}} : \mathbf{Set}_\mathbf{G}$ determined by the pair $(\mathsf{tm}, \mathsf{V} \longrightarrow \alpha^* \mathsf{tm})$; furthermore, we have a morphism $p : \widetilde{\mathsf{var}} \longrightarrow j_* \mathsf{tp}$ under adjointness by the projection $(j^* \widetilde{\mathsf{var}} \equiv \mathsf{tm}) \longrightarrow \mathsf{tp}$. Therefore we may define the collection of variables by realignment and pullback like so:

$$\mathsf{var} : \textstyle\prod_{\mathrm{A:tp}} \{ \mathcal{U} \mid \P \hookrightarrow \mathsf{tm}(\mathrm{A}) \}$$
$$\mathsf{var}(\mathrm{A}) \cong \{ \mathfrak{a} : \widetilde{\mathsf{var}} \mid p(\mathfrak{a}) = \mathrm{A} \}$$

§5.6. THE NORMALIZATION RESULT

§5.6.1. Computability structure of atomic substitutions and "atomic points"

▪ **(5.6.1∗1)** Given an atomic context Γ, we may define a computability structure $(\!(\Gamma)\!) : \mathbf{Set_G}$ of "atomic substitutions into Γ", namely $(\!(\Gamma)\!) = i_! y_{\mathcal{A}}(\Gamma)$. In the specifics of Martin-Löf type theory, $(\!(\Gamma)\!)$ is a computability structure of vectors of variables.

▪ **(5.6.1∗2)** Given a computability structure $X : \mathbf{Set_G}$, it would be appropriate to refer to any morphism of the form $(\!(\Gamma)\!) \longrightarrow X$ as an "atomic point".

▪ **(5.6.1∗3)** The atomic points of $X : \mathbf{Set_G}$ are in natural bijection with the "representable points" of its semantic component i^*X, *i.e.* we have a natural isomorphism of hom sets $\mathrm{Hom}_{\mathbf{Set_G}}((\!(\Gamma)\!), X) \cong \mathrm{Hom}_{\mathbf{Set_A}}(y(\Gamma), i^*X)$.

Proof. By adjointness, using the fact that $(\!(\Gamma)\!) = i_! y_{\mathcal{A}}(\Gamma)$ and $i_! \dashv i^*$.   □

In other words, the "functor of atomic points" of X is naturally isomorphic to the functor of points of i^*X.

▾ **(5.6.1∗4)** We may unfold the definition **(5.6.1∗1)** to gain a better understanding. First of all the syntactic part of $(\!(\Gamma)\!)$ is always $y_{\mathcal{T}}(\alpha(\Gamma))$:

$$\mathrm{Hom}_{\mathbf{Set_T}}(j^*(\!(\Gamma)\!), E)$$

$$\begin{array}{ll}
= \mathrm{Hom}_{\mathbf{Set_T}}(j^* i_! y_{\mathcal{A}}(\Gamma), E) & \text{by def.} \\
\cong \mathrm{Hom}_{\mathbf{Set_G}}(i_! y_{\mathcal{A}}(\Gamma), j_* E) & \text{by } j^* \dashv j_* \\
\cong \mathrm{Hom}_{\mathbf{Set_A}}(y_{\mathcal{A}}(\Gamma), i^* j_* E) & \text{by } i_! \dashv i^* \\
\cong \mathrm{Hom}_{\mathbf{Set_A}}(y_{\mathcal{A}}(\Gamma), \alpha^* E) & \text{by computation} \\
\cong \mathrm{Hom}_{\mathbf{Set_T}}(\alpha_! y_{\mathcal{A}}(\Gamma), E) & \text{by } \alpha_! \dashv \alpha^* \\
\cong \mathrm{Hom}_{\mathbf{Set_T}}(y_{\mathcal{T}}(\alpha(\Gamma)), E) & \alpha_! \text{ is Yoneda extension of } \alpha
\end{array}$$

Now consider an element $y_{\mathcal{A}}(\Delta) \longrightarrow \alpha^* j^*(\!(\Gamma)\!)$ of the base, *i.e.* a map $g : \alpha(\Delta) \longrightarrow \alpha(\Gamma)$; a witness of computability for g is nothing more than an atomic substitution $\Delta \Vdash \gamma : \Gamma \rightsquigarrow g$, which can be seen explicitly considering the computation **(5.5.3∗2)**. This is the sense in which $(\!(\Gamma)\!)$ is the computability structure of atomic substitutions for an atomic context Γ;

§5.6.2. Canonical computability structures and "canonical points"

▪ **(5.6.2∗1)** *The interpretation functor.* The normalization algebra $\mathbf{M}^\circledast$ defined in **(5.4.2∗1)** lying over generic $\mathcal{T}$-model under the open immersion $j : \mathbf{T} \hookrightarrow \mathbf{G}$ corresponds to a locally

Cartesian functor $\mathcal{T} \longrightarrow \mathbf{Set_G}$ in the following configuration:

$$
\begin{array}{ccc}
\mathcal{T} & \xdashrightarrow{\ \mathbf{M^*}\ } & \mathbf{Set_G} \\
& {}_{\mathcal{Y_T}}\searrow & \downarrow {}^{\boldsymbol{j^*}} \\
& & \mathrm{Pr}(\mathcal{T})
\end{array}
$$

▪ **(5.6.2∗2)** The interpretation functor $\mathbf{M^*} : \mathcal{T} \longrightarrow \mathbf{Set_G}$ induces a "canonical" computability structure $[\![\Gamma]\!]$ for each atomic context Γ, namely $[\![\Gamma]\!] = \mathbf{M^*}(\boldsymbol{\alpha}(\Gamma))$.

▪ **(5.6.2∗3)** We have a pointwise-vertical natural transformation $\hat{\jmath}_{\mathsf{atm}} : (\!(-)\!) \longrightarrow [\![-]\!]$ defined by extending the *reflection* of atomic terms to computable terms pointwise:

$$
\hat{\jmath}^{\Gamma}_{\mathsf{atm}} : (\!(\Gamma)\!) \longrightarrow [\![\Gamma]\!]
$$
$$
\hat{\jmath}^{\mathbb{1}}_{\mathsf{atm}}(\cdot) = \cdot
$$
$$
\hat{\jmath}^{\Gamma.A}_{\mathsf{atm}}(\gamma.\mathfrak{a}) = (\hat{\jmath}^{\Gamma}_{\mathsf{atm}}(\gamma),\, \hat{\jmath}_{\mathbf{M^*}(A)}\mathbf{var}(\mathfrak{a}))
$$

▪ **(5.6.2∗4)** Analogous to **(5.6.1∗2)**, it is appropriate to refer to a morphism of the form $[\![\Gamma]\!] \longrightarrow X$ as a "canonical point" of the computability structure $X : \mathbf{Set_G}$. The *functor of canonical points* $\mathrm{Hom}_{\mathbf{Set_G}}([\![-]\!], X) : \mathbf{Set_A}$ can be turned into a further computability structure over $\boldsymbol{j^*}X$ in a canonical way.

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[\![-]\!] \longrightarrow X}{(\!(-)\!) \longrightarrow X}\ \text{(5.6.2∗3)}}{i_! \mathsf{y}_{\mathcal{A}}(-) \longrightarrow X}}{i_! \mathsf{y}_{\mathcal{A}}(-) \longrightarrow \boldsymbol{j}_* \boldsymbol{j^*} X}}{\mathsf{y}_{\mathcal{A}}(-) \longrightarrow \boldsymbol{i^*} \boldsymbol{j}_* \boldsymbol{j^*} X}}{\mathsf{y}_{\mathcal{A}}(-) \longrightarrow \boldsymbol{\alpha^*} \boldsymbol{j^*} X}}{\boldsymbol{\alpha^*} \boldsymbol{j^*} X}
$$

We will write $X_{can} : \mathbf{Set_G}$ for the induced *computability structure of canonical points*. By **(5.6.1∗3)**, the analogous computability structure of atomic points of X is just X itself.

▪ **(5.6.2∗5)** We may define a vertical map $\hat{\jmath}^*_{\mathsf{atm}} : X_{can} \longrightarrow X$ for any computability structure X, *i.e.* one that restricts to the identity under ¶. To construct such a vertical map is the same as to define a morphism configured in $\mathbf{Set_A}$ like so, recalling from **(5.6.1∗3)** that

$i^*X \cong \mathrm{Hom}_{\mathbf{Set_G}}((\!|-|\!),X)$:

$$\mathrm{Hom}_{\mathbf{Set_G}}([\![-]\!],X) \dashrightarrow \mathrm{Hom}_{\mathbf{Set_G}}((\!|-|\!),X)$$
$$\alpha^*j^*X$$

We define the dotted above by restriction along the pointwise vertical natural transformation $\hat{\jmath}_{\mathbf{atm}} : (\!|-|\!) \longrightarrow [\![-]\!]$ defined in **(5.6.2∗3)**.

(5.6.2∗6) *The evaluation morphism.* Writing $\mathbf{M} : \mathcal{T} \longrightarrow \mathbf{Set_G}$ for the generic model $j_* \circ y_{\mathcal{T}}$, we may define a vertical morphism $\mathsf{eval}_\mathrm{T} : \mathbf{M}(\mathrm{T}) \longrightarrow \mathbf{M}^*(\mathrm{T})_{can}$ for any sort $\mathrm{T} : \mathcal{T}$ that abstractly "evaluates" syntax into canonical points of T's canonical computability structure. This is the same as to exhibit a section $\alpha^*y_{\mathcal{T}}(\mathrm{T}) \longrightarrow \mathrm{Hom}_{\mathbf{Set_G}}([\![-]\!],\mathbf{M}^*(\mathrm{T}))$ configured in $\mathbf{Set_A}$ like so:

$$\alpha^*y_{\mathcal{T}}(\mathrm{T}) \dashrightarrow \mathrm{Hom}_{\mathbf{Set_G}}([\![-]\!],\mathbf{M}^*(\mathrm{T}))$$
$$\Big\downarrow \mathbf{M}^*(\mathrm{T})_{can}$$
$$\alpha^*y_{\mathcal{T}}(\mathrm{T})$$

Recalling that $\alpha^*y_{\mathcal{T}}(\mathrm{T}) \cong \mathrm{Hom}_{\mathcal{T}}(\alpha(-),\mathrm{T})$ and $[\![-]\!] = \mathbf{M}^*(\alpha(-))$, we see that the upstairs map can be given by the functorial action $\mathrm{Hom}(\alpha(-),\mathrm{T}) \longrightarrow \mathrm{Hom}(\mathbf{M}^*(\alpha(-)),\mathbf{M}^*(\mathrm{T}))$ ∎

(5.6.2∗7) For a sort T, we will write $[\![-]\!]_\mathrm{T} : \mathbf{M}(\mathrm{T}) \longrightarrow \mathbf{M}^*(\mathrm{T})$ for the following vertical composite:

$$\mathbf{M}(\mathrm{T}) \xrightarrow{\ \mathsf{eval}_\mathrm{T}\ } \mathbf{M}^*(\mathrm{T})_{can} \xrightarrow{\ \hat{\jmath}^*_{\mathbf{atm}}\ } \mathbf{M}^*(\mathrm{T})$$
$$[\![-]\!]_\mathrm{T}$$

The function of $[\![-]\!]_\mathrm{T}$ is to give the normalization-algebra interpretation of any syntactic object by first evaluating and then by reflecting the entire context.

§5.6.3. The normalization function and its correctness

(5.6.3∗1) *The normalization function.* We may define a *vertical* normalization function $\mathsf{nbe}_{\mathsf{tp}} : \mathsf{tp} \longrightarrow \mathsf{nftp}$ in $\mathbf{Set_G}$ that takes a syntactic type to its normal form, using the vertical

maps from **(5.6.2∗5)** and **(5.6.2∗6)**:

$$\mathsf{tp} \xrightarrow{\ [\![-]\!]_{\mathsf{tp}}\ } \mathsf{tp}^* \xrightarrow{\ \downarrow^\diamond_{\mathsf{tp}}\ } \mathsf{nftp}$$
$$\mathsf{nbe}_{\mathsf{tp}}$$

Normalization of terms works in the same way:

$$\mathsf{tm} \xrightarrow{\ [\![-]\!]_{\mathsf{tm}}\ } \mathsf{tm}^* \xrightarrow{\ \lambda(A,a).(A,\downarrow_A a)\ } \textstyle\sum_{A:\mathsf{tp}}\mathsf{nf}(A)$$
$$\mathsf{nbe}_{\mathsf{tm}}$$

(5.6.3∗2) *Soundness and completeness.* Normalization is sound and complete in the following internal sense, fixing two syntactic types A,B : tp:
1) *Soundness* — if $\mathsf{nbe}_{\mathsf{tp}}(A) = \mathsf{nbe}_{\mathsf{tp}}(B)$, then $A = B$.
2) *Completeness* — if $A = B$ then $\mathsf{nbe}_{\mathsf{tp}}(A) = \mathsf{nbe}_{\mathsf{tp}}(B)$.

Proof. Soundness follows from the fact that the normalization function is vertical, hence a section of the unit to the open modality, and hence a monomorphism. Completeness is immediate, by virtue of the fact that normalization is a *function* and we are working with semantic terms rather than raw terms. □

(5.6.3∗3) The fact that soundness and completeness are *immediate* consequences of the construction is a significant advantage of abstract / algebraic techniques like those used here over the conventional operational versions of normalization by evaluation. Indeed, a traditional proof of completeness of normalization can take up to 60 pages of very tricky inductive cases, and likewise for soundness.

§5.6.4. Injectivity of type constructors

(5.6.4∗1) We may now extend the modal injectivity result of **(5.4.1∗7)** to the syntactic type constructors. In particular, we argue that the following proposition holds in $\mathbf{Set}_{\mathbf{G}}$:

$$\forall F_0, F_1 : \textstyle\sum_{A:\mathsf{tp}}(\mathsf{tm}(A) \to \mathsf{tp}).\ \Pi(F_0) = \Pi(F_1) \implies \bullet(F_0 = F_1)$$

Proof. Fixing $F_i = (A_i, B_i)$ such that $\Pi(F_0) = \Pi(F_1)$, we must show that $\bullet(F_0 = F_1)$. By soundness **(5.6.3∗2)** it suffices to show $\bullet(\mathsf{nbe}_{\mathsf{fam}}(F_0) = \mathsf{nbe}_{\mathsf{fam}}(F_1))$, where $\mathsf{nbe}_{\mathsf{fam}}(F_i)$ denotes the pair $(\mathsf{nbe}_{\mathsf{tp}}(A_i), \lambda x.\mathsf{nbe}_{\mathsf{tp}}(B_i(x)))$. By modal injectivity of normal form constructors **(5.4.1∗7)**, it suffices to show that $\mathbf{pi}_\diamond(\mathsf{nbe}_{\mathsf{fam}}(F_0)) = \mathbf{pi}_\diamond(\mathsf{nbe}_{\mathsf{fam}}(F_1))$. But $\mathbf{pi}_\diamond(\mathsf{nbe}_{\mathsf{fam}}(F_i)) = \mathsf{nbe}_{\mathsf{tp}}(\Pi(F_i))$, and we have already assumed $\Pi(F_0) = \Pi(F_1)$. □

§5.6.5. Surjectivity of the normalization function

(5.6.5∗1) It is possible *a priori* for a given type or term to have arbitrarily many distinct normal forms, which would have negative implications for the *computational effectivity* of the normalization function defined in §5.6.3. Therefore we want to prove that the normal form representation is *tight*, in the sense that the normalization function is surjective. To do so, it is necessary to find a strong enough induction motive — which is handily supplied by Fiore [Fio02] and Kaposi [Kap17], adapted below in **(5.6.5∗3)**: we must prove that all normal and neutral forms are *stationary* with respect to the normalization function.

(5.6.5∗2) *Variables are stationary.* For any variable $x : \mathsf{var}(A)$, we have $[\![x]\!]_{\mathsf{tm}(A)} = \mathord{\uparrow}_{[\![A]\!]}\mathbf{var}(x)$. In other words, variables in the normalization algebra are always interpreted by the reflection.

Proof. First we note that the identification above holds already when restricted under the open modality, hence it suffices to argue "upstairs" in the language of the closed subtopos. Therefore, we may fix an atomic context $\Gamma : \mathcal{A}$ and an atomic term $\Gamma \Vdash x : A \rightsquigarrow \boldsymbol{\alpha}(x)$ to compute $[\![x]\!]^{\Gamma}_{\mathsf{tm}(A)} : (\![\Gamma]\!) \longrightarrow \mathsf{tm}^*(A)$, which by definition projects the appropriate atomic term from the vector $\gamma : (\![\Gamma]\!)$, embeds it into the neutrals, and finally reflects it in the chosen computability structure $[\![A]\!]\gamma$. □

(5.6.5∗3) *All normals and neutrals are stationary.* The following facts can be proved by simultaneous induction on normals and neutrals, using **(5.6.5∗2)** in the base case.
1) For any neutral $a : \mathsf{ne}(A)$, we have $[\![a]\!]_{\mathsf{tm}(A)} = \mathord{\uparrow}_{[\![A]\!]}a$.
2) For any normal $a : \mathsf{nf}(A)$, we have $a = \mathsf{nbe}_{\mathsf{tm}}(A, a)$.
3) For any normal $A : \mathsf{nftp}$, we have $A = \mathsf{nbe}_{\mathsf{tp}}(A)$.

(5.6.5∗4) *Normalization is surjective.* It is an immediate corollary of **(5.6.5∗3)** that the normalization functions are surjective: for instance, any normal form $A : \mathsf{nftp}$ is the image of its underlying term $A : \mathsf{tp}$ under $\mathsf{nbe}_{\mathsf{tp}}$.

(5.6.5∗5) *Normalization is an isomorphism.* Soundness **(5.6.3∗2)** states that the normalization function is injective; hence by surjectivity **(5.6.5∗4)**, the normalization function actually tracks a vertical isomorphism $\mathsf{tp} \cong \mathsf{nftp}$.

§5.7. RECURSION-THEORETIC RESULTS

(5.7∗1) Our normalization argument so far has been carried out in ordinary mathematics; we have not yet argued that the normalization algorithm is computable, nor have we shown that our normalization result implies the decidability of judgmental equality.

▦ **(5.7∗2)** Both the syntax of Martin-Löf's type theory as well as its normal forms may be presented by finitely many rules in a conventional deductive system. The specifics of such a "raw term presentation" are irrelevant, and the main point is to observe that the set of normal forms for a given term is recursively enumerable.

▦ **(5.7∗3)** *The function* $\mathsf{nbe_{tp}}$ *is externally recursive.* By **(5.6.3∗1)** and the verticality of the normalization function, the set of normal forms of a type $A : (\!(\Gamma)\!) \longrightarrow \mathsf{tp}$ is non-empty; by **(5.6.5∗5)** this set has exactly one element. Because it is recursively enumerable, we therefore have a terminating recursive function that takes a given type $A : (\!(\Gamma)\!) \longrightarrow \mathsf{tp}$ to its unique normal form $\mathsf{nbe}_{\mathsf{tp}}^{\Gamma}(A) : (\!(\Gamma)\!) \longrightarrow \mathsf{nftp}$.

▦ **(5.7∗4)** *Decidability of judgmental equality.* Hence it is effectively decidable whether two types $A, B : (\!(\Gamma)\!) \longrightarrow \mathsf{tp}$ are equal: first use the search algorithm **(5.7∗3)** to obtain normal forms of A and B, and then compare them. Equality of normal forms is clearly decidable (externally), because the inductive definition of normal forms in $\mathbf{Set_G}$ boils down to a finitary indexed inductive definition in $\mathbf{Set}$.

$$\mathsf{nftp} : \{\mathcal{U} \mid \P \hookrightarrow \mathsf{tp}\}$$
$$\mathsf{var},\mathsf{ne},\mathsf{nf} : \textstyle\prod_{A:\mathsf{tp}}\{\mathcal{U} \mid \P \hookrightarrow \mathsf{tm}(A)\}$$
$$\mathsf{nftp}_\diamond = \mathsf{nftp}$$
$$\mathsf{nftp}_n = \mathsf{nf}(\langle\uparrow^\diamond_{n+1}\rangle U_n)$$
$$\mathsf{netp}_n = \mathsf{ne}(\langle\uparrow^\diamond_{n+1}\rangle U_n)$$

$\mathbf{var} : \{A\}\ \{\mathsf{var}(A) \to \mathsf{ne}(A) \mid \P \hookrightarrow \lambda x.x\}$

$\mathbf{U}^\alpha_n : \{\mathsf{nftp}_\alpha \mid \P \hookrightarrow \langle\uparrow^\alpha_{n+1}\rangle U_n\}$ $\qquad\qquad$ for $n < \alpha$

$\mathbf{up}^\alpha_n : \{\mathsf{netp}_n \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \langle\uparrow^\alpha_n\rangle\}$ $\qquad\qquad$ for $n < \alpha$

$\widetilde{\mathbf{up}}_n : \prod_{A:\mathsf{netp}_n}\{\mathsf{ne}(\langle\uparrow^\diamond_n\rangle A) \to \mathsf{nf}(\langle\uparrow^\diamond_n\rangle A) \mid \P \hookrightarrow \lambda x.x\}$

$\mathbf{pi}_\alpha : \{(\sum_{A:\mathsf{nftp}_\alpha}(\mathsf{var}(\langle\uparrow^\diamond_\alpha\rangle A) \to \mathsf{nftp}_\alpha)) \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \Pi_\alpha\}$

$\mathbf{app} : \{A,B\}\ \{\mathsf{ne}(\Pi(A,B)) \to \prod_{x:\mathsf{nf}(A)}\mathsf{ne}(B(x)) \mid \P \hookrightarrow \lambda f.\lambda x.f(x)\}$

$\mathbf{lam} : \{A,B\}\ \{(\prod_{x:\mathsf{var}(A)}\mathsf{nf}(B(x))) \to \mathsf{nf}(\Pi(A,B)) \mid \P \hookrightarrow \lambda f.\lambda x.f(x)\}$

$\mathbf{sg}_\alpha : \{(\sum_{A:\mathsf{nftp}_\alpha}(\mathsf{var}(\langle\uparrow^\diamond_\alpha\rangle A) \to \mathsf{nftp}_\alpha)) \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \Sigma_\alpha\}$

$\mathbf{split} : \{A,B\}\ \{\mathsf{ne}(\Sigma(A,B)) \to \sum_{x:\mathsf{ne}(A)}\mathsf{ne}(B(x)) \mid \P \hookrightarrow \lambda p.(p.1,p.2)\}$

$\mathbf{pair} : \{A,B\}\ \{(\sum_{x:\mathsf{nf}(A)}\mathsf{nf}(B(x))) \to \mathsf{nf}(\Sigma(A,B)) \mid \P \hookrightarrow \lambda p.(p.1,p.2)\}$

$\mathbf{bool}_\alpha : \{\mathsf{nftp}_\alpha \mid \P \hookrightarrow \mathsf{bool}_\alpha\}$

$\mathbf{tt} : \{\mathsf{nf}(\mathsf{bool}) \mid \P \hookrightarrow \mathsf{tt}\}$

$\mathbf{ff} : \{\mathsf{nf}(\mathsf{bool}) \mid \P \hookrightarrow \mathsf{ff}\}$

$\mathbf{up}_{\mathsf{bool}} : \{\mathsf{ne}(\mathsf{bool}) \to \mathsf{nf}(\mathsf{bool}) \mid \P \hookrightarrow \lambda x.x\}$

$\mathbf{ind}_{\mathsf{bool}} : \{\prod_{C:\mathsf{var}(\mathsf{bool})\to\mathsf{nftp}}\prod_{c_0:\mathsf{nf}(C(\mathsf{tt}))}\prod_{c_1:\mathsf{nf}(C(\mathsf{ff}))}\prod_{x:\mathsf{ne}(\mathsf{bool})}\mathsf{ne}(C(x)) \mid \P \hookrightarrow \mathsf{ind}_{\mathsf{bool}}\}$

Figure 5.1: A summary of the normal form constants for Martin-Löf's type theory.

$$\textbf{record } \mathsf{tp}_\alpha^* : \{\mathcal{V}_{\mathsf{s}(\alpha)} \mid \P \hookrightarrow \mathsf{tp}_\alpha\} \textbf{ where} \qquad \Downarrow_{\mathsf{tp}}^\beta A := A.N_\beta$$
$$\quad \textbf{include } \mathsf{tp}_\alpha^\star \textbf{ as } C \qquad\qquad\qquad\qquad \Uparrow_A := A.\Uparrow$$
$$\quad N : \prod_{\beta > \alpha}\{\mathsf{nftp}_\beta \mid \P \hookrightarrow \langle\uparrow_\alpha^\beta\rangle C\} \qquad\qquad \Downarrow_A := A.\Downarrow$$
$$\quad \Uparrow : \{\mathsf{ne}(C) \to C \mid \P \hookrightarrow \lambda x.x\}$$
$$\quad \Downarrow : \{C \to \mathsf{nf}(C) \mid \P \hookrightarrow \lambda x.x\}$$

$$\boxed{\langle\uparrow_\alpha^\beta\rangle^* : \{\mathsf{tp}_\alpha^* \to \mathsf{tp}_\beta^* \mid \P \hookrightarrow \langle\uparrow_\alpha^\beta\rangle\}}$$

$$\langle\uparrow_\alpha^\beta\rangle^* A \Leftarrow \textbf{extend } \langle\uparrow_\alpha^\beta\rangle^\star A$$
$$\Downarrow_{\mathsf{tp}}^{\gamma\geq\beta}(\langle\uparrow_\alpha^\beta\rangle^* A) = \Downarrow_{\mathsf{tp}}^{\gamma\geq\alpha}A$$
$$\Uparrow_{\langle\uparrow_\alpha^\beta\rangle^* A}a = \Uparrow_A a$$
$$\Downarrow_{\langle\uparrow_\alpha^\beta\rangle^* A}a = \Downarrow_A a$$

$$\boxed{\mathsf{elim}_n^* : \{\mathsf{netp}_n \to \mathsf{tp}_n^* \mid \P \hookrightarrow \lambda A.A\}}$$

$$\mathsf{elim}_n^*(A).\mathsf{syn} = A$$
$$\mathsf{elim}_n^*(A).\mathsf{ext} = \mathsf{ne}(\langle\uparrow_n^\diamond\rangle A)$$
$$\Downarrow_{\mathsf{tp}}^{\beta\geq n}\mathsf{elim}_n^*(A) = \mathbf{up}_n^\beta(A)$$
$$\Uparrow_{\mathsf{elim}_n^*(A)}a = a$$
$$\Downarrow_{\mathsf{elim}_n^*(A)}a = \widetilde{\mathbf{up}}_n(A,a)$$

$$\boxed{U_n^* : \{\mathsf{tp}_{n+1}^* \mid \P \hookrightarrow U_n\}}$$

$$U_n^*.\mathsf{syn} = U_n$$
$$U_n^*.\mathsf{ext} = \mathsf{tp}_n^*$$
$$\Downarrow_{\mathsf{tp}}^{\beta\geq n+1}U_n^* = \mathbf{U}_\beta^n$$
$$\Uparrow_{U_n^*}A = \mathsf{elim}_n^*(A)$$
$$\Downarrow_{U_n^*}A = \Downarrow_{\mathsf{tp}}^n A$$

$$\boxed{\Pi_\alpha^* : \{(\textstyle\sum_{A:\mathsf{tp}_\alpha^*}(A \to \mathsf{tp}_\alpha^*)) \to \mathsf{tp}_\alpha^* \mid \P \hookrightarrow \Pi_\alpha\}}$$

$$\Pi_\alpha^*(A,B) \Leftarrow \textbf{extend } \Pi_\alpha^\star(A,B)$$
$$\Downarrow_{\mathsf{tp}}^{\beta\geq\alpha}\Pi_\alpha^*(A,B) = \mathbf{pi}_\beta\big(\Downarrow_{\mathsf{tp}}^\beta A, \lambda x.\Downarrow_{\mathsf{tp}}^\beta B(\Uparrow_A \mathbf{var}(x))\big)$$
$$\Uparrow_{\Pi_\alpha^*(A,B)}f = \lambda x.\Uparrow_{B(A(x))}\mathbf{app}(f,\Downarrow_A x)$$
$$\Downarrow_{\Pi_\mathcal{V}^*(A,B)}f = \mathbf{lam}(\lambda x.\Downarrow_{B(\Uparrow_A\mathbf{var}(x))}f(\Uparrow_A\mathbf{var}(x)))$$

$$\boxed{\Sigma_\alpha^* : \{(\textstyle\sum_{A:\mathsf{tp}_\alpha^*}(A \to \mathsf{tp}_\alpha^*)) \to \mathsf{tp}_\alpha^* \mid \P \hookrightarrow \Sigma_\alpha\}}$$

$$\Sigma_\alpha^*(A,B) \Leftarrow \textbf{extend } \Sigma_\alpha^\star(A,B)$$
$$\Downarrow_{\mathsf{tp}}^{\beta\geq\alpha}\Sigma_\alpha^*(A,B) = \mathbf{sg}_\beta\big(\Downarrow_{\mathsf{tp}}^\beta A, \lambda x.\Downarrow_{\mathsf{tp}}^\beta B(\Uparrow_A \mathbf{var}(x))\big)$$
$$(\Uparrow_{\Sigma_\alpha^*(A,B)}p).1 = \Uparrow_A \mathbf{split}(p).1$$
$$(\Uparrow_{\Sigma_\alpha^*(A,B)}p).2 = \Uparrow_{B(\Uparrow_A\mathbf{split}(p).1)}\mathbf{split}(p).2$$
$$\Downarrow_{\Sigma_\mathcal{V}^*(A,B)}p = \mathbf{pair}(\Downarrow_A p.1, \Downarrow_{B(p.1)}p.2)$$

$$\boxed{\mathsf{bool}_\alpha^* : \{\mathsf{tp}_\alpha^* \mid \P \hookrightarrow \mathsf{bool}_\alpha\}}$$

$$\mathsf{bool}_\alpha^* \Leftarrow \textbf{extend } \mathsf{bool}_\alpha^\star \langle\text{defined in } (\textbf{5.3}\ast\textbf{8})\rangle$$
$$\Downarrow_{\mathsf{tp}}^{\beta\geq\alpha}\mathsf{bool}_\alpha^* = \mathbf{bool}_\beta$$
$$\Uparrow_{\mathsf{bool}_\alpha^*}b = \mathbf{up}_{\mathsf{bool}}(b)$$
$$\Downarrow_{\mathsf{bool}_\alpha^*}\mathbf{tt} = \mathbf{tt}$$
$$\Downarrow_{\mathsf{bool}_\alpha^*}\mathbf{ff} = \mathbf{ff}$$
$$\Downarrow_{\mathsf{bool}_\alpha^*}\mathbf{up}_{\mathsf{bool}}(x) = \mathbf{up}_{\mathsf{bool}}(x)$$
$$\_\_ : \P \vdash \Downarrow_{\mathsf{bool}_\alpha^*}x = x$$

$$\boxed{\{\mathsf{ind}_{\mathsf{bool}}^* : \textstyle\prod_{C:\mathsf{bool}^*\to\mathsf{tp}^*}\prod_{c_0:C(\mathbf{tt}^*)}\prod_{c_1:C(\mathbf{ff}^*)}\prod_{x:\mathsf{bool}^*}C(x) \mid \P \hookrightarrow \mathsf{ind}_{\mathsf{bool}}\}}$$

$$\mathsf{ind}_{\mathsf{bool}}^*(C,c_0,c_1,\mathbf{tt}) = c_0$$
$$\mathsf{ind}_{\mathsf{bool}}^*(C,c_0,c_1,\mathbf{ff}) = c_1$$
$$\mathsf{ind}_{\mathsf{bool}}^*(C,c_0,c_1,\mathbf{up}_{\mathsf{bool}}(x)) = \Uparrow_{C(\Uparrow_{\mathsf{bool}^*}x)}\mathbf{ind}_{\mathsf{bool}}\big(\Downarrow_{\mathsf{tp}}^\diamond C(\Uparrow_{\mathsf{bool}^*}x), \Downarrow_{C(\mathbf{tt})}c_0, \Downarrow_{C(\mathbf{ff})}c_1, x\big)$$
$$\_\_ : \P \vdash \mathsf{ind}_{\mathsf{bool}}^*(C,c_0,c_1,x) = \mathsf{ind}_{\mathsf{bool}}(C,c_0,c_1,x)$$

Figure 5.2: A summary of the normalization structures for Martin-Löf's type theory.

Part IV

Cubical Type Theory

CARTESIAN CUBICAL TYPE THEORY

※ **(6.0∗1)** Cubical type theories extend Martin-Löf's basic type theory with the structure of an *interval* $\mathbb{I}$ to capture a profound form of *identification* that is both syntactically *and* semantically better-behaved than the one provided by extensional equality types. Unfortunately, this additional structure was incompatible in important ways with both the conventional methods for proving canonicity *as well as* the state-of-the-art methods for proving normalization (Chapter 5).

🌼 **(6.0∗2)** There are several variants of cubical type theory, each corresponding to a different version of the interval. We will mainly focus on the so-called *Cartesian* version of cubical type theory [Ang+19], but our work also applies to the De Morgan variant [Coh+17]. For overviews of several different cube categories and their respective intervals, see Awodey [Awo15] and Buchholtz and Morehouse [BM17].

§6.1. THE INTERVAL IN CUBICAL TYPE THEORY

▪ **(6.1∗1)** *The Cartesian interval.* Cubical type theory begins by adding a judgmental *interval* with two endpoints:

$$\mathbb{I} : \square$$
$$0, 1 : \mathbb{I}$$

It is important for our results that the interval is not a type but a judgment.

(6.1∗2) *Other variants of the interval.* The *Dedekind* variant of cubical type theory enriches the interval with the structure of a bounded distributive lattice with $0 \sqcap r = 0$, $1 \sqcap r = r$, $0 \sqcup x = x$, and $1 \sqcup x = 1$. The De Morgan variant of cubical type theory adds to this an involution $\sim\; : \mathbb{I} \longrightarrow \mathbb{I}$ such that $\sim(r \sqcap s) = \sim r \sqcup \sim s$.

(6.1∗3) *The interval as a figure shape.* The purpose of the interval in cubical type theory is to provide a "shape" for drawing figures in other types. For instance, a function $f : \mathbb{I} \to A$ can be thought of as drawing a *line* figure in A:

$$f : f(0) \longrightarrow f(1)$$

It is appropriate to think of f above as an *identification* of the endpoint $f(0)$ with the endpoint $f(1)$; such an identification is called a "path". Of course, depending on the structure of A there may be any number of such paths. Generalizing to a higher dimension, a function $f : \mathbb{I} \times \mathbb{I} \to A$ draws a *square* figure in A; working generically over $i,j : \mathbb{I}$ we can visualize such a square as follows:

$$
\begin{array}{ccc}
f(0,0) & \xrightarrow{\;f(i,0)\;} & f(1,0) \\
{\scriptstyle f(0,j)}\big\downarrow & f(i,j) & \big\downarrow{\scriptstyle f(1,j)} \\
f(0,1) & \xrightarrow[\;f(i,1)\;]{} & f(1,1)
\end{array}
$$

The surface $f(i,j)$ can be thought of as either an identification $f(0,j) \longrightarrow f(1,j)$ or as an identification $f(i,0) \longrightarrow f(i,1)$.

(6.1∗4) *Heterogeneity.* The interval also provides a *heterogeneous* version of identification: starting with a line of judgments $A : \mathbb{I} \to \square$, the dependent product $\prod_{i:\mathbb{I}} A(i)$ classifies lines in the *total space* of A between a point of $A(0)$ and a point of $A(1)$.

§6.2. JUDGMENTAL UNIVERSE OF COFIBRATIONS

(6.2∗1) Cubical type theory adds a notion of *cofibration* or *cofibrant proposition* classified by a judgmental universe:

$$\mathbb{F} : \square$$
$$[-] : \mathbb{F} \to \square$$

The following two axioms ensure that (1) the judgments classified by $\mathbb{F}$ are proof-irrelevant propositions, and (2) that codes for proof-irrelevant propositions in $\mathbb{F}$ are unique whenever they exist:

$$- : \prod_{\phi:\mathbb{F}} \prod_{p,q:[\phi]} p =_{[\phi]} q$$

$$\_ : \textstyle\prod_{\phi,\psi:\mathbb{F}}([\phi] \cong [\psi]) \to \phi =_\mathbb{F} \psi$$

The second axiom above is sometimes called *propositional univalence*.

(6.2∗2) The name *cofibration* is inspired by model category theory, but it shouldn't be taken too seriously as far as the syntax of cubical type theory is concerned.

(6.2∗3) Given $\phi : \mathbb{F}$, we will in some cases write ϕ for the judgment $[\phi]$ where it causes no ambiguity.

(6.2∗4) *Closure under dimension equality.* The universe of cofibrations is closed under equality of dimensions:

$$(=) : \mathbb{I} \times \mathbb{I} \to \mathbb{F}$$
$$\_ : \textstyle\prod_{r,s:\mathbb{I}}(r =_\mathbb{I} s) \cong [r = s]$$

The second axiom above expresses an *equality reflection* principle for dimension equality; this is the source of many of the difficulties in normalizing cubical type theory.

(6.2∗5) *Closure under dimensional quantification.* $\mathbb{F}$ is closed under universal quantification over the interval:

$$(\forall_\mathbb{I}) : (\mathbb{I} \to \mathbb{F}) \to \mathbb{F}$$
$$\_ : \textstyle\prod_{\phi:\mathbb{I}\to\mathbb{F}}(\textstyle\prod_{i:\mathbb{I}}[\phi(i)]) \cong [\forall_\mathbb{I}(\phi)]$$

(6.2∗6) *Closure under conjunction.* $\mathbb{F}$ is closed under conjunction:

$$(\sqcap) : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$$
$$\_ : \textstyle\prod_{\phi,\psi:\mathbb{F}}([\phi] \times [\psi]) \cong [\phi \sqcap \psi]$$

(6.2∗7) We will close $\mathbb{F}$ under a form of disjunction; disjunction is however a kind of colimit, which takes us outside the logical framework that we developed in Chapter 1. Therefore, we will add the code for disjunctions and then later in this section express a *locality* condition that equips a given judgment with a strict elimination rule for disjunctions of cofibrations.

(6.2∗8) We add a code for disjunctions of cofibrations to the signature:

$$(\sqcup) : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$$
$$\_ : \textstyle\prod_{\phi,\psi:\mathbb{F}}\{\phi\}\,[\phi \sqcup \psi]$$
$$\_ : \textstyle\prod_{\phi,\psi:\mathbb{F}}\{\psi\}\,[\phi \sqcup \psi]$$

(6.2∗9) Write $\curlywedge := (0 = 1)$ and $\top := (0 = 0)$. The reason for writing $\curlywedge$ rather than $\bot$ is to avoid confusion: it is not the case that $\bot$ is the initial object of our syntactic category, but **(6.2∗11)** defines a class of objects that treat it as if it is.

(6.2∗10) Given $r : \mathbb{I}$, we will write ∂r for the *boundary* of r, namely $r = 0 \sqcup r = 1$.

(6.2∗11) *⅃-connectedness.* Let $X : \square$ be an arbitrary form of judgment; we say that X is ⅃-connected when $X \times \mathop{⅃} \cong \mathop{⅃}$. In other words, we have a partial isomorphism $\{⅃\} X \cong \mathbb{1}$. Assuming $⅃$, we will write $[]$ for the unique element of X.

(6.2∗12) $\mathbb{F}$-*locality.* Let $X : \square$ be an arbitrary form of judgment; we say that X is $\mathbb{F}$-*local* when X is ⅃-connected and for any $\phi,\psi : \mathbb{F}$, the collection of partial elements $\{\phi \sqcup \psi\} X$ is isomorphic to the collection of *matching families* below:

$$\{x_0 : \{\phi\} X, x_1 : \{\psi\} X \mid \{\phi \sqcap \psi\} x_0 =_X x_1\}$$

We will write $[\phi \hookrightarrow x_0, \psi \hookrightarrow x_1]$ for the element of X determined by a matching family assuming $\phi \sqcup \psi$.

(6.2∗13) The data of an $\mathbb{F}$-locality structure can be expressed as a "snippet" isLocal : $\square \to \square$ in the logical framework.

(6.2∗14) We assert that the interval, the collection of cofibrations, and the extent of a given cofibration are all $\mathbb{F}$-local:

$$\_\_ : \mathsf{isLocal}(\mathbb{I})$$
$$\_\_ : \mathsf{isLocal}(\mathbb{F})$$
$$\_\_ : \textstyle\prod_{\phi:\mathbb{F}}\mathsf{isLocal}([\phi])$$

§6.3. COMPOSITION STRUCTURE

(6.3∗1) *Homogeneous composition structure.* Let $A : \square$ be a sort; a *homogeneous composition structure* on A is defined to be an element of the folowing sort:

$$\mathsf{HCom}(A) = \textstyle\prod_{r,s:\mathbb{I}}\prod_{\phi:\mathbb{F}}\prod_{a:\prod_{i:\mathbb{I}}\{i=r\sqcup\phi\}A}\{A \mid r = s \sqcup \phi \hookrightarrow a(s)\}$$

Given $h : \mathsf{HCom}(A)$, we will write $h^{r \rightsquigarrow s;\phi}a$ for $h(r,s,\phi,a)$.

(6.3∗2) *Coercion structure.* Let $A : \mathbb{I} \to \square$ be a line of sorts; a *coercion structure* on A is defined to be an element of the following sort:

$$\mathsf{Coe}(A) = \textstyle\prod_{r,s:\mathbb{I}}\prod_{a:A(r)}\{A(s) \mid r = s \hookrightarrow a\}$$

Given $c : \mathsf{Coe}(A)$, we will write $c^{r \rightsquigarrow s}a$ for $c(r,s,a)$.

(6.3∗3) *Composition structure.* Let $A : \mathbb{I} \to \square$ be a line of sorts; a *composition structure* on A is defined to be an element of the following sort:

$$\mathsf{Com}(A) = \textstyle\prod_{r,s:\mathbb{I}}\prod_{\phi:\mathbb{F}}\prod_{a:\prod_{i:\mathbb{I}}\{i=r\sqcup\phi\}A(i)}\{A(s) \mid r = s \sqcup \phi \hookrightarrow a(s)\}$$

Given $c : \mathsf{Com}(A)$, we will write $c^{r \rightsquigarrow s; \phi} a$ for $c(r, s, \phi, a)$.

(6.3*4) *Composition from coercion and homogeneous composition.* Let $A : \mathbb{I} \to \square$ be a line of sorts; given a coercion operation $c : \mathsf{Coe}(A)$ and a line of homogeneous composition operations $h_\bullet : \prod_{i:\mathbb{I}} \mathsf{HCom}(A(i))$, we may define a composition operation $\mathsf{mk\text{-}com}(c, h_\bullet) : \mathsf{Com}(A)$ as follows:

$$\mathsf{mk\text{-}com}(c, h_\bullet)^{r \rightsquigarrow s; \phi} a = h_s^{r \rightsquigarrow s; \phi} \lambda i . c^{i \rightsquigarrow s} a(i)$$

§6.4. TYPE STRUCTURE AND CONNECTIVES

(6.4*1) Cubical type theory is an extension of the signature of basic Martin-Löf type theory; we therefore include this signature, adding to it the assertion that its various sorts are all $\mathbb{F}$-local:

$$\textbf{include } \mathbb{ML}_{base}$$
$$\_ : \mathsf{isLocal}(\mathsf{tp}_\alpha)$$
$$\_ : \textstyle\prod_{A:\mathsf{tp}} \mathsf{isLocal}(\mathsf{tm}(A))$$

(6.4*2) For readability, we will write A for $\mathsf{tm}_\alpha(A)$ when it does not cause ambiguity.

(6.4*3) *Coercion and homogeneous composition.* Following Angiuli, Brunerie, Coquand, Hou (Favonia), Harper, and Licata [Ang+19] and Angiuli, Hou (Favonia), and Harper [AHH18] we add generic coercion and homogeneous composition operations:

$$\mathsf{coe}_\bullet : \textstyle\prod_{A:\mathbb{I}\to\mathsf{tp}} \mathsf{Coe}(A)$$
$$\mathsf{hcom}_\bullet : \textstyle\prod_{A:\mathsf{tp}} \mathsf{HCom}(A)$$

Composition is then a defined notion:

$$\mathsf{com}_\bullet : \textstyle\prod_{A:\mathbb{I}\to\mathsf{tp}} \mathsf{Com}(A)$$
$$\mathsf{com}_A = \mathsf{mk\text{-}com}(\mathsf{coe}_A, \lambda i . \mathsf{hcom}_{A(i)})$$

(6.4*4) *Dependent products and sums.* Equations are added to govern the computational behavior of coercion and composition on the existing connectives:

$$\mathsf{coe}^{r \rightsquigarrow s}_{\lambda j . \Pi(A(j), B(j,-))} f = \lambda x . \mathsf{coe}^{r \rightsquigarrow s}_{\lambda j . B(j, \mathsf{coe}^{s \rightsquigarrow j}_A x)} \mathsf{coe}^{s \rightsquigarrow r}_A x$$
$$\mathsf{coe}^{r \rightsquigarrow s}_{\lambda j . \Sigma(A(j), B(j,-))} (a, b) = \left(\mathsf{coe}^{r \rightsquigarrow s}_A a, \mathsf{coe}^{r \rightsquigarrow s}_{\lambda j . B(j, \mathsf{coe}^{r \rightsquigarrow j}_A a)} b \right)$$
$$\mathsf{hcom}^{r \rightsquigarrow s; \phi}_{\Pi(A,B)} f = \lambda x . \mathsf{hcom}^{r \rightsquigarrow s; \phi}_{B(x)} \lambda i . f(i, x)$$
$$\mathsf{hcom}^{r \rightsquigarrow s; \phi}_{\Sigma(A,B)} p = \left(\mathsf{hcom}^{r \rightsquigarrow s; \phi}_A (\lambda i . p(i).1), \mathsf{com}^{r \rightsquigarrow s; \phi}_{\lambda j . B(\mathsf{hcom}^{r \rightsquigarrow j; \phi}_A (\lambda i . p(i).1))} (\lambda i . p(i).2) \right)$$

(6.4*5) *Weak booleans.* We add equations to govern the behavior of coercion and composition for the (weak) booleans.

$$\mathsf{coe}^{r \rightsquigarrow s}_{\lambda j . \mathsf{bool}} A = A$$

$$\mathsf{ind_{bool}}\left(C,c_0,c_1,\mathsf{hcom}^{r\rightsquigarrow s;\phi}_{\mathsf{bool}}b\right) = \mathsf{com}^{r\rightsquigarrow s;\phi}_{\lambda j.C(\mathsf{hcom}^{r\rightsquigarrow j;\phi}_{\mathsf{bool}}b)}\lambda i.\mathsf{ind_{bool}}\left(C,c_0,c_1,b(i)\right)$$

◼ **(6.4∗6)** *The circle.* We will add a single higher inductive type, the circle S^1.

$$\mathsf{S}^1_\alpha : \mathsf{tp}_\alpha$$
$$\mathsf{base} : \mathsf{S}^1$$
$$\mathsf{loop} : \textstyle\prod_{i:\mathbb{I}}\{\mathsf{S}^1 \mid \partial i \hookrightarrow \mathsf{base}\}$$
$$\mathsf{ind_{S^1}} : \textstyle\prod_{C:\mathsf{S}^1\rightarrow\mathsf{tp}}\prod_{c_b:C(\mathsf{base})}\prod_{c_l:\prod_{i:\mathbb{I}}\{C(\mathsf{loop}(i))|\partial i\hookrightarrow c_b\}}\prod_{x:\mathsf{S}^1}C(x)$$

We add the following computation rules for the induction principle:

$$\mathsf{ind_{S^1}}\left(C,c_b,c_l,\mathsf{base}\right) = c_b$$
$$\mathsf{ind_{S^1}}\left(C,c_b,c_l,\mathsf{loop}(i)\right) = c_l(i)$$

The following equations govern universe lifting, coercion, and composition:

$$\mathsf{S}^1_\alpha = \langle\uparrow^\alpha_0\rangle\mathsf{S}^1_0$$
$$\mathsf{coe}^{r\rightsquigarrow s}_{\lambda\_.\mathsf{S}^1}x = x$$
$$\mathsf{ind_{S^1}}\left(C,c_b,c_l,\mathsf{hcom}^{r\rightsquigarrow s;\phi}_{\mathsf{S}^1}x\right) = \mathsf{com}^{r\rightsquigarrow s;\phi}_{\lambda i.C(\mathsf{hcom}^{r\rightsquigarrow i;\phi}_{\mathsf{S}^1}x)}\lambda i.\mathsf{ind_{S^1}}\left(c,c_b,c_l,x(i)\right)$$

◼ **(6.4∗7)** *The path type.* We add a connective for the path type to each universe level:

$$\mathsf{path}_\alpha : \left(\textstyle\sum_{A:\mathbb{I}\rightarrow\mathsf{tp}_\alpha}\prod_{i:\mathbb{I}}\{\partial i\}\, A(i)\right) \rightarrow \mathsf{tp}_\alpha$$
$$\_ : \{A,a\}\ \left(\textstyle\prod_{i:\mathbb{I}}\{A(i) \mid \partial i \hookrightarrow a\}\right) \cong \mathsf{path}(A,a)$$

The following equations govern universe lifting, coercion, and composition:

$$\langle\uparrow^\beta_\alpha\rangle\mathsf{path}_\alpha(A,a) = \mathsf{path}_\beta(\langle\uparrow^\beta_\alpha\rangle \circ A, a)$$
$$\mathsf{coe}^{r\rightsquigarrow s}_{\lambda j.\mathsf{path}_\diamond(A(j,-),a(j,-))}p = \lambda i.\mathsf{com}^{r\rightsquigarrow s;\partial i}_{\lambda j.A(j,i)}\lambda j.[j = r \hookrightarrow p(i), \partial i \hookrightarrow a(j,i)]_{A(j,i)}$$
$$\mathsf{hcom}^{r\rightsquigarrow s;\phi}_{\mathsf{path}_\diamond(A,a)}p = \lambda j.\mathsf{hcom}^{r\rightsquigarrow s;\phi\sqcup\partial i}_{A(i)}\lambda j.[j = r \sqcup \phi \hookrightarrow p(j), \partial i \hookrightarrow a(i)]$$

⬇ **(6.4∗8)** *Path notation.* We define the following more convenient notation for path types:

$$x \sim_{i.A(i)} y := \mathsf{path}_\alpha(A, \lambda i.[i = 0 \hookrightarrow x, i = 1 \hookrightarrow y])$$
$$x \sim_A y := x \sim_{\_.A} y$$

⬇ **(6.4∗9)** To prepare for specifying the *glue type* we define the following notations for contractibility and type theoretic equivalences:

$$\mathsf{isContr}_\alpha : \mathsf{tp}_\alpha \rightarrow \mathsf{tp}_\alpha$$
$$\mathsf{isContr}_\alpha(A) = \Sigma_\alpha(A, \lambda x.\Pi_\alpha(A, \lambda y.x \sim_A y))$$

$$\mathsf{Equiv}_\alpha : \mathsf{tp}_\alpha \times \mathsf{tp}_\alpha \to \mathsf{tp}_\alpha$$
$$\mathsf{Equiv}_\alpha(A,B) := \Sigma_\alpha(\Pi_\alpha(A,\lambda\_.B), \lambda f.\Pi_\alpha(B, \lambda b.\mathsf{isContr}_\alpha(\Sigma(A, \lambda a.f(a) \sim_B b))))$$

We will treat the projection $\mathsf{Equiv}_\alpha(A,B) \longrightarrow \Pi_\alpha(A,\lambda\_.B)$ implicitly.

(6.4∗10) *Gluing data.* The data involved in the formation of the glue type **(6.4∗11)** below is somewhat complex; hence we define a record to encapsulate it.

> **record** $\mathsf{Desc}_\alpha : \square$ **where**
> $\quad$ phi $: \mathbb{F}$
> $\quad$ B $: \mathsf{tp}_\alpha$
> $\quad$ A $: \{\phi\}\,\mathsf{tp}_\alpha$
> $\quad$ f $: \{\phi\}\,\mathsf{Equiv}(A,B)$

(6.4∗11) *The glue type.* Univalence as well as composition in the universe are implemented by the following *glue* type connective:

$$\mathsf{glue}_\alpha : \prod_{D:\mathsf{Desc}_\alpha}\{\mathsf{tp}_\alpha \mid D.\phi \hookrightarrow D.A\}$$
$$\_ : \{D\}\ \{(\textstyle\sum_{a:\{D.\phi\}\,D.A}\{D.B \mid D.\phi \hookrightarrow D.f(a)\}) \cong \mathsf{glue}_\diamond(D) \mid D.\phi \hookrightarrow -.1\}$$
$$\_ : \{D\}\ \langle\uparrow_\alpha^\beta\rangle\mathsf{glue}_\alpha(D) =_{\mathsf{tp}_\beta} \mathsf{glue}_\beta(D.\phi,\langle\uparrow_\alpha^\beta\rangle D.B,\langle\uparrow_\alpha^\beta\rangle D.A,D.f)$$

(6.4∗12) Given $B : \mathsf{tp}_\alpha$, $A : \{\phi\}\,\mathsf{tp}_\alpha$ and $f : \{\phi\}\,\mathsf{Equiv}_\alpha(A,B)$, we will write $\mathsf{glue}_\alpha\,[B \mid \phi \hookrightarrow (A,f)]$ for $\mathsf{glue}_\alpha(\phi,B,A,f)$. Given $a : \{\phi\}\,A$ and $b : \{B \mid \phi \hookrightarrow f(a)\}$, we will write $\mathsf{englue}\,[b \mid \phi \hookrightarrow a]$ for the corresponding element of $\mathsf{glue}_\alpha\,[B \mid \phi \hookrightarrow (A,f)]$. Given an element $g : \mathsf{glue}_\alpha\,[B \mid \phi \hookrightarrow (A,f)]$, we will write $\mathsf{unglue}(g)$ for the corresponding element of B.

(6.4∗13) *Coercion and composition in the glue type.* The equational laws for coercion and composition in the glue type are very complex, so we refer the reader to their definition by Angiuli, Brunerie, Coquand, Hou (Favonia), Harper, and Licata [Ang+19].

(6.4∗14) *Composition in the universe.* We can define composition in the universe in terms of the glue type, using the fact that coercion is an equivalence.

$$\mathsf{hcom}_{\mathsf{U}_n}^{r \rightsquigarrow s;\phi}A = \mathsf{glue}_n\,\big[A(r) \mid \phi \sqcup (r = s) \hookrightarrow \big(A(s), \big[\phi \hookrightarrow \big(\mathsf{coe}_{\langle\uparrow_n^\diamond\rangle \circ A}^{s \rightsquigarrow r}\,-\,,\ldots\big), r = s \hookrightarrow (\mathsf{id},\ldots)\big]\big)\big]$$

The ellipses above are filled by (coherent) proofs that coercion and the identity function are equivalences; see [Ang+19] for the details.

§6.5. ASPECTS OF CUBICAL COMPUTATION

§6.5.1. Cubical computation and canonical forms

(6.5.1∗1) The study of cubical type theory has since its inception been motivated by essentially computational questions:

1) Can terms in cubical type theory be run like programs with observable results?

2) Can equality of types and terms be decided by a computerized implementation of cubical type theory?

Long before anybody had an inkling of how to *resolve* these questions, they played an important role in guiding the design of the more difficult aspects of cubical type theory. For instance, it is easy to design a version of cubical type theory that treats the composition operation axiomatically, and such a type theory is homotopically well-behaved (as shown by Coquand, Huber, and Sattler [CHS19]); but to support the view of terms as *running programs* it was necessary to include equations that govern the computational behavior of composition at every type connective [Ang+19; AHH18; Coh+17].

(6.5.1∗2) *Semantic and syntactic aspects of coherence.* Designing the computational behavior of composition was highly non-trivial: while any two implementations of composition automatically agree up to homotopy, exhibiting any specific implementation is quite difficult because of various *coherence* or *stability* requirements that are built into dependent type theory. For instance, one may define a composition operation for the collection of elements of the **glue** type; but we recall that **glue** $[B \mid \phi \hookrightarrow A]$ will compute to A if $\phi = \top$, hence it is necessary that our implementation of composition for **glue** $[B \mid \phi \hookrightarrow A]$ agree with the existing implementation of composition for A under ϕ. This property can be seen to be an instance of the general principle that equations in type theory must be stable under substitution.

The semantic models of cubical type theory were developed prior to the introduction of cubical type theory itself. Hence the computational behavior of composition was originally designed from a semantic perspective (denotationally by Cohen, Coquand, Huber, and Mörtberg [Coh+17] and operationally by Angiuli, Hou (Favonia), and Harper [AHH18]); in the latter setting, the coherence requirement can be seen as a kind of "confluence": one must ultimately get the same result by composing in **glue** $[B \mid \top \hookrightarrow A]$ and then reducing, or by reducing and then composing in A.

The syntactic perspective on the same phenomenon is different: the boundary equations for **glue** and for composition are imposed at the outset, and then the question is whether it is possible to control their consequences. Adding further equations that reduce compositions may imply unexpected identifications (by transitivity and congruence). This shows the importance of studying *models* of a theory, in which the interpretations of the various constructs and the consequences of the equational theory are made concrete.

(6.5.1∗3) *Cubical operational computation.* The first substantive results in cubical computation were achieved by Angiuli, Hou (Favonia), and Harper [AHH18] and Huber [Hub18]. Huber proved an operational *canonicity* theorem for cubical type theory; and Angiuli, Hou (Favonia), and Harper proved a related result that establishes the compatibility of cubical type theory with a *computational interpretation* of terms as running programs.

■ **(6.5.1∗4)** We define an *operational notion of computation* by the following data:

1) A nonempty subset $\mathcal{C}$ of the collection of contexts.
2) A reduction relation M $\longrightarrow_\Gamma$ N ranging over untyped terms M,N that are well-scoped in $\Gamma \in \mathcal{C}$.
3) A distinguished set of types $\mathcal{O}_\Gamma$ in contexts $\Gamma \in \mathcal{C}$.
4) For each $O \in \mathcal{O}_\Gamma$, a set of observables $\mathcal{V}_\Gamma^O$ ranging over terms $\Gamma \vdash V : O$.

◆ **(6.5.1∗5)** *Closed computation.* Choosing $\mathcal{C}$ to be the singleton spanned by the empty context, we define ($\longrightarrow$) to be the ordinary untyped operational semantics of the type theory; we then choose $\mathcal{O} = \{\mathsf{bool}\}$ and $\mathcal{V} = \{\mathsf{tt,ff}\}$.

◆ **(6.5.1∗6)** *Cubical computation.* Choose $\mathcal{C}$ to be the set of purely cubical contexts Ψ generated by finite powers of the interval; let ($\longrightarrow_\Psi$) be the untyped operational semantics of cubical type theory [AHH18]. We then choose $\mathcal{O}_\Psi$ to be the singleton spanned by a base type (such as the booleans or the circle), and we choose $\mathcal{V}_\Psi^{\mathsf{bool}}$ to be the set containing $\{\mathsf{tt,ff}\}$ — and potentially additional terms corresponding to formal compositions.

■ **(6.5.1∗7)** *Operational canonicity.* Let $(\mathcal{C}, \longrightarrow, \mathcal{O}, \mathcal{V})$ be an operational notion of computation; relative to these data, an *operational canonicity* result states that for any $\Gamma \vdash M : O$ with $\Gamma \in \mathcal{C}$ and $O \in \mathcal{P}_\Gamma$, there exists $V \in \mathcal{V}_\Gamma^O$ such that both M $\longrightarrow_\Gamma$ V and $\Gamma \vdash M = V : O$.

■ **(6.5.1∗8)** *Computational interpretation.* A computational interpretation relative to an operational notion of computation $(\mathcal{C}, \longrightarrow, \mathcal{O}, \mathcal{V})$ is a *model* of the type theory in which for each $\Gamma \in \mathcal{C}$ and $O \in \mathcal{O}_\Gamma$, the interpretation $\mathsf{Tm}_{[\![\Gamma]\!]}([\![O]\!])$ is a subquotient of the collection $\{M \mid \exists V \in \mathcal{V}_\Gamma(O).M \longrightarrow_\Gamma V\}$.

■ **(6.5.1∗9)** *Canonicity implies computational interpretation.* Let $(\mathcal{C}, \longrightarrow, \mathcal{O}, \mathcal{V})$ be an operational notion of computation satisfying operational canonicity **(6.5.1∗7)**. The canonicity result establishes the compatibility of the type theory with a form of semantics in which the types $O \in \mathcal{O}_\Gamma$ are interpreted by saturating the well-typed observables $\mathcal{V}_\Gamma^O$ under converse evaluation **(6.5.1∗15)**, or a *coherent* variation thereof **(6.5.1∗16)**.

■ **(6.5.1∗10)** *Canonicity from computational interpretation.* In some but not all cases, a computational interpretation can be used to obtain a canonicity result. Assume the following additional properties of the notion of computation and the computational interpretation, fixing $\Gamma \in \mathcal{C}$ and $O \in \mathcal{O}_\Gamma$:

1) *Subject reduction.* If $\Gamma \vdash M : O$ and M $\longrightarrow_\Gamma$ N, then $\Gamma \vdash N : O$ and $\Gamma \vdash M = N : O$.
2) *Alignment.* If $\Gamma \vdash M : O$, then $\Gamma \vdash [\![M]\!] : O$ and $\Gamma \vdash M = [\![M]\!] : O$.

Under these additional assumptions, it is easy to see that canonicity holds. Of course, it is not difficult to define a computational interpretation in which both subject reduction and alignment are violated — hence while the existence of a computational interpretation is of independent interest for philosophical purposes, it is strictly weaker than canonicity.

(6.5.1∗11) It is somewhat difficult to compare the results of Huber [Hub18] with those of Angiuli, Hou (Favonia), and Harper [AHH17; AHH18]. The authors of *op. cit.* had a different motivation that did not entail specifying which type theory they have constructed a model of, and moreover there are rules in the operational semantics of *op. cit.* that cannot be presented by type-theoretic equations on open terms without violating the syntactical presuppositions of the open judgments in a traditional presentation. In particular, the algorithm for coercion in V-types presented in *op. cit.* is only well-typed in purely cubical contexts $\Psi \in \mathcal{C}$. Conventionally, the rules of type theory must make sense in *arbitrary* contexts, so it is not a surprise that this difference in perspective led to bugs[1] in the original RedPRL implementation of computational cubical type theory. This can be fixed by adopting a different algorithm for coercion, *e.g.* that of Angiuli [Ang19].

(6.5.1∗12) Modulo the adjustments to the coercion algorithm described above **(6.5.1∗11)**, it is possible to view the semantics given by Angiuli, Hou (Favonia), and Harper [AHH18] as a computational interpretation of a version of Cartesian cubical type theory à la [Ang+19], in the sense of **(6.5.1∗6)** and **(6.5.1∗8)**, a perspective worked out in more detail by Angiuli [Ang19]. Although Angiuli does not verify the additional assumptions needed to turn this into a canonicity result **(6.5.1∗10)**, these assumptions are likely to be true regardless, or at least validated under minor adjustments to the argument. Huber [Hub18] defines a similar computational interpretation of cubical type theory for which it is more direct to validate these assumptions and in this way obtains his canonicity result.

(6.5.1∗13) *Instability of untyped computation.* The untyped operational semantics of ordinary Martin-Löf type theory is particularly well-adapted: while the operational semantics is usually defined on closed terms, the same rules make sense for open terms as well and exhibit some remarkable properties for an *arbitrary* context Γ and an untyped term M well-scoped in Γ. In addition to subject reduction, Martin-Löf type theory's operational semantics involves the following stability property:

If $\gamma : \Delta \longrightarrow \Gamma$ is an arbitrary substitution and $M \longrightarrow_\Gamma N$, then $\gamma^*M \longrightarrow_\Delta \gamma^*N$.

Unfortunately, the deterministic operational semantics of cubical type theory cannot satisfy untyped stability — in fact, cubical computation is not even stable when restricted to purely cubical contexts $\Psi = [i : \mathbb{I}, j : \mathbb{I}, \dots]$. To see that this is the case, consider the following untyped composite term varying in a cubical context $\Psi, i : \mathbb{I}$:

$$\Psi, i : \mathbb{I} \mid M := \mathsf{hcom}_{\mathsf{bool}}^{0 \rightsquigarrow i; \top} \lambda k.[k = 0 \hookrightarrow \mathsf{tt}, \top \hookrightarrow \mathsf{ff}]$$

Assuming the deterministic operational semantics of [AHH18], we have $M \longrightarrow_{\Psi, i : \mathbb{I}} \mathsf{ff}$ and $\langle 0/i \rangle^*M \longrightarrow_\Psi \mathsf{tt}$, but we do not have $\langle 0/i \rangle^*M \longrightarrow_\Psi \langle 0/i \rangle^*\mathsf{ff}$. Of course, the term that we started with was not well-typed, and it can be seen that ill-typedness is the source of

---

[1] See https://github.com/RedPRL/sml-redprl/issues/692

all such pathologies; for this reason, it is appropriate to say that computation in cubical type theory is inherently typed, in contrast to ordinary type theory.

✳ **(6.5.1∗14)** Many difficulties in the subjective metatheory of cubical type theory can be avoided by defining operational semantics over typed terms (as in Huber [Hub18]) rather than untyped terms; in particular, there is no difficulty with stability **(6.5.1∗13)** for typed terms. We will see below that there are additional difficulties that can only be overcome by passing to the *objective* metatheory, where terms are quotiented by judgmental equality.

☕ **(6.5.1∗15)** *Failure of head expansion.* Head expansion or *converse evaluation* is a property of the computational interpretation of a given type as a relation: if $N \in [\![A]\!]$ and $M \longrightarrow N$, then $M \in [\![A]\!]$ and $M = N \in [\![A]\!]$. The computational interpretation of Martin-Löf type theory can be closed under this condition, and this observation is the work-horse of the soundness theorem. Unfortunately we may deduce from the same counterexample that we used for stability **(6.5.1∗13)** that a non-trivial computational interpretation of cubical type theory cannot be closed under head expansion, because it implies $\mathsf{tt} = \mathsf{ff} \in [\![\mathsf{bool}]\!]$.

✳ **(6.5.1∗16)** *Coherent expansion.* Independently, both Angiuli [Ang19] and Huber [Hub18] have discovered a higher-dimensional generalization of head expansion called *coherent expansion*. Coherent expansion is a work-horse lemma that plays a role in the subjective metatheory of cubical type theory analogous to that of head expansion in the subjective metatheory of Martin-Löf type theory. Unfortunately, the resulting proofs are quite complex and their authors concluded after subsequent investigations that the subjective metatheory could not practically be scaled to more difficult results, such as normalization.

❧ **(6.5.1∗17)** *Objective metatheory.* If one could avoid the need to consider head expansion of any form whatsoever, then surely the burdensome consequences of coherent expansion would also disappear from the metatheory of cubical type theory. This was the prediction of Angiuli that in part motivated Sterling, Angiuli, and Gratzer [SAG19] to reformulate the computational semantics of cubical type theory in an *equational* way, where types and terms are only ever considered up to judgmental equality. To achieve this, we adapted the recent perspective of Coquand [Coq19] in which types are interpreted as proof relevant families indexed in equivalence classes of typed terms rather than proof irrelevant relations on raw terms. In the equational setting, invariance under head expansion *vis-à-vis* well-typed reductions is therefore built into all definitions for free.

§6.5.2. Open computation and normal forms

✳ **(6.5.2∗1)** So far we have discussed the difficulties of computing in *cubical* contexts $\Psi = [i : \mathbb{I}, \ldots]$, all of which which are addressed directly by adopting the "objective metatheory" approach as shown by Sterling, Angiuli, and Gratzer [SAG19; SAG20]. While this form of computation is of philosophical interest because it expresses the sense in

which terms can be run as programs, it is not pertinent to the needs of users of type theory which are mostly centered around computation in general contexts Γ that contain not only dimensions but ordinary variables $x : A$. When programming in type theory, one mainly encounters computation during type checking and elaboration which of course must occur in all contexts; it is true that closed computation pertains to the results of program extraction, but extraction occurs only at the tail end of the programming process.

(6.5.2∗2) *Lack of neutral terms.* We will see that the objective metatheory does not immediately address a new difficulty found in computing with the open terms of cubical type theory. Our analysis of open computation in terms of synthetic normalization by evaluation (Chapter 5) relies on the presentation of type theory in terms of *neutral* and *normal* forms. A neutral form is to be thought of as a variable with various blocked stack frames stuck to it, and at types lacking an η-law the neutrals embed into the normals. The choice of *closed subtopos* for the gluing argument is determined by isolating a figure shape under whose morphisms the neutrals and normals carry a substitution action.

In ordinary type theory, an appropriate figure shape is the category $\mathcal{A}$ of contexts and structural renamings of variables; this choice is possible because neutrals and normals carry structural renaming actions, even though they do not *a priori* support general substitution. The situation for cubical type theory is more complex: the need for a representable interval forces us to include it in the category $\mathcal{A}_{\square}$ of "cubical atomic contexts", but this means that both normals and neutrals must be closed under not only renamings of variables but also under dimension substitutions $\langle 0/i \rangle$ and $\langle 1/i \rangle$ — in fact, in Cartesian cubical type theory one can even see that the diagonal renaming $\langle j/i \rangle$ is already problematic.

To understand why the neutrals of cubical type theory are not closed under $\langle 0/i \rangle$, consider a variable $x : \mathsf{path}(A,a)$; the path application to a generic dimension should be a neutral $x(i) : A(i)$. But the substitution $\langle 0/i \rangle^*(x(i))$ cannot be a neutral, because it must "compute" to the normal form of $a(0)$. Adding further annotations to the neutral does not help: for instance, one might decide to annotate the application with normal forms a_0, a_1 of $a(0), a(1)$ respectively; but even in this case, the substitution action would need to replace the neutral $x(i)$ with the normal a_0. The lack of a genuinely *cubical* notion of neutral form is the reason why normalization for cubical type theory has remained an open question within the community until the present intervention.

(6.5.2∗3) *The frontier of instability.* Prior attempts to understand the neutrals of cubical type theory have cast them as terms that are stuck *under some condition*; for instance, the path application $x(r)$ is stuck *so long as* $r \neq 0 \wedge r \neq 1$ [Ste20]. This characterization of neutrals is semantically a non-starter, however, because the notion of non-equality required here is the external set-theoretic one: in the language of the logos, $r \neq 0$ means that there is no atomic substitution that can identify r and 0: hence $r \neq 0 \wedge r \neq 1$ is always false. This suggests that "conditional stuckness" is not the correct way to think about neutrals for cubical type theory, because the conditions under which a neutral

remains stuck are not expressible in a cubically stable way.

An alternative perspective is to view neutrals in terms of "conditional non-stuckness": rather than specifying conditions under which a neutral remains stuck, we specify the conditions under which a neutral ceases to be stuck. In this case, the neutral $x(r)$ would cease to be stuck on the boundary $\partial r := (r = 0 \sqcup r = 1)$; we will refer to the condition under which a given neutral ceases to be stuck as its *frontier of instability*. Our negative point of view on neutrals has a significant advantage: the frontier of instability for any neutral of cubical type theory is expressible as a cofibration $\phi : \mathbb{F}$.

(6.5.2∗4) *Stabilizing neutral forms.* Let ans be an "answer type" with two constants yes,no : ans. The normal form presentation of ordinary type theory would include normal forms for the constants as well as an embedding of neutral forms of answer type into normal forms of answer type:

$$\mathbf{yes} : \{\mathsf{nf}(\mathsf{ans}) \mid \P \hookrightarrow \mathsf{yes}\}$$

$$\mathbf{no} : \{\mathsf{nf}(\mathsf{ans}) \mid \P \hookrightarrow \mathsf{no}\}$$

$$\mathbf{up}_{\mathsf{ans}} : \{\mathsf{ne}(\mathsf{ans}) \to \mathsf{nf}(\mathsf{ans}) \mid \P \hookrightarrow \lambda x.x\}$$

In cubical type theory, a neutral form that has become "unstuck" is one that needs to compute. Suppose that we have a variable $x : \mathsf{yes} \sim_{\mathsf{ans}} \mathsf{no}$ of path type; the neutral $x(0)$ is unstuck, so its frontier of instability is $\top$ — and unfortunately, we now have an exotic normal form $\mathbf{up}_{\mathsf{ans}}(x(0))$ which ought to be equal to $\mathbf{yes}$ but isn't. The solution to this problem proceeds in two steps.

First of all, we should recognize that a neutral that is unstable everywhere carries no useful computability data; if we write $\mathsf{ne}_\phi(A)$ for the collection of neutrals of type A with frontier of instability ϕ, we will ensure that $\mathsf{ne}_\phi(A)$ is purely syntactic under ϕ:

$$\mathsf{ne}_\phi(A) : \{\mathcal{U} \mid \bullet\phi \hookrightarrow \mathsf{tm}(A)\}$$

Then we will change the $\mathbf{up}_{\mathsf{ans}}$ constructor to require each neutral $a : \mathsf{ne}_\phi(A)$ to be *glued* with or *stabilized* by a compatible partial normal form defined on the frontier of instability:

$$\mathbf{up}_{\mathsf{ans}}[\phi] : \textstyle\prod_{a_0 : \mathsf{ne}_\phi(\mathsf{ans})}\prod_{a_\phi : \{\phi\}\,\{\mathsf{nf}(A)\mid\P\hookrightarrow a_0\}} \{\mathsf{nf}(\mathsf{ans}) \mid \bullet\phi \hookrightarrow [\P \hookrightarrow a_0, \phi \hookrightarrow a_\phi]\}$$

The modification above builds in enough information to the boundary between neutrals and normals to *compute* to the appropriate normal form just as soon as the neutral destabilizes. Because we will ensure that the cofibrations ϕ are only built from disjunctions, conjunctions and dimension equations, the component a_ϕ of the normal form can be represented essentially by left-inversion as a finitary tree branching on disjunctions and whose leaves are all normal forms under a conjunction of equational constraints.

◆ **(6.5.2∗5)** Under the stabilization discipline **(6.5.2∗4)**, the normal form for the path application $x(i)$ given a variable $x : \text{yes} \sim_{\text{ans}} \text{no}$ would therefore be represented like so:

$$\mathbf{up}_{\mathsf{ans}}[\partial i](x(i), [i = 0 \hookrightarrow \mathbf{yes}, i = 1 \hookrightarrow \mathbf{no}])$$

Under $i = 0$ then we have the following equations:

$$\mathbf{up}_{\mathsf{ans}}[\partial i](x(i), [i = 0 \hookrightarrow \mathbf{yes}, i = 1 \hookrightarrow \mathbf{no}])$$
$$= \mathbf{up}_{\mathsf{ans}}[\partial 0](x(0), [0 = 0 \hookrightarrow \mathbf{yes}, 0 = 1 \hookrightarrow \mathbf{no}])$$
$$= [0 = 0 \hookrightarrow \mathbf{yes}, 0 = 1 \hookrightarrow \mathbf{no}]$$
$$= \mathbf{yes}$$

※ **(6.5.2∗6)** It is unconventional (but not unheard of, see [Alt+01]) to impose equations on normal forms, since the purpose of normal forms is to *get rid of* equations. For more complicated type theories, such as cubical type theory or type theory with strict coproducts, a more refined perspective is applicable: a normal form presentation reduces a theory that is hard to decide to a theory that is trivial to decide. Because the finitary disjunctive theory of the interval is decidable, we have no compunction about imposing equations arising from this theory in the definition of normal forms.

CHAPTER 7

NORMALIZATION FOR CUBICAL TYPE THEORY

❦ **(7.0∗1)** The proof of normalization for univalent cubical type theory without universes is joint work with Carlo Angiuli [SA21]. This dissertation extends that result to a cumulative hierarchy of univalent universes.

⚓ **(7.0∗2)** We fix a topos $\mathbf{G}_\square$ equipped with an open $\P : \mathcal{O}_{\mathbf{G}_\square}$, as well as a transfinite and strict hierarchy of strong universes $\mathcal{U} \leq \mathcal{V}_\alpha < \mathcal{W}$ for $\alpha : \mathbf{L}$, defining $\mathcal{V}_{\mathsf{s}(\diamond)} := \mathcal{W}$. Not all of our constructions in this section will be internal to $\mathbf{Set}_{\mathbf{G}_\square}$ in the sense of being preserved under base changes between slices.

▪ **(7.0∗3)** As always, we assume a syntactic algebra for the Cartesian cubical type theory defined in Chapter 6, *i.e.* an algebra valued in $\mathcal{U}_\P$. We additionally assume that the syntactic interval object $\mathbb{I}$ is tiny in the sense of **(2.5∗1)**, and that $\curlywedge \leq \P$.

▪ **(7.0∗4)** We define the normalization structure $\mathbb{I}^\circledast$ of the interval to be $\mathbb{I}$ itself.

▪ **(7.0∗5)** We assume a type of variables for each syntactic type:

$$\mathsf{var} : \textstyle\prod_{\mathrm{A:tp}}\{\mathcal{U} \mid \P \hookrightarrow \mathrm{A}\}$$

§7.1. NORMALIZATION STRUCTURE OF COFIBRATIONS

※ **(7.1∗1)** It is true that in the syntax of cubical type theory, all cofibrations are built up from dimension equality, conjunction, disjunction, and universal quantification over the interval. The obvious computability structure for $\mathbb{F}$ does not have this restriction, however; because this property is important for establishing our decidability result, we will be careful to impose it in the corresponding normalization structure **(7.1∗3)**.

▪ **(7.1∗2)** *Computability structure of cofibrations.* We may define a computability structure $\mathbb{F}^\star : \{\mathcal{U} \mid \P \hookrightarrow \mathbb{F}\}$ by realigning the following subobject of Ω:

$$\mathbb{F}^\star :\cong \{\phi : \Omega \mid \exists \psi : \mathbb{F}.\bigcirc([\psi] =_\Omega \phi)\}$$

The alignment above is possible because the element of $\mathbb{F}$ quantified by $\phi \in \mathbb{F}^\star$ is uniquely determined by propositional univalence of $\mathbb{F}$.

▪ **(7.1∗3)** *Normalization structure of cofibrations.* We define $\mathbb{F}^\circledast$ to be the smallest subobject of $\mathbb{F}^\star$ closed under the following rules:

$$\phi : \mathbb{F}^\star \mid \P \vdash \phi \in \mathbb{F}^\circledast$$
$$\phi,\psi : \mathbb{F}^\star \mid \phi \in \mathbb{F}^\circledast \wedge \psi \in \mathbb{F}^\circledast \vdash (\phi \wedge \psi) \in \mathbb{F}^\circledast$$
$$\phi,\psi : \mathbb{F}^\star \mid \phi \in \mathbb{F}^\circledast \wedge \psi \in \mathbb{F}^\circledast \vdash ((\phi \sqcup \psi) \wedge \bullet(\phi \vee \psi)) \in \mathbb{F}^\circledast$$
$$\phi : \mathbb{I} \to \mathbb{F}^\star \mid \forall i : \mathbb{I}.(\phi(i) \in \mathbb{F}^\circledast) \vdash (\forall i : \mathbb{I}.\phi(i)) \in \mathbb{F}^\circledast$$
$$r,s : \mathbb{I} \mid \top \vdash (r = s) \in \mathbb{F}^\circledast$$

The first rule above ensures that underneath the open modality, $\mathbb{F}^\circledast$ is all of $\mathbb{F}^\star$; hence we retain the alignment $\mathbb{F}^\circledast : \{\mathcal{U} \mid \P \hookrightarrow \mathbb{F}\}$.

◼ **(7.1∗4)** *Closure under connectives.* The normalization structure $\mathbb{F}^{\circledast}$ is closed under the following connectives:

$$(=^{\circledast}) : \{\mathbb{I} \times \mathbb{I} \to \mathbb{F}^{\circledast} \mid \P \hookrightarrow (=)\}$$
$$(r =^{\circledast} s) := (r = s)$$

$$(\sqcap^{\circledast}) : \{\mathbb{F}^{\circledast} \times \mathbb{F}^{\circledast} \to \mathbb{F}^{\circledast} \mid \P \hookrightarrow (\sqcap)\}$$
$$(\phi \sqcap^{\circledast} \psi) = \phi \wedge \psi$$

$$(\sqcup^{\circledast}) : \{\mathbb{F}^{\circledast} \times \mathbb{F}^{\circledast} \to \mathbb{F}^{\circledast} \mid \P \hookrightarrow (\sqcup)\}$$
$$(\phi \sqcup^{\circledast} \psi) = (\phi \sqcup \psi) \wedge \bullet(\phi \vee \psi)$$

$$(\forall^{\circledast}) : \{(\mathbb{I} \to \mathbb{F}^{\circledast}) \to \mathbb{F}^{\circledast} \mid \P \hookrightarrow (\forall)\}$$
$$\forall^{\circledast}(\phi) = \forall i : \mathbb{I}.\phi(i)$$

We provide the definitions above to indicate the interpretations of each cofibration connective in the normalization algebra, but in practice we will simply write $r = s$ and $\phi \wedge \psi$ and $\forall i.\phi(i)$; on the other hand, we will write $\phi \sqcup^{\circledast} \psi$ for the disjunction rather than expanding it to its more complex definition.

◼ **(7.1∗5)** If E is any type in the internal language of $\mathbf{Set}_{\mathbf{G}_{\square}}$, then E is $\mathbb{F}^{\circledast}$-local if and only if $\bigcirc$E is $\mathbb{F}$-local. In other words, locality can fail only at the syntactical level.

Proof. It is obvious that E being $\mathbb{F}^{\circledast}$-local is sufficient for $\bigcirc$E to be $\mathbb{F}$-local; we will check that this condition is also necessary. Assume that $\bigcirc$E is $\mathbb{F}$-local; first we must show that E is $\curlywedge$-connected, but this follows from $\mathbb{F}$-locality of $\bigcirc$E combined with our assumption **(7.0∗3)** that $\curlywedge \leq \P$. Next we fix $\phi, \psi : \mathbb{F}^{\circledast}$ to check that the collection of partial elements $\{\phi \sqcup^{\circledast} \psi\}$ E is isomorphic to the collection of matching families for ϕ, ψ in E. It suffices to check this condition on both $\bigcirc$E and $\bullet$E; the former we have by assumption, and the latter we have because $\phi \sqcup^{\circledast} \psi \leq \bullet(\phi \vee \psi)$. $\square$

◼ **(7.1∗6)** *Syntactic stabilization.* Let $\phi : \Omega$ be a proposition; fix $X_{\circ} : \mathcal{U}_{\P}$ and $X : \{\mathcal{U} \mid \bullet\phi \hookrightarrow X_{\circ}\}$ and $Y : \{\phi\}\{\mathcal{U} \mid \P \hookrightarrow X_{\circ}\}$. We define the *syntactic stabilization* of X by Y along ϕ as follows:

> **record** $X \wr_{\phi} Y : \{\mathcal{U} \mid \bullet\phi \hookrightarrow [\P \hookrightarrow X, \phi \hookrightarrow Y]\}$ **where**
> **constructor** $[- \mid \phi \hookrightarrow -]$
> base : X
> part : $\{\phi\}\{Y \mid \P \hookrightarrow \text{base}\}$

It is worth checking the alignments above:

$$\P \vdash X \wr_{\phi} Y \cong \textstyle\sum_{b:X}\{\phi\}\{Y \mid \top \hookrightarrow b\} \cong X$$

$$\phi \vdash X \wr_\phi Y \cong X_\circ \wr_\top Y \cong \sum_{b:X_\circ}\{Y \mid \P \hookrightarrow b\} \cong \sum_{b:\bigcirc Y}\{Y \mid \P \hookrightarrow b\} \cong Y$$

The intuition for this type is that X is a type that destabilizes to its syntactic part under ϕ; meanwhile, Y is a type with the same syntactic part that may *not* destabilize. Then the syntactic stabilization of X by Y along ϕ consists of *glued* elements $[x \mid \phi \hookrightarrow y]$ that restrict on $\bullet\phi$ to the partial element $[\P \hookrightarrow x, \phi \hookrightarrow y]$.

§7.2. NORMAL AND NEUTRAL FORMS

■ **(7.2∗1)** The collections of normal and neutral forms for cubical type theory are similar to those of Chapter 5, except that we must account for frontiers of instability.

$$\mathsf{nftp} : \{\mathcal{U} \mid \P \hookrightarrow \mathsf{tp}\}$$
$$\mathsf{var},\mathsf{nf} : \textstyle\prod_{A:\mathsf{tp}}\{\mathcal{U} \mid \P \hookrightarrow A\}$$
$$\mathsf{ne}_\bullet : \textstyle\prod_{\phi:\mathbb{F}*}\prod_{A:\mathsf{tp}}\{\mathcal{U} \mid \bullet\phi \hookrightarrow A\}$$

We have the following derived collections:

$$\mathsf{nftp}_\diamond = \mathsf{nftp}$$
$$\mathsf{nftp}_n = \mathsf{nf}(\langle\uparrow^\diamond_{\mathsf{s}(n)}\rangle U_n)$$
$$\mathsf{netp}^\phi_n = \mathsf{ne}_\phi(\langle\uparrow^\diamond_{\mathsf{s}(n)}\rangle U_n)$$

◆ **(7.2∗2)** *Stabilized neutrals.* We may stabilize the neutrals against the normals via $\mathsf{ne}_\phi(A) \wr_\phi \mathsf{nf}(A)$. Later on, we will stabilize neutrals against different computability structures to generalize Tait's *reflect* operation to the cubical setting.

⚑ **(7.2∗3)** *Reflection and reification operations.* We define some notations for expressing the closure of a computability structure under Tait's yoga §5.3. Let $E : \{\mathcal{W} \mid \P \hookrightarrow B\}$ be a type in some universe that is aligned over the terms of type B; we may express the closure of E under a *stabilized* Tait reflection operation and an ordinary Tait reification operation as follows:

$$\mathsf{Reflect}[\phi](B,E) := \{\mathsf{ne}(B) \wr_\phi E \to E \mid \bullet\phi \hookrightarrow \lambda x.x\}$$
$$\mathsf{Reify}(B,E) := \{E \to \mathsf{nf}(B) \mid \P \hookrightarrow \lambda x.x\}$$

⚑ **(7.2∗4)** *Homogeneous composition operation.* Let $A : \mathsf{tp}_\alpha$ be a syntactic type and let $E : \{\mathcal{W} \mid \P \hookrightarrow \langle\uparrow^\diamond_\alpha\rangle A\}$ be a type in some universe aligned over elements of A. A well-aligned homogeneous composition structure for (E,A) is an element of the following $\bigcirc$-connected and hence $\bullet$-modal type:

$$\mathsf{HCom}^\star(A,E) := \textstyle\prod_{r,s:\mathbb{I}}\prod_{\phi:\mathbb{F}*}\prod_{a:\prod_{i:\mathbb{I}}\{(i=r)\sqcup^*\phi\}E}$$
$$\{E \mid \bullet((i=r)\sqcup^*\phi) \hookrightarrow [\P \hookrightarrow \mathsf{hcom}^{r\leadsto s;\phi}_{\langle\uparrow^\diamond_\alpha\rangle A}a, (i=r)\sqcup^*\phi \hookrightarrow a(s)]\}$$

(7.2∗5) *Variables.* We add an inductive clause to treat every variable as a totally stable neutral:

$$\mathsf{var} : \{A\}\ \{\mathsf{var}(A) \to \mathsf{ne}_\perp(A) \mid \P \hookrightarrow \lambda x.x\}$$

(7.2∗6) *Universe structure.* The normal forms of universes and their lifting coercions are more complex in the cubical setting:

$$\mathsf{U}_n^\alpha : \{\mathsf{nftp}_\alpha \mid \P \hookrightarrow \langle \uparrow_{n+1}^\alpha \rangle \mathsf{U}_n\} \qquad\qquad\qquad\text{for } n < \alpha$$

$$\mathsf{up}_n^\alpha[\phi] : \textstyle\prod_{A:\mathsf{netp}_n^\phi} \prod_{A_\phi:\{\phi\}} {}_{\{\mathsf{nftp}_\alpha \mid \bullet\phi \hookrightarrow \langle\uparrow_n^\alpha\rangle A\}}\{\mathsf{nftp}_\alpha \mid \bullet\phi \hookrightarrow [\P \hookrightarrow \langle\uparrow_n^\alpha\rangle A, \phi \hookrightarrow A_\phi]\} \quad \text{for } n < \alpha$$

$$\widetilde{\mathsf{up}}_n[\phi,\psi] : \textstyle\prod_{A:\mathsf{netp}_n^\phi} \{\mathsf{ne}_\psi(\langle\uparrow_n^\diamond\rangle A) \wr_{\phi\wedge\psi} \mathsf{nf}(\langle\uparrow_n^\diamond\rangle A) \to \mathsf{nf}(\langle\uparrow_n^\diamond\rangle A) \mid \bullet(\phi\wedge\psi) \hookrightarrow \lambda x.x\}$$

(7.2∗7) *Stuck Kan operations.* A new source of normal forms in cubical type theory is the coercion and homogeneous composition operations on neutral types:

$$\frac{\phi : \mathbb{F}^* \qquad A : \mathsf{netp}_n^\phi \qquad r,s : \mathbb{I} \qquad \psi : \mathbb{F}^* \qquad a : \prod_{i:\mathbb{I}}\{r = s \vee \psi\}\, \mathsf{nf}(\langle\uparrow_n^\diamond\rangle A) \\ h : \{\phi\}\,\{\mathsf{nf}(\langle\uparrow_n^\diamond\rangle A) \mid \bullet(r = s \vee \psi) \hookrightarrow \P \hookrightarrow \mathsf{hcom}_{\langle\uparrow_n^\diamond\rangle A}^{r\rightsquigarrow s;a}, r = s \vee \psi \hookrightarrow a(s)\}}{\mathsf{hcom}_n[\phi](A,r,s,\psi,a,h) : \{\mathsf{nf}(\langle\uparrow_n^\diamond\rangle A) \mid \bullet(\phi \vee r = s \vee \psi) \hookrightarrow \P \hookrightarrow \mathsf{hcom}_{\langle\uparrow_n^\diamond\rangle A}^{r\rightsquigarrow s;\psi} a, \phi \hookrightarrow h, r = s \vee \psi \hookrightarrow a(s)\}}$$

$$\frac{\phi : \mathbb{I} \to \mathbb{F}^* \qquad A : \prod_{i:\mathbb{I}}\mathsf{netp}_n^{\phi(i)} \qquad r,s : \mathbb{I} \qquad a : \mathsf{nf}(\langle\uparrow_n^\diamond\rangle A(r)) \\ c : \{\forall i.\phi(i)\}\,\{\mathsf{nf}(\langle\uparrow_n^\diamond\rangle A(s)) \mid \bullet(r = s) \hookrightarrow [\P \hookrightarrow \mathsf{coe}_{\langle\uparrow_n^\diamond\rangle\circ A}^{r\rightsquigarrow s} a, r = s \hookrightarrow a]\}}{\mathsf{coe}_n[\phi](A,r,s,a,c) : \{\mathsf{nf}(\langle\uparrow_n^\diamond\rangle A(s)) \mid \bullet((\forall i.\phi(i)) \vee r = s) \hookrightarrow \P \hookrightarrow \mathsf{coe}_{\langle\uparrow_n^\diamond\rangle A}^{r\rightsquigarrow s} a, \forall i.\phi(i) \hookrightarrow c, r = s \hookrightarrow a\}}$$

(7.2∗8) *Standard connectives.* The normal forms for the standard connectives do not change except to preserve frontiers of instability.

$$\mathsf{sg}_\alpha : \{(\textstyle\sum_{A:\mathsf{nftp}_\alpha}(\mathsf{var}(\langle\uparrow_\alpha^\diamond\rangle A) \to \mathsf{nftp}_\alpha)) \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \Sigma_\alpha\}$$

$$\mathsf{split} : \{A,B,\phi\}\,\{\mathsf{ne}_\phi(\Sigma(A,B)) \to \textstyle\sum_{x:\mathsf{ne}_\phi(A)}\mathsf{ne}_\phi(B(x)) \mid \bullet\phi \hookrightarrow \lambda p.(p.1,p.2)\}$$

$$\mathsf{pair} : \{A,B\}\,\{(\textstyle\sum_{x:\mathsf{nf}(A)}\mathsf{nf}(B(x))) \to \mathsf{nf}(\Sigma(A,B)) \mid \P \hookrightarrow \lambda p.(p.1,p.2)\}$$

$$\mathsf{pi}_\alpha : \{(\textstyle\sum_{A:\mathsf{nftp}_\alpha}(\mathsf{var}(\langle\uparrow_\alpha^\diamond\rangle A) \to \mathsf{nftp}_\alpha)) \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \Pi_\alpha\}$$

$$\mathsf{app} : \{A,B,\phi\}\,\{\mathsf{ne}_\phi(\Pi(A,B)) \to \textstyle\prod_{x:\mathsf{nf}(A)}\mathsf{ne}_\phi(B(x)) \mid \bullet\phi \hookrightarrow \lambda f.\lambda x.f(x)\}$$

$$\mathsf{lam} : \{A,B\}\,\{(\textstyle\prod_{x:\mathsf{var}(A)}\mathsf{nf}(B(x))) \to \mathsf{nf}(\Pi(A,B)) \mid \P \hookrightarrow \lambda f.\lambda x.f(x)\}$$

(7.2∗9) *The path type.* The path type is one location where the frontier of instability on a neutral changes.

$$\mathsf{path}_\alpha : \{(\textstyle\sum_{A:\mathbb{I}\to\mathsf{nftp}\alpha}\prod_{i:\mathbb{I}}\{\partial^* i\}\,\mathsf{nf}(\langle\uparrow_\alpha^\diamond\rangle A(i))) \to \mathsf{nftp}_\alpha \mid \P \hookrightarrow \mathsf{path}_\alpha\}$$

$$\mathsf{papp} : \{A,a,\phi\}\,\{\mathsf{ne}_\phi(\mathsf{path}_\diamond(A,a)) \to \textstyle\prod_{i:\mathbb{I}}\mathsf{ne}_{\phi\sqcup*\partial*i}(A(i)) \mid \P \hookrightarrow \lambda p.\lambda i.p(i)\}$$

$$\mathsf{plam} : \{A\}\,\{\textstyle\prod_{a:\prod_{i:\mathbb{I}}\mathsf{nf}(A(i))}\mathsf{nf}(\mathsf{path}_\diamond(A,\lambda i.a(i))) \mid \P \hookrightarrow \lambda p.\lambda i.p(i)\}$$

(7.2∗10) *The glue type.*

$\mathbf{glue}_\alpha : \prod_{\phi:\mathbb{F}^*} \prod_{B:\mathsf{nftp}_\alpha} \prod_{A:\{\phi\}\,\mathsf{nftp}_\alpha} \prod_{f:\{\phi\}\,\mathsf{nf}(\mathsf{Equiv}(\langle\uparrow^\diamond_\alpha\rangle A,\langle\uparrow^\diamond_\alpha\rangle B))} \{\mathsf{nftp}_\alpha \mid \bullet\phi \hookrightarrow [\phi \hookrightarrow A, \P \hookrightarrow \mathsf{glue}_\alpha \ [\phi \mid B \hookrightarrow (A,f)]]\}$

$\mathbf{englue} : \{D\}\ \{\mathsf{ne}_\psi(\mathsf{glue}_\diamond(D)) \to \mathsf{ne}_{\psi\sqcup *D.\phi}(D.\mathsf{base}) \mid \P \hookrightarrow \mathsf{unglue}\}$

$\mathbf{englue} : \{D\}\ \prod_{a:\{D.\phi\}\,\mathsf{nf}(D.A)} \prod_{b:\{\mathsf{nf}(D.B)\mid D.\mathsf{phi}\hookrightarrow a\}} \{\mathsf{nf}(\mathsf{glue}_\diamond(D)) \mid \bullet\phi \hookrightarrow [\phi \hookrightarrow a, \P \hookrightarrow \mathsf{englue}\ [b \mid \phi \hookrightarrow a]]\}$

▪ **(7.2∗11)** *The booleans.* The normal forms for the booleans are similar to those of **(5.3∗10)**, except we must stabilize the neutrals before they are lifted into the normals; we also add formal composite terms.

$$\mathbf{bool}_\alpha : \{\mathsf{nftp}_\alpha \mid \P \hookrightarrow \mathsf{bool}_\alpha\}$$
$$\mathbf{tt} : \{\mathsf{nf}(\mathsf{bool}) \mid \P \hookrightarrow \mathsf{tt}\}$$
$$\mathbf{ff} : \{\mathsf{nf}(\mathsf{bool}) \mid \P \hookrightarrow \mathsf{ff}\}$$
$$\mathbf{up}_{\mathsf{bool}}[\phi] : \mathsf{Reflect}[\phi](\mathsf{bool},\mathsf{nf}(\mathsf{bool}))$$
$$\mathbf{hcom}_{\mathsf{bool}} : \mathsf{HCom}^\star(\mathsf{bool},\mathsf{nf}(\mathsf{bool}))$$
$$\mathbf{ind}_{\mathsf{bool}}[\phi] : \left\{ \prod_{C:\mathsf{var}(\mathsf{bool})\to\mathsf{nftp}} \prod_{c_0:\mathsf{nf}(C(\mathsf{tt}))} \prod_{c_1:\mathsf{nf}(C(\mathsf{ff}))} \prod_{x:\mathsf{ne}_\phi(\mathsf{bool})} \mathsf{ne}_\phi(C(x)) \mid \P \hookrightarrow \mathsf{ind}_{\mathsf{bool}} \right\}$$

▪ **(7.2∗12)** *The circle.*

$$\mathbf{S}^1_\alpha : \{\mathsf{nftp}_\alpha \mid \P \hookrightarrow \mathsf{S}^1_\alpha\}$$
$$\mathbf{base} : \{\mathsf{nf}(\mathsf{S}^1) \mid \P \hookrightarrow \mathsf{base}\}$$
$$\mathbf{loop} : \prod_{i:\mathbb{I}} \{\mathsf{nf}(\mathsf{S}^1) \mid \bullet\partial^* i \hookrightarrow [\P \hookrightarrow \mathsf{loop}(i), \partial^* i \hookrightarrow \mathsf{base}]\}$$
$$\mathbf{up}_{\mathsf{S}^1}[\phi] : \mathsf{Reflect}[\phi](\mathsf{S}^1,\mathsf{nf}(\mathsf{S}^1))$$
$$\mathbf{hcom}_{\mathsf{S}^1} : \mathsf{HCom}^\star(\mathsf{S}^1,\mathsf{nf}(\mathsf{S}^1))$$
$$\mathbf{ind}_{\mathsf{S}^1}[\phi] : \left\{ \prod_{C:\mathsf{var}(\mathsf{S}^1)\to\mathsf{nftp}} \prod_{c_b:\mathsf{nf}(C(\mathsf{base}))} \prod_{c_l:\prod_{i:\mathbb{I}}\{C(\mathsf{loop}(i))\mid\partial i\hookrightarrow c_b\}} \prod_{x:\mathsf{ne}_\phi(\mathsf{S}^1)} \mathsf{ne}_\phi(C(x)) \mid \P \hookrightarrow \mathsf{ind}_{\mathsf{S}^1} \right\}$$

§7.2.1. Explicit construction

※ **(7.2.1∗1)** The collections of normal and neutral forms can be constructed as in §5.4.1 by a mutual indexed inductive definition. We will define the following families:

$$[a \in^\phi_{\mathsf{ne}} A] : \mathcal{U}_{\backslash\bullet\phi} \qquad (\phi : \mathbb{F}^*, A : \mathsf{tp}, a : A)$$
$$[A \ni_{\mathsf{nf}} a] : \mathcal{U}_{\backslash\P} \qquad (A : \mathsf{tp}, a : A)$$
$$[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A] : \mathcal{U}_{\backslash\P} \qquad (A : \mathsf{tp})$$

We define $[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} A]$ to be $[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A]$ when $\alpha = \diamond$ and $[\mathcal{U}_n \ni_{\mathsf{nf}} A]$ when $\alpha = n$. To define an indexed inductive family of ψ-connected types is the same as to define an indexed *quotient*-inductive family of ordinary types, by adding the additional following clauses that collapse the fiber to a point when restricted over ψ:

$$\frac{A:\mathsf{tp} \qquad z:\P}{\mathsf{pt}(A,z):[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A]} \qquad \frac{A:\mathsf{tp} \qquad \widetilde{A}:[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A] \qquad z:\P}{\widetilde{A} =_{[\mathsf{tp}_\diamond \ni_{\mathsf{nf}} A]} \mathsf{pt}(A,z)} \qquad \frac{A:\mathsf{tp}_\alpha \qquad a:A \qquad z:\P}{\mathsf{pt}(A,a,z):[A \ni_{\mathsf{nf}} a]}$$

$$\frac{A:\mathsf{tp}_\alpha \qquad a:A \qquad \widetilde{a}:[A \ni_{\mathsf{nf}} a] \qquad z:\P}{\widetilde{a} =_{[A \ni_{\mathsf{nf}} a]} \mathsf{pt}(A,a,z)} \qquad \frac{\phi:\mathbb{F}^* \qquad A:\mathsf{tp}_\alpha \qquad a:A \qquad z:\bullet\phi}{\mathsf{pt}(A,a,z):[a \in_{\mathsf{ne}}^\phi A]}$$

$$\frac{\phi:\mathbb{F}^* \qquad A:\mathsf{tp}_\alpha \qquad a:A \qquad \widetilde{a}:[a \in_{\mathsf{ne}}^\phi A] \qquad z:\bullet\phi}{\widetilde{a} =_{[a \in_{\mathsf{ne}}^\phi A]} \mathsf{pt}(A,a,z)}$$

In this section, we will give as examples a few representative clauses of this inductive definition.

◆ **(7.2.1∗2)** *Basic judgmental structure.* We add inductive clauses to substantiate the basic judgmental structure of cubical type theory with universes:

$$\frac{(n < \alpha)}{U_n^\alpha : [\mathsf{tp}_\alpha \ni_{\mathsf{nf}} \langle\uparrow_{n+1}^\alpha\rangle U_n]} \qquad \frac{\phi:\mathbb{F}^* \qquad A:\mathsf{tp}_n \qquad (n \le \alpha)}{\widetilde{A}:[A \in_{\mathsf{ne}}^\phi \langle\uparrow_{n+1}^\diamond\rangle U_n] \qquad \widetilde{A}_\phi:\{\phi\}\,[\mathsf{tp}_n \ni_{\mathsf{nf}} A]}{\mathsf{up}_n^\alpha\{A\}(\widetilde{A},\widetilde{A}_\phi) : [\mathsf{tp}_\alpha \ni_{\mathsf{nf}} \langle\uparrow_n^\alpha\rangle A]}$$

$$\frac{\phi,\psi:\mathbb{F}^* \qquad A:\mathsf{tp}_n \qquad a:\langle\uparrow_n^\diamond\rangle A}{\widetilde{A}:[A \in_{\mathsf{ne}}^\phi \langle\uparrow_{n+1}^\diamond\rangle U_n] \qquad \widetilde{a}:[a \in_{\mathsf{ne}}^\psi \langle\uparrow_n^\diamond\rangle A]}{\widetilde{\mathsf{up}}_n\{A,a\}(\widetilde{A},\widetilde{a}) : [\langle\uparrow_n^\diamond\rangle A \ni_{\mathsf{nf}} a]} \qquad \frac{A:\mathsf{tp}_n \qquad \widetilde{a}:\mathsf{var}(A)}{\mathsf{var}\{A\}(\widetilde{a}):[\widetilde{a} \in_{\mathsf{ne}}^\perp A]}$$

After the inductive definitions are completed, we may take their total spaces to define the collections of neutral and normal forms of terms and types by realignment:

$$\mathsf{nftp}_\alpha : \{\mathcal{U} \mid \P \hookrightarrow \mathsf{tp}_\alpha\} \qquad\qquad \mathsf{nf} : \textstyle\prod_{A:\mathsf{tp}}\{\mathcal{U} \mid \P \hookrightarrow \mathsf{tm}_\alpha(A)\}$$

$$\mathsf{nftp}_\alpha \cong \textstyle\sum_{A:\mathsf{tp}_\alpha}[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} A] \qquad \mathsf{nf}(A) \cong \textstyle\sum_{a:A}[A \ni_{\mathsf{nf}} a]$$

$$\mathsf{ne}_\phi : \textstyle\prod_{A:\mathsf{tp}}\{\mathcal{U} \mid \bullet\phi \hookrightarrow A\}$$

$$\mathsf{ne}_\phi(A) \cong \textstyle\sum_{a:A}[a \in_{\mathsf{ne}}^\phi A]$$

The realignments above are possible because each fiber $[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} A], [A \ni_{\mathsf{nf}} a], [a \in_{\mathsf{ne}}^\phi A]$ was valued in an appropriate closed subuniverse $\mathcal{U}_{\backslash\P}$ or $\mathcal{U}_{\backslash\bullet\phi}$.

▪ **(7.2.1∗3)** Each subtype $\{\mathsf{nftp}_\alpha \mid \P \hookrightarrow A\}$ is isomorphic to $[\mathsf{tp}_\alpha \ni_{\mathsf{nf}} A]$. Likewise, $\{\mathsf{nf}(A) \mid \P \hookrightarrow a\}$ is isomorphic to $[A \ni_{\mathsf{nf}} a]$ and $\{\mathsf{ne}_\phi(A) \mid \P \hookrightarrow a\}$ is isomorphic to $[a \in_{\mathsf{ne}}^\phi A]$.

◆ **(7.2.1∗4)** As an example, we show how to use the isomorphisms **(7.2.1∗3)** explicitly to define the following normal form constructor:

$$\mathsf{up}_n^\alpha[\phi] : \textstyle\prod_{A:\mathsf{netp}_n^\phi}\prod_{A_\phi:\{\phi\}}\{\mathsf{nftp}_\alpha|\P\hookrightarrow\langle\uparrow_n^\alpha\rangle A\}\{\mathsf{nftp}_\alpha \mid \bullet\phi \hookrightarrow [\P \hookrightarrow \langle\uparrow_n^\alpha\rangle A, \phi \hookrightarrow A_\phi]\}$$

Fixing A and A_ϕ, we note by definition of netp_n^ϕ, nftp_α that we have $\widetilde{A} : [A \in_{\mathsf{ne}}^\phi \langle \uparrow_{n+1}^\diamond \rangle U_n]$ and $\widetilde{A}_\phi : \{\phi\}\, [\langle \uparrow_n^\diamond \rangle A \ni_{\mathsf{nf}} A]$. Hence we choose the pair $(A, \mathsf{up}_n^\alpha \{A\}(\widetilde{A}, \widetilde{A}_\phi))$.

§7.2.2. Injectivity of normal form constructors

※ **(7.2.2∗1)** We will use roughly the same technique as **(5.4.1∗7)** to prove a modal injectivity result for the dependent product normal form constructor. The argument is more intricate, however, due the presence of non-trivial boundaries on cubical normal forms.

■ **(7.2.2∗2)** We begin by defining a modal predicate $\mathsf{isPi} : \mathsf{nftp} \to \Omega_{\backslash \P}$ satisfying the following universal property:

$$\forall X : \mathsf{nftp}.\mathsf{isPi}(X) \Leftrightarrow \bullet \exists F.X = \mathbf{pi}_\diamond(F) \qquad (7.2.2\ast2\ast1)$$

We proceed by induction on the normal types:

$$\mathsf{isPi}(\mathbf{U}_n^\diamond) = \bullet\bot \qquad \mathsf{isPi}(\mathbf{up}_n^\diamond[\phi](A,A_\phi)) = \bullet\exists\_ : \phi.\mathsf{isPi}(A_\phi) \qquad \mathsf{isPi}(\mathbf{sg}_\diamond(A,B)) = \bullet\bot$$

$$\mathsf{isPi}(\mathbf{pi}_\diamond(A,B)) = \top \qquad \mathsf{isPi}(\mathbf{path}_\diamond(A,a)) = \bullet\bot$$

$$\mathsf{isPi}(\mathbf{glue}_\diamond(\phi,B,A,f)) = \bullet\exists\_ : \phi.\mathsf{isPi}(A) \qquad \mathsf{isPi}(\mathbf{bool}_\diamond) = \bullet\bot \qquad \mathsf{isPi}(\mathbf{S}_\diamond^1) = \bullet\bot$$

The universal property Eq. (7.2.2∗2∗1) follows again by induction.

Proof. Let X be a normal type such that $\mathsf{isPi}(X)$ holds; we must check that $\bullet\exists F.X = \mathbf{pi}_\diamond(F)$; we proceed by induction on X. The induction cases can be organized into three representave categories.

1) *Total case.* If $X = \mathbf{pi}_\diamond(F)$, then we choose F itself.
2) *Partial case.* If $X = \mathbf{up}_n^\diamond[\phi](A,A_\phi)$, our assumption $\mathsf{isPi}(X)$ computes to $\bullet\exists\_ : \phi.\mathsf{isPi}(A_\phi)$. Because our goal is $\bullet$-modal we may assume $\phi = \top$ and $\mathsf{isPi}(A_\phi)$, so by induction we have $\bullet\exists F.A_\phi = \mathbf{pi}_\diamond(F)$. Because $\phi = \top$ we also have $\mathbf{up}_n^\phi[\top](A,A_\phi) = A_\phi$, hence $\bullet(X = \mathbf{pi}_\diamond(F))$.
3) *Empty case.* If $X = \mathbf{U}_n^\diamond$, our assumption $\mathsf{isPi}(X)$ computes to $\bullet\bot$, which implies any $\bullet$-modal proposition.

Conversely, assume $\bullet\exists F.X = \mathbf{pi}_\diamond(F)$. Because $\mathsf{isPi}(X)$ is $\bullet$-modal, we may unconditionally assume some family F such that $X = \mathbf{pi}_\diamond(F)$. Then we have $\mathsf{isPi}(X) = \top$. □

■ **(7.2.2∗3)** *Modal injectivity.* The normal form constructor $\mathbf{pi}$ is semantically injective in the sense that the following holds:

$$\forall F,F' : \mathsf{nffam.}\ (\mathbf{pi}_\diamond(F) = \mathbf{pi}_\diamond(F')) \implies \bullet(F = F')$$

Proof. We will use the same argument as **(5.4.1∗7)**, adjusted for the more complex boundaries that appear in the cubical setting. We define a function to "invert" the **pi**$_\diamond$ constructor within the extent of isPi:

$$\mathsf{unPi} : \{X : \mathsf{nftp} \mid \mathsf{isPi}(X)\} \to \bullet\mathsf{nffam}$$
$$\mathsf{unPi}(\mathbf{pi}_\diamond(A,B)) = \eta_{\backslash\P}(A,B)$$
$$\mathsf{unPi}(\mathbf{up}_n^\diamond[\phi](A,A_\phi)) = \mathsf{unPi}(A_\phi)$$
$$\mathsf{unPi}(\mathbf{glue}_\diamond(\phi,B,A,f)) = \mathsf{unPi}(A)$$

It is worth carefully justifying the partial cases for $\mathbf{up}_n^\diamond[\phi]$ and $\mathbf{glue}_\diamond(\phi,B,A,F)$; in the latter (representative) case, we have $\bullet\exists\_ : \phi.\mathsf{isPi}(A)$ by assumption, but because we are constructing an element of the $\bullet$-modal type $\bullet\mathsf{nffam}$, we may assume ϕ and $\mathsf{isPi}(A)$; hence it is possible to make the recursive call $\mathsf{unPi}(A)$. $\qquad\square$

§7.3. TYPES AND UNIVERSES

§7.3.1. Computability structure of types

※ **(7.3.1∗1)** Recall that we have assumed in **(7.0∗3)** that the interval is tiny **(2.5∗1)**, *i.e.* we have an adjunction $(-)^{\mathbb{I}} \dashv (-)_{\mathbb{I}}$ in $\mathbf{Set}_{\mathbf{G}_\square}$. While this adjunction does descend to each slice of $\mathbf{Set}_{\mathbf{G}_\square}$, it is not preserved by base change; therefore, one must be cautious when using it. Thankfully, we will only have need of it at the borders of our construction.

◼ **(7.3.1∗2)** The "root" functor $(-)_{\mathbb{I}} : \mathbf{Set}_{\mathbf{G}_\square} \longrightarrow \mathbf{Set}_{\mathbf{G}_\square}$ preserves $\bullet$-modal objects.

Proof. Let $E : \mathbf{Set}_{\mathbf{G}_\square}$ be a $\bullet$-modal object; equivalently, E is $\bigcirc$-connected in the sense that $\{\P\} E \cong \mathbf{1}$ or equivalently $\{\P\} (E \cong \mathbf{1})$. We will check that $E_{\mathbb{I}}$ is $\bigcirc$-connected by a computation, fixing an arbitrary sheaf $F : \mathbf{Set}_{\mathbf{G}_\square}$:

$$\mathrm{Hom}_{\mathbf{Set}_{\mathbf{G}_\square}}(F,\{\P\} E_{\mathbb{I}})$$
$$\cong \mathrm{Hom}_{\mathbf{Set}_{\mathbf{G}_\square}}(F \times \P, E_{\mathbb{I}})$$
$$\cong \mathrm{Hom}_{\mathbf{Set}_{\mathbf{G}_\square}}((F \times \P)^{\mathbb{I}}, E)$$

Evaluating with either endpoint of the interval and then projecting, we have a morphism $(F \times \P)^{\mathbb{I}} \to \P$; hence $\mathrm{Hom}_{\mathbf{Set}_{\mathbf{G}_\square}}((F \times \P)^{\mathbb{I}}, E)$ is isomorphic to $\mathrm{Hom}_{\mathbf{Set}_{\mathbf{G}_\square}}((F \times \P)^{\mathbb{I}}, \mathbf{1})$.

$$\ldots \quad \cong \mathrm{Hom}_{\mathbf{Set}_{\mathbf{G}_\square}}((F \times \P)^{\mathbb{I}}, \mathbf{1})$$
$$\cong \mathrm{Hom}_{\mathbf{Set}_{\mathbf{G}_\square}}(F \times \P, \mathbf{1})$$
$$\cong \mathrm{Hom}_{\mathbf{Set}_{\mathbf{G}_\square}}(F, \mathbf{1})$$

Thus $E_{\mathbb{I}}$ is $\bigcirc$-connected or $\bullet$-modal. $\qquad\square$

(7.3.1∗3) *Pretype computability structure.* We recall the computability structure of types from **(4.4∗4)**:

$$
\textbf{record } \mathsf{tp}_\alpha^\star : \{\mathcal{V}_{\mathsf{s}(\alpha)} \mid \P \hookrightarrow \mathsf{tp}_\alpha\} \textbf{ where}
$$
$$
\mathsf{syn} : \mathsf{tp}_\alpha
$$
$$
\mathsf{ext} : \{\mathcal{V} \mid \P \hookrightarrow \langle\Uparrow_\alpha^\Diamond\rangle\mathsf{syn}\}
$$

We will refer to an element of $\mathsf{tp}_\alpha^\star$ as a *pretype computability structure*, and the goal of this section is to add additional structures corresponding to the Kan operations.

(7.3.1∗4) hcom-*type computability structure.* We may define the type of hcom-*type computability structures* with the following alignment using **(7.2∗4)**:

$$
\textbf{record } \mathsf{hcomtp}_\alpha^\star : \{\mathcal{V}_{\mathsf{s}(\alpha)} \mid \P \hookrightarrow \mathsf{tp}_\alpha\} \textbf{ where}
$$
$$
\textbf{include } \mathsf{tp}_\alpha^\star
$$
$$
\mathsf{hcom} : \mathsf{HCom}^\star(\mathsf{syn},\mathsf{ext})
$$

The alignment above is correct because $\mathsf{HCom}^\star$ is valued in $\bullet$-modal types.

(7.3.1∗5) *Kan computability structure.* We additionally equip the type computability structure with coercion operations using a variant of the method of Licata, Orton, Pitts, and Spitters [Lic+18]. Let $A : \mathbb{I} \to \mathsf{tp}_\alpha^\star$ be a *line* of pretype computability structures; a well-aligned coercion structure for A is an element of the following type, once again $\bullet$-modal:

$$
\textstyle\prod_{r,s:\mathbb{I}}\prod_{a:A(r)}\{A(s) \mid \bullet(r = s) \hookrightarrow [\P \hookrightarrow \mathsf{coe}_A^{r\rightsquigarrow s}a, r = s \hookrightarrow a]\}
$$

The above determines a morphism $\mathsf{Coe}_\alpha^\star : (\mathsf{tp}_\alpha^\star)^{\mathbb{I}} \to \mathcal{V}_{\backslash\P}^\alpha$, and hence by adjoint transpose, a morphism $(\mathsf{Coe}_\alpha^\star)^\sharp : \mathsf{tp}_\alpha^\star \to (\mathcal{V}_{\backslash\P}^\alpha)_{\mathbb{I}}$. We may therefore combine the coercion and homogeneous composition structures to obtain a universe $\mathsf{tp}_\alpha^\square$ of *Kan* computability structures, writing $\dot{\mathcal{W}}$ for the object $\sum_{A:\mathcal{W}}A$ of pointed types classified by a universe $\mathcal{W}$:

$$
\begin{array}{ccccc}
\mathsf{tp}_\alpha^\square & \dashrightarrow & \mathsf{coetp}_\alpha^\star & \dashrightarrow & (\dot{\mathcal{V}}_{\backslash\P}^\alpha)_{\mathbb{I}} \\
\downarrow & & \downarrow & & \downarrow \\
\mathsf{hcomtp}_\alpha^\star & \xrightarrow[\pi]{} & \mathsf{tp}_\alpha^\star & \xrightarrow[(\mathsf{Coe}_\alpha^\star)^\sharp]{} & (\mathcal{V}_{\backslash\P}^\alpha)_{\mathbb{I}}
\end{array}
\qquad (7.3.1{*}5{*}1)
$$

We need to check that $\mathsf{tp}_\alpha^\square$ is aligned over tp_α, which we can do by applying the open modality to the outer pullback square, recalling the alignment of $\mathsf{hcomtp}_\alpha^\star$ and using the

fact that root functor preserves $\bigcirc$-connected objects **(7.3.1∗2)**.

$$
\begin{array}{ccc}
\bigcirc\mathsf{tp}^{\square}_{\alpha} \longrightarrow \bigcirc(\dot{\mathcal{V}}^{\alpha}_{\backslash\P})_{\mathbb{I}} & \bigcirc\mathsf{tp}^{\square}_{\alpha} \rightarrow \bigcirc((\dot{\mathcal{V}}^{\alpha}_{\backslash\P})_{\mathbb{I}}) & \bigcirc\mathsf{tp}^{\square}_{\alpha} \longrightarrow 1 \\
\downarrow\quad\quad\quad\downarrow & \downarrow\quad\quad\quad\downarrow & \downarrow\quad\quad\quad\downarrow \\
\bigcirc\mathsf{hcomtp}^{\star}_{\alpha} \longrightarrow \bigcirc(\mathcal{V}^{\alpha}_{\backslash\P})_{\mathbb{I}} & \mathsf{tp}_{\alpha} \rightarrow \bigcirc((\mathcal{V}^{\alpha}_{\backslash\P})_{\mathbb{I}}) & \mathsf{tp}_{\alpha} \longrightarrow 1
\end{array}
$$

Hence $\bigcirc\mathsf{tp}^{\square}_{\alpha} \cong (\mathsf{tp}_{\alpha} \times_1 1) \cong \mathsf{tp}_{\alpha}$, so we may align $\mathsf{tp}^{\square}_{\alpha}$ strictly over tp_{α}.

(7.3.1∗6) *Notation for Kan computability structures.* Considering Diagram 7.3.1∗5∗1 it is easy to see by transpose that every line of Kan computability structures $\mathrm{A} : \mathbb{I} \to \mathsf{tp}^{\square}_{\alpha}$ possesses a coercion operation $\mathsf{coe}_{\mathrm{A}}$. But how do we construct a specific Kan computability structure? The use of the root functor $(-)_{\mathbb{I}}$ in the definition of $\mathsf{tp}^{\square}_{\alpha}$ implies we must take parameters into account. Therefore we do not speak of constructing elements of $\mathsf{tp}^{\square}_{\alpha}$, but rather of constructing functions $\mathrm{E} \to \mathsf{tp}^{\square}_{\alpha}$. Suppose we have the folowing data:

$$
\begin{aligned}
\mathrm{A}[-] &: \mathrm{E} \to \mathsf{tp}^{\star}_{\alpha} \\
h[-] &: \textstyle\prod_{e:\mathrm{E}}\mathsf{HCom}^{\star}(\mathrm{A}(e)) \\
c[-] &: \textstyle\prod_{e:\mathrm{E}^{\mathbb{I}}}\mathsf{Coe}^{\star}(\lambda i.\mathrm{A}(e(i)))
\end{aligned}
$$

Then we specify the corresponding Kan computability structure $\mathrm{A}'[-]$ like so:

$$
\begin{aligned}
\mathrm{A}'[e] &\Leftarrow \mathbf{extend}\ \mathrm{A}[e] \\
\mathsf{hcom}^{r\rightsquigarrow s;\psi}_{\mathrm{A}'[e]}a &= h[e](r,s,\psi,a) \\
\mathsf{coe}^{r\rightsquigarrow s}_{\lambda i.\mathrm{A}'[e(i)]}a &= c[e](r,s,a)
\end{aligned}
$$

§7.3.2. Normalization structure of types

(7.3.2∗1) The normalization structure of types **(5.3∗1)** from Chapter 5 is almost sufficient, replacing the underlying computability structures with the Kan computability structures of **(7.3.1∗5)**. We would find, however, that the "induction hypothesis" of the Tait reflection operation is too weak to close the normalization structures under connectives. It is easy to see that a reflection operation with type $\prod_{\phi:\mathbb{F}\ast}\{\mathsf{ne}_{\phi}(\mathrm{A}) \to \mathrm{A} \mid \P \hookrightarrow \lambda x.x\}$ would be far too strong, considering the extreme case where $\phi = \top$ under which $\mathsf{ne}_{\phi}(\mathrm{A})$ destabilizes to $\mathsf{tm}(\mathrm{A})$. In that case, the reflection operator amounts to a choice of computability data for all terms simultaneously. The solution here is to stabilize the neutrals not against the normals, but against the given computability structure; in this way, the **up** operators for neutral forms of base types are essentially serialized versions of Tait reflection.

(7.3.2∗2) *Normalization structure of types.* We define the normalization structure of types, adjusting the reflection operator for $\mathrm{A} : \mathsf{tp}^{\square}_{\alpha}$ from **(5.3∗1)** to employ the stabilized reflection

structure defined in **(7.2∗3)**.

$$\textbf{record } \mathsf{tp}^{\circledast}_{\alpha} : \{\mathcal{V}_{\mathsf{s}(\alpha)} \mid \P \hookrightarrow \mathsf{tp}_{\alpha}\} \textbf{ where}$$
$$\textbf{include } \mathsf{tp}^{\square}_{\alpha} \textbf{ as } \mathsf{C}$$
$$\mathsf{N} : \textstyle\prod_{\beta \geq \alpha}\{\mathsf{nftp}_{\beta} \mid \P \hookrightarrow \langle\uparrow^{\beta}_{\alpha}\rangle\mathsf{ext}\}$$
$$\text{↯} : \textstyle\prod_{\phi:\mathbb{F}^{\circledast}}\mathsf{Reflect}[\phi](\mathsf{syn},\mathsf{ext})$$
$$\text{↴} : \mathsf{Reify}(\mathsf{syn},\mathsf{ext})$$

§7.3.3. Universe level coercions

(7.3.3∗1) *Kan computability structure.*

$$\boxed{\langle\uparrow^{\beta}_{\alpha}\rangle^{\square} : \{\mathsf{tp}^{\square}_{\alpha} \to \mathsf{tp}^{\square}_{\beta} \mid \P \hookrightarrow \langle\uparrow^{\beta}_{\alpha}\rangle\}}$$

$$(\langle\uparrow^{\beta}_{\alpha}\rangle^{\square}\mathrm{A}) \Leftarrow \textbf{extend } \langle\uparrow^{\beta}_{\alpha}\rangle^{\star}\mathrm{A}$$
$$\mathsf{hcom}^{r \rightsquigarrow s;\phi}_{\langle\uparrow^{\beta}_{\alpha}\rangle^{\square}\mathrm{A}} a = \mathsf{hcom}^{r \rightsquigarrow s;\phi}_{\mathrm{A}} a$$
$$\mathsf{coe}^{r \rightsquigarrow s}_{\lambda i.\langle\uparrow^{\beta}_{\alpha}\rangle^{\square}\mathrm{A}(i)} a = \mathsf{coe}^{r \rightsquigarrow s}_{\mathrm{A}} a$$

(7.3.3∗2) *Normalization structure.*

$$\boxed{\langle\uparrow^{\beta}_{\alpha}\rangle^{\circledast} : \{\mathsf{tp}^{\circledast}_{\alpha} \to \mathsf{tp}^{\circledast}_{\beta} \mid \P \hookrightarrow \langle\uparrow^{\beta}_{\alpha}\rangle\}}$$

$$\langle\uparrow^{\beta}_{\alpha}\rangle^{\circledast}\mathrm{A} \Leftarrow \textbf{extend } \langle\uparrow^{\beta}_{\alpha}\rangle^{\square}\mathrm{A}$$
$$\text{↴}^{\gamma \geq \beta}_{\mathsf{tp}}(\langle\uparrow^{\beta}_{\alpha}\rangle^{\circledast}\mathrm{A}) = \text{↴}^{\gamma \geq \alpha}_{\mathsf{tp}}\mathrm{A}$$
$$\text{↯}^{\phi}_{\langle\uparrow^{\beta}_{\alpha}\rangle^{\circledast}\mathrm{A}} a = \text{↯}^{\phi}_{\mathrm{A}} a$$
$$\text{↴}_{\langle\uparrow^{\beta}_{\alpha}\rangle^{\circledast}\mathrm{A}} a = \text{↴}_{\mathrm{A}} a$$

§7.3.4. Stabilized neutral types

(7.3.4∗1) Ultimately we will need to implement the Tait reflection operation for universes in §7.3.5, which entails taking a stabilized neutral type $\mathrm{A} : \mathsf{netp}^{\phi}_{n} \wr_{\phi} \mathsf{tp}^{\circledast}_{n}$ to a normalization structure that restricts to A on $\bullet\phi$. What should the elements of this normalization structure be? We recall from our experience with ordinary type theory (Chapter 5) that a neutral type can only contain neutral elements; because every type in cubical type theory is equipped with Kan operations, we must add to this a way to form formal coercions and homogeneous compositions in a neutral type. In this section, we define the normalization structure $\mathsf{elim}^{\circledast}_{n}[\phi](\mathrm{A})$ of *formal elimination trees* with type A, comprised of neutrals, homogeneous compositions, and coercions.

(7.3.4∗2) The presence of formal coercions and formal homogeneous compositions is reminiscent of and inspired by the Cavallo–Harper account of indexed higher inductive types in cubical type theory [Cav21; CH19].

(7.3.4∗3) *Computability structure.* We define $\mathbf{elim}_n^\phi : \{\mathsf{netp}_n^\phi \wr_\phi \mathsf{tp}_n^\star \to \mathcal{U} \mid \bullet\phi \hookrightarrow \langle\uparrow_n^\diamond\rangle^\star\}$ to be the smallest so-aligned family of types closed under the following constructors:

$$\frac{\phi,\psi : \mathbb{F}^* \qquad A : \mathsf{netp}_n^\phi \wr_\phi \mathsf{tp}_n^\star}{\mathbf{up}_A[\psi] : \mathsf{Reflect}[\psi](\langle\uparrow_n^\diamond\rangle A, \mathbf{elim}_n^\phi(A))}$$

$$\frac{\phi : \mathbb{F}^* \quad A : \mathsf{netp}_n^\phi \wr_\phi \mathsf{tp}_n^\star \quad r,s : \mathbb{I} \quad \psi : \mathbb{F}^* \quad a : \prod_{i:\mathbb{I}}\{r = s \vee \psi\}\,\mathbf{elim}_n^\phi(A)}{h : \{\phi\}\,\{\mathbf{elim}_n^\phi(A) \mid \bullet(r = s \vee \psi) \hookrightarrow \P \hookrightarrow \mathsf{hcom}_{\langle\uparrow_n^\diamond\rangle A}^{r\rightsquigarrow s;a}, r = s \vee \psi \hookrightarrow a(s)\}}{\mathbf{hcom}[\phi](A,r,s,\psi,a,h) : \{\mathbf{elim}_n^\phi(A) \mid \bullet(\phi \vee r = s \vee \psi) \hookrightarrow \P \hookrightarrow \mathsf{hcom}_{\langle\uparrow_n^\diamond\rangle A}^{r\rightsquigarrow s;\psi}a, \phi \hookrightarrow h, r = s \vee \psi \hookrightarrow a(s)\}}$$

$$\frac{\phi : \mathbb{I} \to \mathbb{F}^* \quad A : \prod_{i:\mathbb{I}}\mathsf{netp}_n^{\phi(i)} \wr_{\phi(i)} \mathsf{tp}_n^\star \quad r,s : \mathbb{I} \quad a : \mathbf{elim}_n^{\phi(r)}(A(r))}{c : \{\forall i.\phi(i)\}\,\{\mathbf{elim}_n^{\phi(s)}(A(s)) \mid \bullet(r = s) \hookrightarrow \P \hookrightarrow \mathsf{coe}_{\langle\uparrow_n^\diamond\rangle\circ A}^{r\rightsquigarrow s}a, r = s \hookrightarrow a\}}{\mathbf{coe}[\phi](A,r,s,a,c) : \{\mathbf{elim}_n^{\phi(s)}(A(s)) \mid \bullet((\forall i.\phi(i)) \vee r = s) \hookrightarrow \P \hookrightarrow \mathsf{coe}_{\langle\uparrow_n^\diamond\rangle A}^{r\rightsquigarrow s}a, \forall i.\phi(i) \hookrightarrow c, r = s \hookrightarrow a\}}$$

Using the above, we may define a computability structure for stabilized neutral types.

$$\boxed{\mathsf{elim}_n^\star[\phi] : \{\mathsf{netp}_n^\phi \wr_\phi \mathsf{tp}_n^\star \to \mathsf{tp}_n^\star \mid \bullet\phi \hookrightarrow \lambda A.A\}}$$

$$\mathsf{elim}_n^\star[\phi](A).\mathsf{syn} = A$$
$$\mathsf{elim}_n^\star[\phi](A).\mathsf{ext} = \mathbf{elim}_n^\phi(A)$$

(7.3.4∗4) *Kan computability structure.*

$$\boxed{\mathsf{elim}_n^\square[\phi] : \{\mathsf{netp}_n^\phi \wr_\phi \mathsf{tp}_n^\square \to \mathsf{tp}_n^\square \mid \bullet\phi \hookrightarrow \lambda A.A\}}$$

$$\mathsf{elim}_n^\square[\phi](A) \Leftarrow \mathbf{extend}\ \mathsf{elim}_n^\star[\phi](A)$$
$$\mathsf{hcom}_{\mathsf{elim}_n^\square[\phi](A)}^{r\rightsquigarrow s;\psi}a = \mathbf{hcom}[\phi](A,r,s,\psi,a,\mathsf{hcom}_A^{r\rightsquigarrow s;\psi}a)$$
$$\mathsf{coe}_{\lambda i.\mathsf{elim}_n^\square[\phi(i)](A(i))}^{r\rightsquigarrow s}a = \mathbf{coe}[\phi](A,r,s,a,\mathsf{coe}_A^{r\rightsquigarrow s}a)$$

In the definition of coercion above, we used the fact that universe levels are discrete and so any line $\mathbf{L}^{\mathbb{I}}$ is constant. The boundary under ϕ is satisfied trivially given the boundaries of the **hcom**/**coe** constructors.

(7.3.4∗5) *Normalization structure.* We extend the Kan computability structure of stabilized neutral types **(7.3.4∗3)** to a normalization structure.

$$\boxed{\mathsf{elim}_n^{\circledast}[\phi] : \{\mathsf{netp}_n^{\phi} \wr_{\phi} \mathsf{tp}_n^{\circledast} \to \mathsf{tp}_n^{\circledast} \mid \bullet\phi \hookrightarrow \lambda A.A\}}$$

$$\mathsf{elim}_n^{\circledast}[\phi](A) \Leftarrow \mathbf{extend}\ \mathsf{elim}_n^{\square}[\phi](A)$$

$$\Downarrow_{\mathsf{tp}}^{\beta\geq n}(\mathsf{elim}_n^{\circledast}[\phi](A)) = \mathbf{up}_n^{\alpha}[\phi](A.\mathsf{base}, \Downarrow_{\mathsf{tp}}^{\beta}A)$$

$$\hat{\imath}_{\mathsf{elim}_n^{\circledast}[\phi](A)}^{\psi}\,a = \mathbf{up}_A[\psi](a)$$

$$\Downarrow_{\mathsf{elim}_n^{\circledast}[\phi](A)}\mathbf{up}_A[\psi](a) = \widetilde{\mathbf{up}}_n[\phi,\psi](A.\mathsf{base}, [a.\mathsf{base} \mid \phi\wedge\psi \hookrightarrow \Downarrow_A a])$$

$$\Downarrow_{\mathsf{elim}_n^{\circledast}[\phi](A)}\mathbf{hcom}[\phi](A,r,s,\psi,a,h) = \mathbf{hcom}_n[\phi](A.\mathsf{base},r,s,\psi, \lambda i.\Downarrow_{\mathsf{elim}_n^{\circledast}[\phi](A)}a(i), \Downarrow_A h)$$

$$\Downarrow_{\mathsf{elim}_n^{\circledast}[\phi(s)](A)}\mathbf{coe}[\phi](A,r,s,a,c) = \mathbf{coe}_n[\phi](\lambda i.A.\mathsf{base}(i), r,s, \Downarrow_{\mathsf{elim}_n^{\circledast}[\phi(r)](A(r))}a, \Downarrow_{A(s)}c)$$

§7.3.5. Universes

(7.3.5∗1) *Computability structure.* For each finite universe level n, we define a computability structure for the universe U_n.

$$\boxed{\mathsf{U}_n^{\star} : \{\mathsf{tp}_{n+1}^{\star} \mid \P \hookrightarrow \mathsf{U}_n\}}$$

$$\mathsf{U}_n^{\star}.\mathsf{syn} = \mathsf{U}_n$$

$$\mathsf{U}_n^{\star}.\mathsf{ext} = \mathsf{tp}_n^{\circledast}$$

(7.3.5∗2) *Kan computability structure.*

$$\boxed{\mathsf{U}_n^{\square} : \{\mathsf{tp}_{n+1}^{\square} \mid \P \hookrightarrow \mathsf{U}_n\}}$$

$$\mathsf{U}_n^{\square} \Leftarrow \mathbf{extend}\ \mathsf{U}_n^{\star}$$

$$\mathsf{hcom}_{\mathsf{U}_n^{\square}}^{r\rightsquigarrow s;\phi}A = \langle\text{as in }(\mathbf{6.4∗14})\rangle$$

$$\mathsf{coe}_{\lambda i.\mathsf{U}_n^{\square}}^{r\rightsquigarrow s}A = A$$

(7.3.5∗3) *Normalization structure.* We extend the Kan computability structure of universes **(7.3.5∗2)** to a normalization structure, making use of **(7.3.4∗5)**.

$$\boxed{\mathsf{U}_n^{\circledast} : \{\mathsf{tp}_{n+1}^{\circledast} \mid \P \hookrightarrow \mathsf{U}_n\}}$$

$$\mathsf{U}_n^{\circledast} \Leftarrow \mathbf{extend}\ \mathsf{U}_n^{\square}$$

$$\Downarrow_{\mathsf{tp}}^{\beta\geq n+1}\mathsf{U}_n^{\circledast} = \mathbf{U}_{\beta}^{n}$$

$$\hat{\imath}_{\mathsf{U}_n^{\circledast}}^{\phi}A = \mathsf{elim}_n^{\circledast}[\phi](A)$$

$$\Downarrow_{\mathsf{U}_n^{\circledast}}A = \Downarrow_{\mathsf{tp}}^{n}A$$

§7.4. CLOSURE UNDER CONNECTIVES

§7.4.1. Dependent sum types

▨ **(7.4.1∗1)** *Kan computability structure.* The underlying computability structure $\Sigma_\alpha^\star$ of dependent sums is inherited from **(4.4.1∗3)**.

$$\boxed{\Sigma_\alpha^\square : \{(\textstyle\sum_{A:\mathsf{tp}_\alpha^\square}(A \to \mathsf{tp}_\alpha^\square)) \to \mathsf{tp}_\alpha^\square \mid \P \hookrightarrow \Sigma_\alpha\}}$$

$$\Sigma_\alpha^\square(A,B) \Leftarrow \mathbf{extend}\ \Sigma_\alpha^\star(A,B)$$

$$\mathsf{hcom}_{\Sigma_\alpha^\square(A,B)}^{r \rightsquigarrow s;\phi} p = \langle\text{as in }\mathbf{(6.4∗4)}\rangle$$

$$\mathsf{coe}_{\lambda i.\Sigma_\alpha^\square(A(i),B(i))}^{r \rightsquigarrow s} p = \langle\text{as in }\mathbf{(6.4∗4)}\rangle$$

▨ **(7.4.1∗2)** *Normalization structure.*

$$\boxed{\Sigma_\alpha^\circledast : \{(\textstyle\sum_{A:\mathsf{tp}_\alpha^\circledast}(A \to \mathsf{tp}_\alpha^\circledast)) \to \mathsf{tp}_\alpha^\circledast \mid \P \hookrightarrow \Sigma_\alpha\}}$$

$$\Sigma_\alpha^\circledast(A,B) \Leftarrow \mathbf{extend}\ \Sigma_\alpha^\square(A,B)$$

$$\Downarrow_{\mathsf{tp}}^{\beta \geq \alpha}\Sigma_\alpha^\circledast(A,B) = \mathbf{sg}_\beta(\Downarrow_{\mathsf{tp}}^\beta A, \lambda x.\Downarrow_{\mathsf{tp}}^\beta B(\Uparrow_A^\perp[\mathbf{var}(x) \mid \curlywedge \hookrightarrow []]))$$

$$\Uparrow_{\Sigma_\alpha^\circledast(A,B)}^\phi p = \mathbf{let}\ (p_1,p_2) = \mathbf{split}(p.\mathsf{base})\ \mathbf{in}$$
$$\mathbf{let}\ \tilde{p}_1 = \Uparrow_A^\phi[p_1 \mid \phi \hookrightarrow p.1]\ \mathbf{in}$$
$$(\tilde{p}_1, \Uparrow_{B(\tilde{p}_1)}^\phi[p_2 \mid \phi \hookrightarrow p.2])$$

$$\Downarrow_{\Sigma_\alpha^\circledast(A,B)} p = \mathbf{pair}(\Downarrow_A p.1, \Downarrow_{B(p.1)} p.2)$$

§7.4.2. Dependent product types

▨ **(7.4.2∗1)** *Kan computability structure.* The underlying computability structure $\Pi_\alpha^\star$ of dependent sums is inherited from **(4.4.1∗1)**.

$$\boxed{\Pi_\alpha^\square : \{(\textstyle\sum_{A:\mathsf{tp}_\alpha^\square}(A \to \mathsf{tp}_\alpha^\square)) \to \mathsf{tp}_\alpha^\square \mid \P \hookrightarrow \Pi_\alpha\}}$$

$$\Pi_\alpha^\square(A,B) \Leftarrow \mathbf{extend}\ \Pi_\alpha^\star(A,B)$$

$$\mathsf{hcom}_{\Pi_\alpha^\square(A,B)}^{r \rightsquigarrow s;\phi} p = \langle\text{as in }\mathbf{(6.4∗4)}\rangle$$

$$\mathsf{coe}_{\lambda i.\Pi_\alpha^\square(A(i),B(i))}^{r \rightsquigarrow s} p = \langle\text{as in }\mathbf{(6.4∗4)}\rangle$$

■ **(7.4.2∗2)** *Normalization structure.*

$$\boxed{\Pi_\alpha^* : \{(\textstyle\sum_{A:tp_\alpha^*}(A \to tp_\alpha^*)) \to tp_\alpha^* \mid \P \hookrightarrow \Pi_\alpha\}}$$

$$\Pi_\alpha^*(A,B) \Leftarrow \textbf{extend } \Pi_\alpha^\square(A,B)$$

$$\Downarrow_{tp}^{\beta \geq \alpha}\Pi_\alpha^*(A,B) = \textbf{pi}_\beta(\Downarrow_{tp}^\beta A, \lambda x.\Downarrow_{tp}^\beta B(\mathcal{I}_A^\perp[\textbf{var}(x) \mid \bot \hookrightarrow []]))$$

$$\mathcal{I}_{\Pi_\alpha^*(A,B)}^\phi f = \lambda x.\mathcal{I}_{B(x)}^\phi[\textbf{app}(f.\textbf{base}, \Downarrow_A x) \mid \phi \hookrightarrow f(x)]$$

$$\Downarrow_{\Pi_\alpha^*(A,B)} f = \textbf{lam}(\lambda x.\textbf{let } \tilde{x} = \mathcal{I}_A^\perp[\textbf{var}(x) \mid \bot \hookrightarrow []] \textbf{ in } \Downarrow_{B(\tilde{x})} f(\tilde{x}))$$

§7.4.3. Path types

■ **(7.4.3∗1)** *Computability structure.* We may define the computability structure of the path type by realignment as follows:

$$\boxed{path_\alpha^\star : \{(\textstyle\sum_{A:\mathbb{I}\to tp_\alpha^\star}\textstyle\prod_{i:\mathbb{I}}\{\partial^* i\}\, A(i)) \to tp_\alpha^\star \mid \P \hookrightarrow path_\alpha\}}$$

$$path_\alpha^\star(A,a).\textsf{syn} = path(A,a)$$

$$path_\alpha^\star(A,a).\textsf{ext} \cong \textstyle\prod_{i:\mathbb{I}}\{A(i) \mid \partial^* i \hookrightarrow a(i)\}$$

In order to make the path type constructor commute with universe level lifts, we must realign $path_\alpha^\star(A,a)$ along the $\bullet\bigvee_{n<\alpha}\exists A'.A = \langle\uparrow_n^\alpha\rangle^\star \circ A'$ as in **(4.4.3∗5)**.

■ **(7.4.3∗2)** *Kan computability structure.*

$$\boxed{path_\alpha^\square : \{(\textstyle\sum_{A:\mathbb{I}\to tp_\alpha^\square}\textstyle\prod_{i:\mathbb{I}}\{\partial^* i\}\, A(i)) \to tp_\alpha^\square \mid \P \hookrightarrow path_\alpha\}}$$

$$path_\alpha^\square(A,a) \Leftarrow \textbf{extend } path_\alpha^\star(A,a)$$

$$\textsf{hcom}_{path_\alpha^\square(A,a)}^{r\rightsquigarrow s;\psi}p = \langle\text{as in } \textbf{(6.4∗7)}\rangle$$

$$\textsf{coe}_{\lambda j.path_\alpha^\square(A(j),a(j))}^{r\rightsquigarrow s}p = \langle\text{as in } \textbf{(6.4∗7)}\rangle$$

■ **(7.4.3∗3)** *Normalization structure.*

$$\boxed{path_\alpha^* : \{(\textstyle\sum_{A:\mathbb{I}\to tp_\alpha^*}\textstyle\prod_{i:\mathbb{I}}\{\partial^* i\}\, A(i)) \to tp_\alpha^* \mid \P \hookrightarrow path_\alpha\}}$$

$$path_\alpha^*(A,a) \Leftarrow \textbf{extend } path_\alpha^\square(A,a)$$

$$\Downarrow_{tp}^{\beta \geq \alpha}path_\alpha^*(A,a) = \textbf{path}_\beta(\lambda i.\Downarrow_{tp}^\beta A(i), \lambda i.\Downarrow_{A(i)} a(i))$$

$$\mathcal{I}_{path_\alpha^*(A,a)}^\phi p = \lambda i.\mathcal{I}_{path_\alpha^*(A,a)}^{\phi\sqcup^* \partial^* i}[\textbf{papp}(p.\textbf{base},i) \mid \phi \sqcup^* \partial^* i \hookrightarrow [\phi \hookrightarrow p(i), \partial^* i \hookrightarrow a(i)]]$$

$$\Downarrow_{path_\alpha^*(A,a)} p = \textbf{plam}(\lambda i.\Downarrow_{A(i)} p(i))$$

§7.4.4. Glue types

(7.4.4∗1) For a token $\mu \in \{\star,\square,\dagger\}$, we will write Desc_α^μ for the following records:

$$\textbf{record } \mathsf{Desc}_\alpha^\mu : \{\mathcal{V}_{\mathsf{s}(\alpha)} \mid \P \hookrightarrow \mathsf{Desc}_\alpha\} \textbf{ where}$$
$$\phi : \mathbb{F}^{\circledast}$$
$$\mathsf{B} : \mathsf{tp}_\alpha^\mu$$
$$\mathsf{A} : \{\phi\}\, \mathsf{tp}_\alpha^\mu$$
$$\mathsf{f} : \{\phi\}\, \mathsf{Equiv}(\mathsf{A},\mathsf{B})$$

Given $\mathsf{D} : \mathsf{Desc}_\alpha^\star$ we will write $\langle\uparrow_\alpha^\beta\rangle^\star \mathsf{D} : \mathsf{Desc}_\beta^\star$ for the following lifted gluing data:

$$(\langle\uparrow_\alpha^\beta\rangle^\star \mathsf{D}).\phi = \mathsf{D}.\phi$$
$$(\langle\uparrow_\alpha^\beta\rangle^\star \mathsf{D}).\mathsf{B} = \langle\uparrow_\alpha^\beta\rangle^\star \mathsf{D}.\mathsf{B}$$
$$(\langle\uparrow_\alpha^\beta\rangle^\star \mathsf{D}).\mathsf{A} = \langle\uparrow_\alpha^\beta\rangle^\star \mathsf{D}.\mathsf{A}$$
$$(\langle\uparrow_\alpha^\beta\rangle^\star \mathsf{D}).\mathsf{f} = f$$

(7.4.4∗2) *Computability structure.*

$$\boxed{\mathsf{glue}_\alpha^\star : \{\textstyle\prod_{\mathsf{D}:\mathsf{Desc}_\alpha^\star}\{\mathsf{tp}_n^\star \mid \mathsf{D}.\phi \hookrightarrow \mathsf{D}.\mathsf{A}\} \mid \P \hookrightarrow \mathsf{glue}_\alpha\}}$$

$$\mathsf{glue}_\alpha^\star(\mathsf{D}).\mathsf{syn} = \mathsf{glue}_\alpha^\star(\mathsf{D})$$

$$\mathsf{glue}_\alpha^\star(\mathsf{D}).\mathsf{ext} \cong \textstyle\sum_{a:\{\mathsf{D}.\phi\}\,\mathsf{D}.\mathsf{A}}\{\mathsf{D}.\mathsf{B} \mid \mathsf{D}.\phi \hookrightarrow \mathsf{D}.\mathsf{f}(a)\}$$

The realignment above is taken with respect to $\bullet \mathsf{D}.\phi$. As in **(4.4.3∗5)** and **(7.4.3∗1)** we must realign once more at $\bullet\bigvee_{n<\alpha}\exists \mathsf{D}'.\mathsf{D} = \langle\uparrow_n^\alpha\rangle^\star \circ \mathsf{D}'$.

(7.4.4∗3) *Kan computability structure.*

$$\boxed{\mathsf{glue}_\alpha^\square : \{\textstyle\prod_{\mathsf{D}:\mathsf{Desc}_\alpha^\square}\{\mathsf{tp}_n^\square \mid \mathsf{D}.\phi \hookrightarrow \mathsf{D}.\mathsf{A}\} \mid \P \hookrightarrow \mathsf{glue}_\alpha\}}$$

$$\mathsf{glue}_\alpha^\square(\mathsf{D}) \Leftarrow \textbf{include } \mathsf{glue}_\alpha^\star(\mathsf{D})$$

$$\mathsf{hcom}_{\mathsf{glue}_\alpha^\square(\mathsf{D})}^{r\rightsquigarrow s;\psi}\, a = \langle\text{as in } \textbf{(6.4∗13)}\rangle$$

$$\mathsf{coe}_{\lambda i.\mathsf{glue}_\alpha^\square(\mathsf{D}(i))}^{r\rightsquigarrow s}\, a = \langle\text{as in } \textbf{(6.4∗13)}\rangle$$

(7.4.4∗4) *Normalization structure.*

$$\boxed{\mathsf{glue}_\alpha^\circledast : \{\textstyle\prod_{\mathsf{D}:\mathsf{Desc}_\alpha^\circledast}\{\mathsf{tp}_n^\circledast \mid \mathsf{D}.\phi \hookrightarrow \mathsf{D}.\mathsf{A}\} \mid \P \hookrightarrow \mathsf{glue}_\alpha\}}$$

$$\mathsf{glue}_\alpha^\circledast(\mathsf{D}) \Leftarrow \textbf{include } \mathsf{glue}_\alpha^\square(\mathsf{D})$$

$$\downarrow_{\mathsf{tp}}^{\beta\geq\alpha}\mathsf{glue}_\alpha^\circledast(\mathsf{D}) = \textbf{glue}_\alpha\big(\phi, \downarrow_{\mathsf{tp}}^\beta\mathsf{D}.\mathsf{B}, \downarrow_{\mathsf{tp}}^\beta\mathsf{D}.\mathsf{A}, \downarrow_{\mathsf{Equiv}(\mathsf{D}.\mathsf{A},\mathsf{D}.\mathsf{A})}\mathsf{D}.\mathsf{f}\big)$$

$$\Uparrow_{\mathsf{glue}_\alpha^\circledast(\mathsf{D})}^\psi g = \mathsf{englue}\,\big[\Uparrow_{\mathsf{D}.\mathsf{B}}^{\psi\sqcup^\circledast\phi}[\textbf{unglue}(g.\mathsf{base}) \mid \psi\sqcup^\circledast\phi \hookrightarrow [\phi \hookrightarrow f\big(\Uparrow_\mathsf{A}^\psi g\big), \psi \hookrightarrow \mathsf{unglue}(g)]] \mid \mathsf{D}.\phi \hookrightarrow \Uparrow_{\mathsf{D}.\mathsf{A}}^\psi g\big]$$

$$\downarrow_{\mathsf{glue}_\alpha^\circledast(\mathsf{D})}g = \textbf{englue}(\downarrow_\mathsf{A}g, \downarrow_\mathsf{B}\mathsf{unglue}(g))$$

§7.4.5. The weak booleans

(**7.4.5∗1**) *Computability structure.* We first define $\mathbf{bool} : \{\mathcal{U} \mid \P \hookrightarrow \mathsf{bool}\}$ to be the smallest type aligned over the syntactic booleans closed under the following constructors:

$$\mathbf{tt} : \{\mathbf{bool} \mid \P \hookrightarrow \mathsf{tt}\}$$

$$\mathbf{ff} : \{\mathbf{bool} \mid \P \hookrightarrow \mathsf{ff}\}$$

$$\mathbf{up}_{\mathbf{bool}}[\phi] : \mathsf{Reflect}[\phi](\mathsf{bool},\mathbf{bool})$$

$$\mathbf{hcom}_{\mathbf{bool}} : \mathsf{HCom}^{\star}(\mathsf{bool},\mathbf{bool})$$

Explicitly this can be done by defining an indexed inductive family of ○-connected types indexed in bool and taking a dependent sum; as we saw in earlier chapters, an indexed inductive family of ○-connected types is just a *quotient*-indexed inductive family of ordinary types. For intuition, $\mathbf{bool}$ should be thought of as the type of *values* of boolean type. We then define the computability structure of the booleans as follows:

$$\boxed{\mathsf{bool}^{\star}_{\alpha} : \{\mathsf{tp}^{\star}_{\alpha} \mid \P \hookrightarrow \mathsf{bool}_{\alpha}\}}$$

$$\mathsf{bool}^{\star}_{\alpha}.\mathsf{syn} = \mathsf{bool}_{\alpha}$$

$$\mathsf{bool}^{\star}_{\alpha}.\mathsf{ext} = \mathbf{bool}$$

(**7.4.5∗2**) *Kan computability structure.*

$$\boxed{\mathsf{bool}^{\square}_{\alpha} : \{\mathsf{tp}^{\square}_{\alpha} \mid \P \hookrightarrow \mathsf{bool}_{\alpha}\}}$$

$$\mathsf{bool}^{\square}_{\alpha} \Leftarrow \textbf{extend } \mathsf{bool}^{\star}_{\alpha}$$

$$\mathsf{hcom}^{r \rightsquigarrow s;\phi}_{\mathsf{bool}^{\square}_{\alpha}} b = \mathbf{hcom}_{\mathbf{bool}}(r,s,\phi,b)$$

$$\mathsf{coe}^{r \rightsquigarrow s}_{\lambda\_.\mathsf{bool}^{\square}_{\alpha}} b = b$$

(**7.4.5∗3**) *Normalization structure.*

$$\boxed{\mathsf{bool}^{\divideontimes}_{\alpha} : \{\mathsf{tp}^{\divideontimes}_{\alpha} \mid \P \hookrightarrow \mathsf{bool}_{\alpha}\}}$$

$$\mathsf{bool}^{\divideontimes}_{\alpha} \Leftarrow \textbf{extend } \mathsf{bool}^{\square}_{\alpha}$$

$$\Downarrow^{\beta \geq \alpha}_{\mathsf{tp}} \mathsf{bool}^{\divideontimes}_{\alpha} = \mathbf{bool}_{\beta}$$

$$\Uparrow^{\phi}_{\mathsf{bool}^{\divideontimes}_{\alpha}} b = \mathbf{up}_{\mathbf{bool}}[\phi](b)$$

$$\Downarrow_{\mathsf{bool}^{\divideontimes}_{\alpha}} \mathbf{tt} = \mathsf{tt}$$

$$\Downarrow_{\mathsf{bool}^{\divideontimes}_{\alpha}} \mathbf{ff} = \mathsf{ff}$$

$$\Downarrow_{\mathsf{bool}^{\divideontimes}_{\alpha}} \mathbf{up}_{\mathbf{bool}}[\phi](b) = \mathsf{up}_{\mathbf{bool}}[\phi]([b.\mathsf{base} \mid \phi \hookrightarrow \Downarrow_{\mathsf{bool}^{\divideontimes}_{\alpha}} b])$$

$$\Downarrow_{\mathsf{bool}^{\divideontimes}_{\alpha}} \mathbf{hcom}_{\mathbf{bool}}(r,s,\phi,b) = \mathsf{hcom}_{\mathsf{bool}}(r,s,\phi,\lambda i.\Downarrow_{\mathsf{bool}^{\divideontimes}_{\alpha}} b(i))$$

(**7.4.5∗4**) *Induction principle.*

$$\boxed{\left\{\mathsf{ind}^{\circledast}_{\mathsf{bool}} : \prod_{C:\mathsf{bool}^{\circledast}\to\mathsf{tp}^{\circledast}} \prod_{c_0:C(\mathsf{tt})} \prod_{c_1:C(\mathsf{ff})} \prod_{x:\mathsf{bool}^{\circledast}} C(x) \mid \P \hookrightarrow \mathsf{ind}_{\mathsf{bool}}\right\}}$$

$$\mathsf{ind}^{\circledast}_{\mathsf{bool}}(C,c_0,c_1,\mathsf{tt}) = c_0$$

$$\mathsf{ind}^{\circledast}_{\mathsf{bool}}(C,c_0,c_1,\mathsf{ff}) = c_1$$

$$\mathsf{ind}^{\circledast}_{\mathsf{bool}}(C,c_0,c_1,\mathsf{up}_{\mathsf{bool}}[\phi](b)) = \mathbf{let}\ \mathfrak{C}(x) = \mathop{\text{\reflectbox{$\lozenge$}}}_{\downarrow\mathsf{tp}} C\big(\overrightarrow{\iota^{\perp}_{\mathsf{bool}^{\circledast}}}[\mathbf{var}(x) \mid \perp \hookrightarrow []]\big)\ \mathbf{in}$$
$$\mathbf{let}\ \mathfrak{c}_0 = \mathop{\downarrow}_{C(\mathsf{tt})} c_0\ \mathbf{in}$$
$$\mathbf{let}\ \mathfrak{c}_1 = \mathop{\downarrow}_{C(\mathsf{ff})} c_1\ \mathbf{in}$$
$$\overrightarrow{\iota^{\phi}_{C(\mathsf{up}_{\mathsf{bool}}[\phi](b))}}[\mathbf{ind}_{\mathsf{bool}}(\mathfrak{C}, \mathfrak{c}_0, \mathfrak{c}_1, b.\mathsf{base}) \mid \phi \hookrightarrow \mathsf{ind}^{\circledast}_{\mathsf{bool}}(C,c_0,c_1,b)]$$

$$\mathsf{ind}^{\circledast}_{\mathsf{bool}}(C,c_0,c_1,\mathsf{hcom}_{\mathsf{bool}}(r,s,\phi,b)) = \mathsf{com}^{r\leadsto s;\phi}_{\lambda j.C(\mathsf{hcom}^{r\leadsto j;\phi}_{\mathsf{bool}^{\circledast}} b)} \lambda i.\mathsf{ind}^{\circledast}_{\mathsf{bool}}(C,c_0,c_1,b(i))$$

§7.4.6. The circle

■ **(7.4.6∗1)** *Computability structure.* We define $\mathbf{S}^1 : \{\mathcal{U} \mid \P \hookrightarrow \mathsf{S}^1\}$ to be the smallest type aligned over the elements of the syntactic circle closed under the following constructors:

$$\mathbf{base} : \{\mathbf{S}^1 \mid \P \hookrightarrow \mathsf{base}\}$$
$$\mathbf{loop} : \prod_{i:\mathbb{I}}\{\mathbf{S}^1 \mid \bullet\partial^{\circledast}i \hookrightarrow [\P \hookrightarrow \mathsf{loop}(i), \partial^{\circledast}i \hookrightarrow \mathbf{base}]\}$$
$$\mathbf{up}_{\mathsf{S}^1}[\phi] : \mathsf{Reflect}[\phi](\mathsf{S}^1, \mathbf{S}^1)$$
$$\mathbf{hcom}_{\mathsf{S}^1} : \mathsf{HCom}^{\star}(\mathsf{S}^1, \mathbf{S}^1)$$

We may then define the computability structure of the circle as follows:

$$\boxed{(\mathsf{S}^1_{\alpha})^{\star} : \{\mathsf{tp}^{\star}_{\alpha} \mid \P \hookrightarrow \mathsf{S}^1_{\alpha}\}}$$

$$(\mathsf{S}^1_{\alpha})^{\star}.\mathsf{syn} = \mathsf{S}^1_{\alpha}$$
$$(\mathsf{S}^1_{\alpha})^{\star}.\mathsf{ext} = \mathbf{S}^1$$

■ **(7.4.6∗2)** *Kan Computability structure.*

$$\boxed{(\mathsf{S}^1_{\alpha})^{\square} : \{\mathsf{tp}^{\square}_{\alpha} \mid \P \hookrightarrow \mathsf{S}^1_{\alpha}\}}$$

$$(\mathsf{S}^1_{\alpha})^{\square} \Leftarrow \mathbf{extend}\ (\mathsf{S}^1_{\alpha})^{\star}$$
$$\mathsf{hcom}^{r\leadsto s;\phi}_{(\mathsf{S}^1_{\alpha})^{\square}} b = \mathbf{hcom}_{\mathsf{S}^1}(r,s,\phi,b)$$
$$\mathsf{coe}^{r\leadsto s}_{\lambda\_.(\mathsf{S}^1_{\alpha})^{\square}} b = b$$

■ **(7.4.6∗3)** *Normalization structure.*

$$\boxed{(\mathsf{S}^1_\alpha)^{\circledast} : \{\mathsf{tp}^{\circledast}_\alpha \mid \P \hookrightarrow \mathsf{S}^1_\alpha\}}$$

$$(\mathsf{S}^1_\alpha)^{\circledast} \Leftarrow \mathbf{extend}\ (\mathsf{S}^1_\alpha)^{\square}$$

$$\downarrow^{\beta \geq \alpha}_{\mathsf{tp}}(\mathsf{S}^1_\alpha)^{\circledast} = \mathsf{S}^1_\beta$$

$$\mathbb{7}^\phi_{(\mathsf{S}^1_\alpha)^{\circledast}}b = \mathbf{up}_{\mathsf{S}^1}[\phi](b)$$

$$\downarrow_{(\mathsf{S}^1_\alpha)^{\circledast}}\mathbf{base} = \mathbf{base}$$

$$\downarrow_{(\mathsf{S}^1_\alpha)^{\circledast}}\mathbf{loop}(i) = \mathbf{loop}(i)$$

$$\downarrow_{(\mathsf{S}^1_\alpha)^{\circledast}}\mathbf{up}_{\mathsf{S}^1}[\phi](b) = \mathbf{up}_{\mathsf{S}^1}[\phi]([b.\mathbf{base} \mid \phi \hookrightarrow \downarrow_{(\mathsf{S}^1_\alpha)^{\circledast}}b])$$

$$\downarrow_{(\mathsf{S}^1_\alpha)^{\circledast}}\mathbf{hcom}_{\mathsf{S}^1}(r,s,\phi,b) = \mathbf{hcom}_{\mathsf{S}^1}(r,s,\phi,\lambda i.\downarrow_{(\mathsf{S}^1_\alpha)^{\circledast}}b(i))$$

■ **(7.4.6∗4)** *Induction principle.*

$$\boxed{\left\{\mathsf{ind}^{\circledast}_{\mathsf{S}^1} : \prod_{C:(\mathsf{S}^1)^{\circledast} \to \mathsf{tp}^{\circledast}}\prod_{c_b:C(\mathbf{base})}\prod_{c_l:\prod_{i:\mathbb{I}}\{C(\mathbf{loop}(i))\mid \partial i \hookrightarrow c_b\}}\prod_{x:(\mathsf{S}^1)^{\circledast}}C(x) \mid \P \hookrightarrow \mathsf{ind}_{\mathsf{S}^1}\right\}}$$

$$\mathsf{ind}^{\circledast}_{\mathsf{S}^1}(C,c_b,c_l,\mathbf{base}) = c_b$$

$$\mathsf{ind}^{\circledast}_{\mathsf{S}^1}(C,c_b,c_l,\mathbf{loop}(i)) = c_l(i)$$

$$\mathsf{ind}^{\circledast}_{\mathsf{S}^1}(C,c_b,c_l,\mathbf{up}_{\mathsf{S}^1}[\phi](b)) = \mathbf{let}\ \mathfrak{C}(x) = \downarrow^{\lozenge}_{\mathsf{tp}}C\big(\mathbb{7}^{\perp}_{(\mathsf{S}^1)^{\circledast}}[\mathbf{var}(x) \mid \bot \hookrightarrow []]\big)\ \mathbf{in}$$
$$\mathbf{let}\ \mathfrak{c}_b = \downarrow_{C(\mathbf{base})}c_b\ \mathbf{in}$$
$$\mathbf{let}\ \mathfrak{c}_l(i) = \downarrow_{C(\mathbf{loop}(i))}c_l(i)\ \mathbf{in}$$
$$\mathbb{7}^\phi_{C(\mathbf{up}_{\mathsf{S}^1}[\phi](b))}[\mathbf{ind}_{\mathsf{S}^1}(\mathfrak{C}, \mathfrak{c}_b, \mathfrak{c}_l, b.\mathbf{base}) \mid \phi \hookrightarrow \mathsf{ind}^{\circledast}_{\mathsf{S}^1}(C,c_b,c_l,b)]$$

$$\mathsf{ind}^{\circledast}_{\mathsf{S}^1}(C,c_b,c_l,\mathbf{hcom}_{\mathsf{S}^1}(r,s,\phi,b)) = \mathbf{com}^{r\rightsquigarrow s;\phi}_{\lambda j.C(\mathbf{hcom}^{r\rightsquigarrow j;\phi,b}_{(\mathsf{S}^1)^{\circledast}}b)}\ \lambda i.\mathsf{ind}^{\circledast}_{\mathsf{S}^1}(C,c_b,c_l,b(i))$$

§7.5. THE CUBICAL NORMALIZATION TOPOS

※ **(7.5∗1)** In this section, we construct a topos $\mathbf{G}_{\square}$ satisfying all the assumptions of the foregoing constructions.

§7.5.1. The cubical atomic figure shape

■ **(7.5.1∗1)** As in §5.5.1 we will define a notion of *cubical atomic context*, this time adding context extensions by the interval.

$$\boxed{\Gamma\ ctx \rightsquigarrow \mathrm{X}}\ presupposing\ \mathrm{X} : \mathcal{T}_{\square}$$

$$\frac{}{\mathbb{1}\ ctx \rightsquigarrow \mathbf{1}_{\mathcal{T}_{\square}}} \qquad \frac{\Gamma\ ctx \rightsquigarrow \mathrm{X} \qquad \mathrm{A} : \mathrm{X} \longrightarrow \mathsf{tp}}{\Gamma.\mathrm{A}\ ctx \rightsquigarrow \sum_{x:\mathrm{X}}\mathsf{tm}(\mathrm{A}(x))} \qquad \frac{\Gamma\ ctx \rightsquigarrow \mathrm{X}}{\Gamma.\mathbb{I}\ ctx \rightsquigarrow \mathrm{X} \times \mathbb{I}}$$

▨ **(7.5.1∗2)** *Atomic terms.* For cubical type theory, an "atomic term" is either a term variable or a dimension; the term variables are defined as in **(5.5.1∗2)**:

$$\boxed{\Gamma \Vdash \mathfrak{a} : A \rightsquigarrow a} \; presupposing \; \begin{cases} \Gamma \; ctx \; \rightsquigarrow X \\ A : X \to \mathsf{tp} \\ a : \prod_{x:X}\mathsf{tm}(A(x)) \end{cases}$$

TOP VARIABLE

$$\dfrac{\Gamma \; ctx \; \rightsquigarrow X \qquad A : X \to \mathsf{tp}}{\Gamma.A \Vdash \mathbf{q}_A : A \circ \pi_1 \rightsquigarrow \pi_2}$$

POP VARIABLE

$$\dfrac{\Gamma \; ctx \; \rightsquigarrow X \qquad A,B : X \to \mathsf{tp} \qquad a : \prod_{x:X}\mathsf{tm}(A)(x) \qquad \Gamma \Vdash \mathfrak{a} : A \rightsquigarrow a}{\Gamma.B \Vdash \mathbf{p}_A(\mathfrak{a}) : A \circ \pi_1 \rightsquigarrow a \circ \pi_1}$$

The above extends pointwise to give a notion of simultaneous atomic substitutions:

$$\boxed{\Delta \Vdash \gamma : \Gamma \rightsquigarrow y} \; presupposing \; \begin{cases} \Delta \; ctx \; \rightsquigarrow X \\ \Gamma \; ctx \; \rightsquigarrow Y \\ y : X \to Y \end{cases}$$

NIL

$$\dfrac{\Delta \; ctx \; \rightsquigarrow X}{\Delta \Vdash \cdot : \mathbb{1} \rightsquigarrow !_X}$$

SNOC

$$\dfrac{\begin{matrix}\Delta \; ctx \; \rightsquigarrow X \qquad \Gamma \; ctx \; \rightsquigarrow Y \\ A : Y \to \mathsf{tp} \qquad y : X \to Y \qquad a : \prod_{x:X}\mathsf{tm}(A(y(x))) \\ \Delta \Vdash \gamma : \Gamma \rightsquigarrow y \qquad \Delta \Vdash \mathfrak{a} : A \circ y \rightsquigarrow a\end{matrix}}{\Delta \Vdash \gamma.\mathfrak{a} : \Gamma.A \rightsquigarrow (y,a)}$$

DIMENSION

$$\dfrac{\begin{matrix}\Delta \; ctx \; \rightsquigarrow X \qquad \Gamma \; ctx \; \rightsquigarrow Y \qquad y : X \to Y \\ \Delta \Vdash \gamma : \Gamma \rightsquigarrow y \qquad r : X \to \mathbb{I}\end{matrix}}{\Delta \Vdash \gamma.r : \Gamma.\mathbb{I} \rightsquigarrow (y,r)}$$

▨ **(7.5.1∗3)** *Cubical atomic figure shape.* It is not difficult to see that **(7.5.1∗1)** and **(7.5.1∗2)** give rise to a category $\mathcal{A}_\square$ equipped with a functor $\alpha_\square : \mathcal{A}_\square \to \mathcal{T}_\square$ into the syntactic category of cubical type theory, just as in §5.5.1. Writing $\mathbf{T}_\square$ and $\mathbf{A}_\square$ for the quasi-affine topoi $\widehat{\mathcal{T}_\square}$ and $\widehat{\mathcal{A}_\square}$ respectively, we then have a *cubical atomic figure shape* $\alpha_\square : \mathbf{A}_\square \to \mathbf{T}_\square$ given by the corresponding essential morphism of topoi.

▨ **(7.5.1∗4)** *The interval is preserved.* The cubical atomic figure shape preserves the interval, in the sense that we have $\alpha_\square^* \mathsf{y}_{\mathcal{T}_\square}(\mathbb{I}) \cong \mathsf{y}_{\mathcal{A}_\square}(\mathbb{1}.\mathbb{I})$.

Proof. This is established by computing naturally in $\Gamma : \mathcal{A}_\square$, by inversion on the inductive definition of cubical atomic substitutions into any context of the form $\Delta.\mathbb{I}$ from **(7.5.1∗2)**.

$$\mathrm{Hom}_{\mathrm{Pr}(\mathcal{A}_\square)}(\mathsf{y}_{\mathcal{A}_\square}(\Gamma), \mathsf{y}_{\mathcal{A}_\square}(\mathbb{1}.\mathbb{I}))$$

$$\cong \mathrm{Hom}_{\mathcal{A}_\square}(\Gamma, \mathbb{1}.\mathbb{I})$$
$$\cong \mathrm{Hom}_{\mathcal{T}_\square}(\alpha_\square(\Gamma), \mathbb{I})$$
$$\cong \mathrm{Hom}_{\mathrm{Pr}(\mathcal{T}_\square)}(\mathsf{y}_{\mathcal{T}_\square}(\alpha_\square(\Gamma)), \mathsf{y}_{\mathcal{T}_\square}(\mathbb{I}))$$
$$\cong \mathrm{Hom}_{\mathrm{Pr}(\mathcal{T}_\square)}((\alpha_\square)_! \mathsf{y}_{\mathcal{A}_\square}(\Gamma), \mathsf{y}_{\mathcal{T}_\square}(\mathbb{I}))$$
$$\cong \mathrm{Hom}_{\mathrm{Pr}(\mathcal{A}_\square)}(\mathsf{y}_{\mathcal{A}_\square}(\Gamma), \alpha_\square^* \mathsf{y}_{\mathcal{T}_\square}(\mathbb{I})) \qquad \square$$

§7.5.2. The normalization topos and its open and closed subtopoi

▣ **(7.5.2∗1)** Using the same construction as §5.5.3, we may define a suitable topos $\mathbf{G}_\square$ by considering the Artin gluing of the figure shape $\alpha_\square : \mathbf{A}_\square \to \mathbf{T}_\square$.

As ever, we will write ¶ for the *syntactic open* that presents $\mathbf{T}_\square$ as $(\mathbf{G}_\square)_\P$, *i.e.* the subterminal object in $\mathbf{Set}_{\mathbf{G}_\square}$ that presents $\mathrm{Pr}(\mathcal{T}_\square)$ as $(\mathbf{Set}_{\mathbf{G}_\square})_{/\P}$.

▣ **(7.5.2∗2)** *The syntactic algebra.* The syntactic algebra $\mathbf{M}$ for cubical type theory in $\mathbf{Set}_{\mathbf{G}_\square}$ is obtained by pushing forward the generic model of cubical type theory from $\mathbf{Set}_{\mathbf{T}_\square}$ along the open immersion $j : \mathbf{T}_\square \hookrightarrow \mathbf{G}_\square$ as follows:

▣ **(7.5.2∗3)** Write $\mathbb{I} : \mathbf{Set}_{\mathbf{G}_\square}$ for the purely syntactic interval $j_* \mathsf{y}_{\mathcal{T}_\square}(\mathbb{I})$ embedded into the glued logos, and write $0,1 : \mathbb{I}$ for its endpoints. It is not the case that $\bot$ is the initial object in $\mathbf{Set}_{\mathbf{G}_\square}$, nor in $\mathbf{Set}_{\mathbf{T}_\square}$ — this can be seen because $\bot$ is representable in $\mathrm{Pr}(\mathcal{T}_\square)$. On the other hand, we will observe that $\alpha_\square^* \bot$ is the initial object of $\mathbf{Set}_{\mathbf{A}_\square}$ and hence a sheaf on $\mathbf{G}_\square$ can fail to treat $\bot$ as false *only* in a syntactic way.

▣ **(7.5.2∗4)** The restriction of $\bot$ over $\mathbf{A}_\square$ is empty, *i.e.* we have $\alpha_\square^* \bot = \bot : \mathcal{O}_{\mathbf{A}_\square}$.

Proof. Because $\mathbf{A}_\square$ is presented by $\widehat{\mathcal{A}_\square}$, it is enough to observe that $\mathrm{Hom}_{\mathcal{T}_\square}(\boldsymbol{\alpha}_\square(\Gamma), \curlywedge)$ is the empty set for all cubical atomic contexts $\Gamma : \mathcal{A}_\square$. To see that this holds, suppose to the contrary that we have a term $u : \boldsymbol{\alpha}_\square(\Gamma) \longrightarrow \curlywedge$ in cubical type theory for some Γ.

We may define a non-standard model of $\mathcal{T}_\square$ in the category $\mathbf{Set}_\square$ of Cartesian cubical sets, in which the interval is interpreted by the generic strictly bipointed set and hence $[\![0]\!] \neq [\![1]\!]$, and yet the interpretation of every atomic context Γ has a global point $t_\Gamma : \mathbf{1}_{\mathbf{Set}_\square} \longrightarrow [\![\boldsymbol{\alpha}_\square(\Gamma)]\!]$. By composition with the interpretation of our assumption u, we have a global point $[\![u]\!] \circ t_\gamma$ of $[\![\curlywedge]\!]$ and thence a contradiction. Such a model construction is possible by defining $[\![\mathsf{tp}]\!]$ to be the singleton cubical set, with $[\![\mathsf{tm}]\!](*)$ a singleton as well. The closure of this interpretation under the constructs of cubical type theory is immediate for "negative" connectives; for "positive" types like the booleans and the circle, we observe that the corresponding induction principles target only elements of *types* (singletons), and hence are uniquely determined. $\qquad\square$

(7.5.2∗5) Suppose that cubical type theory had *types* that strictly classified proofs of arbitrary cofibrations; then **(7.5.2∗4)** would be refuted.

(7.5.2∗6) In the frame of opens $\mathcal{O}_{\mathbf{G}_\square}$, we have $\curlywedge \leq \P$.

Proof. It suffices to check that $\boldsymbol{j}^*\curlywedge \leq \boldsymbol{j}^*\P$ and $\boldsymbol{i}^*\curlywedge \leq \boldsymbol{i}^*\P$, but the former is immediate because $\boldsymbol{j}^*\P = \top$ by definition. As for the restriction over $\mathbf{A}_\square$, we make use of our lemma **(7.5.2∗4)** in the following computation:

$$\boldsymbol{i}^*\curlywedge = \boldsymbol{i}^*\boldsymbol{j}_*\curlywedge = \boldsymbol{\alpha}^*\curlywedge = \perp \qquad\qquad \square$$

(7.5.2∗7) The interval $\mathbb{I} : \mathbf{Set}_{\mathbf{G}_\square}$ is tiny.

Proof. By the gluing lemma for tiny objects **(2.5∗3)** it suffices to check that both $\boldsymbol{j}^*\mathbb{I}$ and $\boldsymbol{i}^*\mathbb{I}$ are tiny sheaves on $\mathbf{T}_\square$ and $\mathbf{A}_\square$ respectively. Because both $\mathcal{T}_\square$ and $\mathcal{A}_\square$ have finite products, by **(2.5∗2)** it is enough to observe that both presheaves $\boldsymbol{j}^*\mathbb{I}$ and $\boldsymbol{i}^*\mathbb{I}$ are represented by $\mathbb{I} : \mathcal{T}_\square$ and $\mathbb{1}.\mathbb{I} : \mathcal{A}_\square$ respectively, the latter following from **(7.5.1∗4)**. $\qquad\square$

(7.5.2∗8) *Computability structure of variables.* The type of variables $\mathsf{var}(-)$ may be constructed exactly as in **(5.5.3∗3)**.

§7.6. THE NORMALIZATION RESULT

§7.6.1. The normalization function

(7.6.1∗1) *Preliminaries.* Write $\mathbf{M}^*$ for the normalization algebra for $\mathcal{T}_\square$ induced by the foregoing constructions. In this section, we deduce the normalization theorem for cubical type theory; we inherit from §5.6 the definitions of the constants depicted in Fig. 7.1.
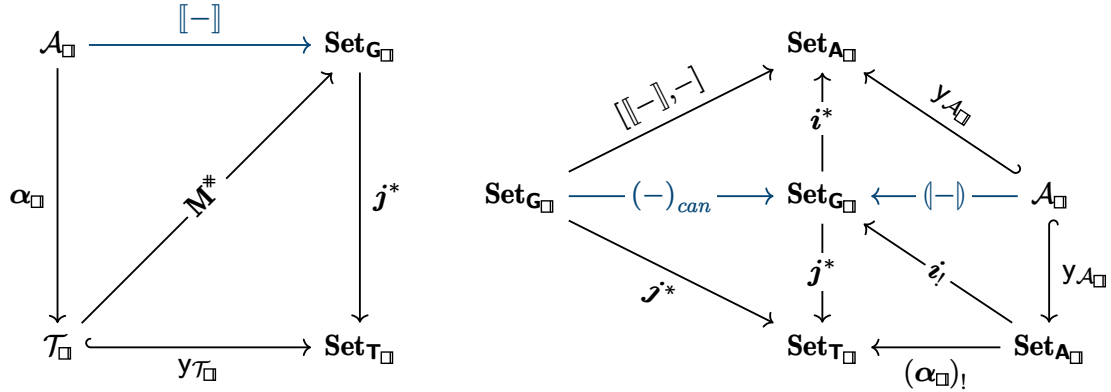
Figure 7.1: A summary of three functors used to implement the normalization function.

(7.6.1∗2) *Reflection of atomic substitutions.* The pointwise vertical natural transformation $\overline{\jmath}_{\mathsf{atm}} : (\!|-|\!) \longrightarrow [\![-]\!]$ from **(5.6.2∗3)** can defined in the cubical setting as in the following way, using the fact that variables have empty frontiers of instability:

$$\overline{\jmath}^{\Gamma}_{\mathsf{atm}} : (\!|\Gamma|\!) \longrightarrow [\![\Gamma]\!]$$
$$\overline{\jmath}^{\mathbb{1}}_{\mathsf{atm}}(\cdot) = \cdot$$
$$\overline{\jmath}^{\Gamma.A}_{\mathsf{atm}}(\gamma.\mathfrak{a}) = \big(\overline{\jmath}^{\Gamma}_{\mathsf{atm}}(\gamma), \overline{\jmath}^{\downarrow}_{\mathbf{M}^{\divideontimes}(A)}[\mathbf{var}(\mathfrak{a}) \mid \perp \hookrightarrow [\!]]\big)$$
$$\overline{\jmath}^{\Gamma.\mathbb{I}}_{\mathsf{atm}}(\gamma.r) = \big(\overline{\jmath}^{\Gamma}_{\mathsf{atm}}(\gamma), r\big)$$

(7.6.1∗3) For any sort $\mathrm{T} : \mathcal{T}_{\square}$, we likewise have a vertical "evaluation" morphism $\mathsf{eval}_{\mathrm{T}} : \mathbf{M}(\mathrm{T}) \longrightarrow \mathbf{M}^{\divideontimes}(\mathrm{T})_{can}$ just as in **(5.6.2∗6)**, and hence by composition with $\overline{\jmath}_{\mathsf{atm}}$ from **(7.6.1∗2)** we obtain a vertical map $[\![-]\!]_{\mathrm{T}} : \mathbf{M}(\mathrm{T}) \longrightarrow \mathbf{M}^{\divideontimes}(\mathrm{T})$ taking syntax to computability data. When it causes no ambiguity, we will write $[\![-]\!]$ for $[\![-]\!]_{\mathrm{T}}$.

(7.6.1∗4) *The normalization function.* Using the above we may define a sound and complete normalization function for types and terms exactly as in **(5.6.3∗1)**:

$$\mathsf{tp} \xrightarrow{[\![-]\!]_{\mathsf{tp}}} \mathsf{tp}^{\divideontimes} \xrightarrow{\downarrow^{\lozenge}_{\mathsf{tp}}} \mathsf{nftp} \qquad\qquad \mathsf{tm} \xrightarrow{[\![-]\!]_{\mathsf{tm}}} \mathsf{tm}^{\divideontimes} \xrightarrow{\lambda(A,a).(A, \downarrow_A a)} \sum_{A:\mathsf{tp}}\mathsf{nf}(A)$$
$$\underset{\mathsf{nbe}_{\mathsf{tp}}}{\nearrow} \qquad\qquad\qquad\qquad \underset{\mathsf{nbe}_{\mathsf{tm}}}{\nearrow}$$

Soundness and completeness follow immediately from the argument of **(5.6.3∗2)**.

§7.6.2. Idempotence of normalization

(7.6.2∗1) The purpose of this section is to show that the normalization function is *idempotent* in the sense that normalizing the underlying term of a normal form yields the same normal form. This can be achieved by showing that every normal form is *fixed* or *stationary* with respect to normalization, which we verify by considering several representative cases, *sc.* (7.6.2∗4) and (7.6.2∗8) through (7.6.2∗12).

(7.6.2∗2) *Standard stabilization.* Let $a : \mathsf{ne}_\phi(A)$ be a neutral; in this section, we will write $\langle\!\langle a \rangle\!\rangle$ for the stabilized neutral $[a \mid \phi \hookrightarrow [\![a]\!]] : \mathsf{ne}_\phi(A) \wr_\phi [\![A]\!]$.

(7.6.2∗3) We define what it means for a normal or neutral form to be *stationary* with respect to normalization:

1) A neutral form $a : \mathsf{ne}_\phi(A)$ is stationary when $[\![a]\!] = \hat{\gamma}^\phi_{[\![A]\!]}\langle\!\langle a \rangle\!\rangle$.
2) A normal form $a : \mathsf{nf}(A)$ is stationary when $a = \mathsf{nbe}_{\mathsf{tm}}(A,a)$.
3) A normal form $A : \mathsf{nftp}$ is stationary when $A = \mathsf{nbe}_{\mathsf{tp}}(A)$.

(7.6.2∗4) *Variables are stationary.* For any variable $x : \mathsf{var}(A)$, we have $[\![x]\!] = \hat{\gamma}^\perp_{[\![A]\!]}[\mathsf{var}(x) \mid \perp \hookrightarrow [\,]]$. In other words, variables in the normalization algebra are always interpreted by the reflection.

Proof. Exactly as in (5.6.5∗2). □

(7.6.2∗5) Let $A : \mathsf{netp}^\phi_n$ be a stationary neutral type; then $[\![A]\!]$ is the normalization structure $\mathsf{elim}^*_n[\phi](\langle\!\langle A \rangle\!\rangle)$ of formal elimination trees.

Proof. By computation.

$$[\![A]\!] = \hat{\gamma}^\phi_{[\![\mathsf{U}_n]\!]}\langle\!\langle A \rangle\!\rangle \qquad\qquad \text{by assumption}$$
$$= \hat{\gamma}^\phi_{\mathsf{U}^*_n}\langle\!\langle A \rangle\!\rangle \qquad\qquad \text{unfolding}$$
$$= \mathsf{elim}^*_n[\phi](\langle\!\langle A \rangle\!\rangle) \qquad\qquad \text{unfolding} \qquad\qquad \square$$

(7.6.2∗6) Let $A : \mathsf{netp}^\phi_n$ be a stationary neutral type and let $a : \mathsf{ne}_\psi(\langle\uparrow^\diamond_n\rangle A)$ be a stationary neutral element of A. Then the evaluation $[\![a]\!]$ is the formal elimination tree $\mathbf{up}_{\langle\!\langle A \rangle\!\rangle}[\psi](\langle\!\langle a \rangle\!\rangle)$.

Proof. By computation.

$$[\![a]\!] = \hat{\gamma}^\psi_{[\![A]\!]}\langle\!\langle a \rangle\!\rangle \qquad\qquad \text{assumption}$$
$$= \hat{\gamma}^\psi_{\mathsf{elim}^*_n[\phi](\langle\!\langle A \rangle\!\rangle)}\langle\!\langle a \rangle\!\rangle \qquad\qquad \textbf{(7.6.2∗5)}$$
$$= \mathbf{up}_{\langle\!\langle A \rangle\!\rangle}[\psi](\langle\!\langle a \rangle\!\rangle) \qquad\qquad \text{unfolding} \qquad\qquad \square$$

(7.6.2∗7) *Evaluation of stuck coercions.* Let $A : \prod_{i:\mathbb{I}} \mathsf{netp}_n^{\phi(i)}$ be a line of neutral types, let $r,s : \mathbb{I}$ be dimensions, and let $a : \mathsf{nf}(\langle\uparrow_n^\diamond\rangle A(r))$ be a normal form. Then we have $[\![\mathsf{coe}^{r\leadsto s}_{\langle\uparrow_n^\diamond\rangle\circ A} a]\!] = \mathbf{coe}[\phi](\langle\!\langle A\rangle\!\rangle, r, s, [\![a]\!], [\![\mathsf{coe}^{r\leadsto s}_{\langle\uparrow_n^\diamond\rangle\circ A} a]\!])$.

Proof. By computation:

$$
[\![\mathsf{coe}^{r\leadsto s}_{\langle\uparrow_n^\diamond\rangle\circ A} a]\!]
$$

$$
= \mathsf{coe}^{r\leadsto s}_{\lambda i.[\![A(i)]\!]} [\![a]\!] \hspace{3cm} \text{unfolding}
$$

$$
= \mathsf{coe}^{r\leadsto s}_{\lambda i.\mathsf{elim}_n^*[\phi(i)](\langle\!\langle A\rangle\!\rangle(i))} [\![a]\!] \hspace{2cm} \textbf{(7.6.2∗5)}
$$

$$
= \mathbf{coe}[\phi](\langle\!\langle A\rangle\!\rangle, r, s, [\![a]\!], \mathsf{coe}^{r\leadsto s}_{[\![A]\!]} [\![a]\!]) \hspace{1cm} \text{unfolding}
$$

$$
= \mathbf{coe}[\phi](\langle\!\langle A\rangle\!\rangle, r, s, [\![a]\!], [\![\mathsf{coe}^{r\leadsto s}_{\langle\uparrow_n^\diamond\rangle\circ A} a]\!]) \hspace{1cm} \text{folding} \hspace{1cm} \square
$$

(7.6.2∗8) *Elements of neutral types are stationary.* Let $A : \mathsf{netp}_n^\phi$ be a neutral type and let $a : \mathsf{ne}_\psi(\langle\uparrow_n^\diamond\rangle A) \wr_{\phi\wedge\psi} \mathsf{nf}(\langle\uparrow_n^\diamond\rangle A)$ be a stabilized neutral of type A. If A is stationary and $a.\mathsf{base}$ is stationary and $a.\mathsf{part}$ is stationary under $\phi \wedge \psi$, then $\widetilde{\mathbf{up}}_n[\phi,\psi](A,a)$ is also stationary.

Proof. We compute $\mathsf{nbe}_{\mathsf{tm}}(A,\widetilde{\mathbf{up}}_n[\phi,\psi](A,a))$.

$$
\mathsf{nbe}_{\mathsf{tm}}(A,\widetilde{\mathbf{up}}_n[\phi,\psi](A,a))
$$

$$
= \mathsf{nbe}_{\mathsf{tm}}(A,a) \hspace{3cm} \text{syntactic boundary}
$$

$$
= \wr_{[\![A]\!]} [\![a]\!] \hspace{3cm} \text{unfolding}
$$

$$
= \wr_{\mathsf{elim}_n^*[\phi](\langle\!\langle A\rangle\!\rangle)} \mathbf{up}_{\langle\!\langle A\rangle\!\rangle}(\langle\!\langle a\rangle\!\rangle) \hspace{1.5cm} \textbf{(7.6.2∗5)} \text{ and } \textbf{(7.6.2∗6)}
$$

$$
= \widetilde{\mathbf{up}}_n[\phi,\psi](\langle\!\langle A\rangle\!\rangle.\mathsf{base}, [\langle\!\langle a\rangle\!\rangle.\mathsf{base} \mid \phi\wedge\psi \hookrightarrow \wr_{\langle\!\langle A\rangle\!\rangle}\langle\!\langle a\rangle\!\rangle]) \hspace{0.5cm} \text{unfolding } \textbf{(7.3.4∗5)}
$$

$$
= \widetilde{\mathbf{up}}_n[\phi,\psi](A, [a \mid \phi\wedge\psi \hookrightarrow \wr_{[\![A]\!]}[\![a]\!]]) \hspace{1cm} \text{unfolding, } \phi\wedge\psi\text{-boundary}
$$

$$
= \widetilde{\mathbf{up}}_n[\phi,\psi](A, [a \mid \phi\wedge\psi \hookrightarrow \mathsf{nbe}_{\mathsf{tm}}(A,a)]) \hspace{1cm} \text{folding}
$$

$$
= \widetilde{\mathbf{up}}_n[\phi,\psi](A, [a \mid \phi\wedge\psi \hookrightarrow a]) \hspace{1cm} \text{assumption}
$$

$$
= \widetilde{\mathbf{up}}_n[\phi,\psi](A,a) \hspace{3cm} \eta\text{-contraction} \hspace{1cm} \square
$$

(7.6.2∗9) *Boundary-free type constructors are stationary.* Let $F = (A,B) : \mathsf{nffam}$ be a stationary family, in the sense that A is stationary and each $B(x)$ is stationary. Then $\mathbf{pi}_\diamond(F)$ is stationary in the same sense.

Proof. This follows by computation, fixing $F = (A,B)$.

$$
\mathsf{nbe}_{\mathsf{tp}}(\mathbf{pi}_\diamond(A,B))
$$

$$
= \mathsf{nbe}_{\mathsf{tp}}(\Pi(A,B)) \hspace{3cm} \text{syntactic boundary}
$$

$$
= \mathbf{pi}_\diamond(\mathsf{nbe}_{\mathsf{tp}}(A), \lambda x.\mathsf{nbe}_{\mathsf{tp}}(B(x))) \hspace{1cm} \text{unfolding } \textbf{(7.4.2∗2)}
$$

$$= \mathbf{pi}_\diamond(A,B) \qquad\qquad\qquad \text{assumption} \qquad\qquad \square$$

(7.6.2∗10) *Type constructors with boundaries are stationary too.* Fix $\psi : \mathbb{F}^*$, $B : \mathsf{nftp}$, $A : \{\psi\}\,\mathsf{nftp}$, and $f : \{\psi\}\,\mathsf{nf}(\mathsf{Equiv}(A,B))$ such that B is stationary and A,f are stationary under ψ. Then $\mathbf{glue}_\diamond(\psi,B,A,f)$ is stationary.

Proof. This follows in the same way as **(7.6.2∗9)**.

$$
\begin{aligned}
&\mathsf{nbe_{tp}}(\mathbf{glue}_\diamond(\psi,A,B,f)) \\
&\quad = \mathsf{nbe_{tp}}(\mathsf{glue}\,[\,B \mid \psi \hookrightarrow (A,f)\,]) && \text{syntactic boundary} \\
&\quad = \mathbf{glue}_\diamond(\psi,\mathsf{nbe_{tp}}(B),\mathsf{nbe_{tp}}(A),\mathsf{nbe_{tm}}(\mathsf{Equiv}(A,B),f)) && \text{unfolding } \textbf{(7.4.4∗4)} \\
&\quad = \mathbf{glue}_\diamond(\psi,B,A,f) && \text{assumption} \qquad \square
\end{aligned}
$$

(7.6.2∗11) *Function application is stationary.* Fix $f : \mathsf{ne}_\phi(\Pi(A,B))$ and $a : \mathsf{nf}(A)$ such that f and a are both stationary. Then $\mathbf{app}(f,a)$ is stationary.

Proof. By computation.

$$
\begin{aligned}
&\llbracket \mathbf{app}(f,a) \rrbracket \\
&\quad = \llbracket f(a) \rrbracket && \text{syntactic boundary} \\
&\quad = \llbracket f \rrbracket \llbracket a \rrbracket && \text{unfolding} \\
&\quad = (\natural^\phi_{\llbracket \Pi(A,B) \rrbracket}\langle\!\langle f \rangle\!\rangle)\llbracket a \rrbracket && \text{assumption} \\
&\quad = \natural^\phi_{\llbracket B \rrbracket \llbracket a \rrbracket}[\mathbf{app}(f,\natural_{\llbracket A \rrbracket}\llbracket a \rrbracket) \mid \phi \hookrightarrow \llbracket f \rrbracket\llbracket a \rrbracket] && \text{unfolding } \textbf{(7.4.2∗2)} \\
&\quad = \natural^\phi_{\llbracket B(a) \rrbracket}[\mathbf{app}(f,\mathsf{nbe_{tm}}A,a) \mid \phi \hookrightarrow \llbracket f(a) \rrbracket] && \text{folding} \\
&\quad = \natural^\phi_{\llbracket B(a) \rrbracket}[\mathbf{app}(f,a) \mid \phi \hookrightarrow \llbracket f(a) \rrbracket] && \text{assumption} \\
&\quad = \natural^\phi_{\llbracket B(a) \rrbracket}[\mathbf{app}(f,a) \mid \phi \hookrightarrow \llbracket \mathbf{app}(f,a) \rrbracket] && \text{syntactic boundary} \\
&\quad = \natural^\phi_{\llbracket B(a) \rrbracket}\langle\!\langle \mathbf{app}(f,a) \rangle\!\rangle && \text{folding} \qquad \square
\end{aligned}
$$

(7.6.2∗12) *Normal coercions are stationary.* Fix the formation data of a stuck coercion:

$$\phi : \mathbb{I} \to \mathbb{F}^* \qquad A : \prod_{i:\mathbb{I}}\mathsf{netp}_n^{\phi(i)} \qquad r,s : \mathbb{I} \qquad a : \mathsf{nf}(\langle\uparrow_n^\diamond\rangle A(r))$$

$$c : \{\forall i.\phi(i)\}\,\{\mathsf{nf}(\langle\uparrow_n^\diamond\rangle A(s)) \mid \bullet(r = s) \hookrightarrow [\P \hookrightarrow \mathsf{coe}^{r \rightsquigarrow s}_{\langle\uparrow_n^\diamond\rangle \circ A}a, r = s \hookrightarrow a]\}$$

Assume in addition that each $A(i)$ is stationary, a is stationary, and c is stationary underneath $\forall i.\phi(i)$. Then the stuck coercion $\mathbf{coe}_n[\phi](A,r,s,a,c)$ is also stationary.

Proof. By computation:

$$
\begin{aligned}
&\mathsf{nbe_{tm}}(\mathrm{A}(s),\mathbf{coe}_n[\phi](\mathrm{A},r,s,a,c))\\
&\quad = \mathsf{nbe_{tm}}\big(\mathrm{A}(s),\mathsf{coe}^{r\rightsquigarrow s}_{\langle\uparrow^\circ_n\rangle\circ\mathrm{A}}a\big) && \text{syntactic boundary}\\
&\quad = \textstyle\int_{[\![\mathrm{A}(s)]\!]}\big[\![\mathsf{coe}^{r\rightsquigarrow s}_{\langle\uparrow^\circ_n\rangle\circ\mathrm{A}}a]\!\big] && \text{unfolding}\\
&\quad = \textstyle\int_{\mathsf{elim}^*_n[\phi(s)](\langle\!\langle\mathrm{A}\rangle\!\rangle(s))}\mathbf{coe}[\phi]\big(\langle\!\langle\mathrm{A}\rangle\!\rangle,r,s,[\![a]\!],[\![\mathsf{coe}^{r\rightsquigarrow s}_{\langle\uparrow^\circ_n\rangle\circ\mathrm{A}}a]\!]\big) && \textbf{(7.6.2$*$5)}, \textbf{(7.6.2$*$7)}\\
&\quad = \mathbf{coe}_n[\phi]\big(\mathrm{A},r,s,\textstyle\int_{\mathsf{elim}^*_n[\phi(r)](\langle\!\langle\mathrm{A}\rangle\!\rangle(r))}[\![a]\!],\int_{[\![\mathrm{A}(s)]\!]}[\![\mathsf{coe}^{r\rightsquigarrow s}_{\langle\uparrow^\circ_n\rangle\circ\mathrm{A}}a]\!]\big) && \text{unfolding } \textbf{(7.3.4$*$5)}\\
&\quad = \mathbf{coe}_n[\phi]\big(\mathrm{A},r,s,\mathsf{nbe_{tm}}(\mathrm{A}(r),[\![a]\!]),\textstyle\int_{[\![\mathrm{A}(s)]\!]}[\![\mathsf{coe}^{r\rightsquigarrow s}_{\langle\uparrow^\circ_n\rangle\circ\mathrm{A}}a]\!]\big) && \text{folding}\\
&\quad = \mathbf{coe}_n[\phi]\big(\mathrm{A},r,s,a,\textstyle\int_{[\![\mathrm{A}(s)]\!]}[\![\mathsf{coe}^{r\rightsquigarrow s}_{\langle\uparrow^\circ_n\rangle\circ\mathrm{A}}a]\!]\big) && \text{assumption}\\
&\quad = \mathbf{coe}_n[\phi]\big(\mathrm{A},r,s,a,\textstyle\int_{[\![\mathrm{A}(s)]\!]}c\big) && \text{syntactic boundary}\\
&\quad = \mathbf{coe}_n[\phi]\big(\mathrm{A},r,s,a,\mathsf{nbe_{tm}}(\mathrm{A}(s),c)\big) && \text{folding}\\
&\quad = \mathbf{coe}_n[\phi](\mathrm{A},r,s,a,c) && \text{assumption} \qquad \square
\end{aligned}
$$

◼ **(7.6.2$*$13)** *Idempotence.* All normal forms of types, normal forms of terms, and neutral forms of terms are stationary in the sense of **(7.6.2$*$3)**.

Proof. By simultaneous induction on normals and neutrals, for which lemmas **(7.6.2$*$4)** and **(7.6.2$*$8)** through **(7.6.2$*$12)** exhibit representative cases. $\qquad\square$

◼ **(7.6.2$*$14)** *Normalization is an isomorphism.* We conclude from **(7.6.2$*$13)** that the normalization functions for types and for terms are surjective, just as in **(5.6.5$*$4)**. The normalization functions are already injective by soundness **(7.6.1$*$4)**, hence they are isomorphisms as in **(5.6.5$*$5)**.

§7.7. RECURSION-THEORETIC RESULTS

※ **(7.7$*$1)** Just as in §5.7, we can show that the normalization function is tracked by a recursive function and that judgmental equality for cubical type theory is hence decidable.

◼ **(7.7$*$2)** *The function* $\mathsf{nbe_{tp}}$ *is externally recursive.* By **(7.6.2$*$14)**, the (external) set of normal forms for a given type $\mathrm{A} : (\!|\Gamma|\!) \longrightarrow \mathsf{tp}$ has exactly one element. Because this set is also recursively enumerable, we have a terminating recursive function taking A to its normal form $\mathsf{nbe}^\Gamma_{\mathsf{tp}}(\mathrm{A}) : (\!|\Gamma|\!) \longrightarrow \mathsf{nftp}$.

◼ **(7.7$*$3)** *Decidability of judgmental equality.* It is effectively decidable whether two types $\mathrm{A},\mathrm{B} : (\!|\Gamma|\!) \longrightarrow \mathsf{tp}$ are equal, using the search algorithm **(7.7$*$2)** to obtain normal forms of A and B, and then comparing them. Equality of normal forms is externally decidable because the quotienting is relative to boundaries tracked by cofibrations, and the theory of cofibrations over the interval is externally decidable.

Part V

Prospects

CHAPTER 8

A PLAN FOR PROGRAMMING LANGUAGES

In this dissertation, we have necessarily restricted our attention to the applications of synthetic Tait computability to the syntactic metatheory of cubical dependent type theory, culminating in our proof of normalization (Chapter 7). On the other hand, we expect a deeper and more transformative marriage between synthetic Tait computability and programming languages than can be seen from the vantage point of pure type theory.

This "new plan" for programming languages locates its point of departure in the *phase distinction*, first conceptualized by Harper, Mitchell, and Moggi [HMM90] and Moggi [Mog89] as an abstraction of the noninterference between static and dynamic code in ML languages, and very recently identified by Sterling and Harper [SH21] with the Artin gluing of open/closed partitions of generalized spaces. The contribution of *op. cit.* was to observe that the relationship between syntax and semantics in logical relations arguments has identical formal properties to the relationship between static and dynamic code, and that several other widely appreciated PL constructs can be profitably expressed in the same way. The purpose of this chapter is to set out an agenda for the next several years of research into programming languages *vis-à-vis* the phase distinction.

§8.1. TWO PHASE DISTINCTIONS FOR PROGRAM MODULES

§8.1.1. The static–dynamic phase distinction

(8.1.1∗1) The origins of synthetic Tait computability lie in the investigation by Sterling and Harper [SH21] of the static–dynamic phase distinction for program modules in the presence of *weak structure sharing* à la SML '97 [Mil+97]. In ML-style module systems, one has a distinction between the *static part* of a module (its type components) and the *dynamic part* (its value or program components); our understanding of the static–dynamic geometry of ML modules was facilitated by the following three observations:

1) The static part of a module is independent of its dynamic part.
2) The dynamic part of a module depends on its static part.
3) The static part of a module function space is again another function space.

The phase-splitting interpretation of modules by Harper, Mitchell, and Moggi [HMM90] makes this relationship quite clear; every module signature splits into a dependent sum $[u : \mathrm{K} \,;\, \mathrm{T}(u)]$ where $\vdash \mathrm{K} : \mathbf{kind}$ is a classifier from the static sublanguage and $u : \mathrm{K} \vdash \mathrm{T}(u) : \mathbf{type}$ is a *dependent* classifier from the dynamic sublanguage. A function from $[u : \mathrm{K} \,;\, \mathrm{T}(u)]$ to $[u : \mathrm{K}' \,;\, \mathrm{T}'(u)]$ splits into a pair of a static function $f : \mathrm{K} \to \mathrm{K}'$ and a dynamic *dependent* function $g : \prod_{u:\mathrm{K}} \mathrm{T}(u) \to \mathrm{T}'(f(u))$.

(8.1.1∗2) *The phase distinction as Artin gluing.* In the extreme case that the static and dynamic sublanguages are the same language $\mathcal{T}$, the phase-splitting interpretation of modules is nothing more than the unary parametricity-translation of dependent type theory, in which a type is interpreted as a family of types. Every type theory $\mathcal{T}$ then can be equipped with a module language $\mathcal{T}^{\to}$ embedding two (static, dynamic) copies of $\mathcal{T}$ that can be projected by means of the codomain and domain functors $\mathcal{T}^{\to} \to \mathcal{T}$ respectively.

More realistically, the static sublanguage may be a restriction of the dynamic sublanguage; then one expects an embedding $\iota : \mathcal{T}_{st} \hookrightarrow \mathcal{T}_{dyn}$; reverse-engineering the phase-splitting interpretation of modules, we obtain a new module language in the *comma category* or *Artin gluing* $\mathcal{M} := \{\mathcal{T}_{dyn}\} \downarrow \iota$ obtained like so:

$$
\begin{array}{ccc}
\mathcal{M} & \dashrightarrow & \mathcal{T}_{dyn}^{\to} \\
\big\downarrow{\scriptstyle static\ projection} & \lrcorner & \big\downarrow{\mathrm{cod}} \\
\mathcal{T}_{st} & \xhookrightarrow{\ \iota\ } & \mathcal{T}_{dyn}
\end{array}
$$

An object of the category of signatures and modules $\mathcal{M}$ is given by an object $\mathrm{K} : \mathcal{T}_{st}$ together with an object $\mathrm{T} : (\mathcal{T}_{dyn})_{/\iota(\mathrm{K})}$; in type theoretic notation, this would be a static classifier $\vdash \mathrm{K} : \mathbf{kind}$ together with a dependent dynamic classifier $u : \iota(\mathrm{K}) \vdash \mathrm{T}(u) : \mathbf{type}$,

just as before. A morphism $(K,T) \longrightarrow (K',T')$ is given by a morphism $f : K \longrightarrow K'$ in $\mathcal{T}_{st}$ together with a morphism $T \longrightarrow f^*T'$ in the slice $(\mathcal{T}_{dyn})_{/\iota(K')}$, *i.e.* a dependent function $u : \iota(K) \vdash T(u) \to T'(f(u))$.

◻ **(8.1.1∗3)** *Static projection as weakening.* Suppose that the dynamic sublanguage $\mathcal{T}_{dyn}$ has an *empty* type $\vdash \varnothing :$ **type** satisfying the universal property of a strict initial object, *i.e.* the slice $(\mathcal{T}_{dyn})_{/\varnothing}$ is equivalent to the terminal category. Then we can define a special module $\S : \mathcal{M}$ with the remarkable property that $\mathcal{T}_{st} \simeq \mathcal{M}_{/\S}$ and that under this identification, the static projection is simply the weakening functor $\S^* : \mathcal{M} \longrightarrow \mathcal{M}_{/\S}$. This surprising property means that in a syntactic calculus for ML modules we would be able to "extract" the static part of any module by simply adding a variable $u : \S$ to the context; moreover, the static equivalence $\Gamma \vdash M \equiv_{st} N : S$ of Dreyer, Crary, and Harper [DCH03] can be tested by checking the ordinary equivalence $\Gamma, u : \S \vdash M \equiv N : S$ in an extended context.

The special module $\S$, which we call the *static open*, can be defined like so:

$$\S := [u : \mathbf{1} \,;\, \varnothing] \hspace{3cm} \text{(static open)}$$

It is not difficult to identify the slice $\mathcal{M}_{/\S}$ with the static sublanguage $\mathcal{T}_{st}$, because the presence of the empty type in the dynamic component has the effect of trivializing the dynamic components of any signature or module in its presence. For instance, we immediately have $\Gamma, u : \S \vdash [u : K \,;\, T(u)] \cong [u : K \,;\, \mathbf{1}]$ and therefore also also $\Gamma, u : \S \vdash [f \,;\, g] \equiv [f \,;\, g'] : [u : K \,;\, T(u)]$.

※ **(8.1.1∗4)** *The synthetic phase distinction.* The reformulation of static projection in terms of weakening by a special module **(8.1.1∗3)** has immediately profitable implications for the design of languages that evince phase distinctions. Rather than working explicitly with the (somewhat intricate) phase-splitting interpretation as we have above, we may simply consider type theories $\mathcal{T}$ extended by an *abstract* type $\S$ that is proof-irrelevant in the sense that $x, y : \S \vdash x \equiv y : \S$.

This abstract type $\S$ then *generates* a phase distinction in $\mathcal{T}$. The static part of a type T can be accessed *synthetically* as the function space $\bigcirc T := (\S \to T)$, and the dynamic part is the join $\bullet T := \S \vee T$ constructed as a quotient inductive type below:

$$\textbf{data } \S \vee T : \textbf{type where}$$
$$\textsf{inl} : \S \to \S \vee T$$
$$\textsf{inr} : T \to \S \vee T$$
$$\textsf{glue} : \textstyle\prod_{x:\S, y:T} \textsf{inl}(x) \equiv \textsf{inr}(y)$$

Both $\bigcirc, \bullet$ are idempotent monadic modalities in $\mathcal{T}$, with units $\eta_\bigcirc(x) = \lambda\_ : \S.x$ and $\eta_\bullet(x) = \textsf{inr}(x)$. The phase-splitting interpretation can be recovered in the synthetic setting by the Artin's fracture theorem [AGV72], which states that any type T is isomorphic to a dependent sum built up from its static and dynamic parts:

$$T \cong \textstyle\sum_{x:\bigcirc T} \{y : \bullet T \mid \textsf{map}_\bullet(\eta_\bigcirc, y) \equiv_{\bullet\bigcirc T} \eta_\bullet(x)\} \hspace{2cm} \text{(fracture theorem)}$$

The fracture theorem is therefore a synthetic version of the phase-splitting interpretation; it proves that nothing is lost by passing to the synthetic phase distinction. On the other hand, something quite significant is gained by adopting the synthetic approach: it becomes very obvious how to accommodate an entire *lattice* of distinct phase distinctions simultaneously in a modular way without incurring a blow-up in complexity. A major contribution of Sterling and Harper [SH21] was to overlay the *syntactic–semantic* phase distinction atop the existing static–dynamic phase distinction in order to prove representation independence results for ML modules.

§8.1.2. A phase distinction for Reynolds' relational parametricity

☯ **(8.1.2∗1)** In this dissertation we have mainly emphasized a phase distinction between syntactic and semantic. A modification of this idea was employed by Sterling and Harper [SH21] in which one has a notion of *semantic part* that depends not on a single syntactic part, but on two of them! The resulting phase distinction captures the *heterogeneous binary logical relations* that are typical of Reynolds' relational parametricity [Rey83].

Rather than extending the type theory by a single abstract proposition ¶, we instead add two disjoint propositions $\P_L, \P_R$ with $\P_L \wedge \P_R = \bot$ signifying the "left" and "right" copies of syntax respectively. Then the disjunction $\P_{LR} \coloneqq \P_L \vee \P_R$ allows one to work with both copies of the syntax simultaneously, and the corresponding fracture theorem expresses the fact that every type is a semantic (proof-relevant) relation between its left-syntactic and right-syntactic parts.

※ **(8.1.2∗2)** *Building correspondences synthetically.* We will write $\bigcirc_q T \coloneqq (\P_q \to T)$ and $\bullet_q T \coloneqq \P_q \vee T$ for the modalities corresponding to $q \in \{L,R,LR\}$. We furthermore have subuniverses $\mathcal{U}_{/q}$ and $\mathcal{U}_{\backslash q}$ classifying types for which the units $\eta_{\bigcirc_q}$ and $\eta_{\bullet_q}$ respectively are isomorphisms. Given a pair of types $A_L : \mathcal{U}_{/L}, A_R : \mathcal{U}_{/R}$ and a (proof-relevant) relation $S : A_L \times A_R \to \mathcal{U}_{\backslash LR}$, we may piece together a single type $\tilde{S} : \mathcal{U}$ such that $\bigcirc_L(\tilde{S} \cong A_L)$ and $\bigcirc_R(\tilde{S} \cong A_R)$.

$$\tilde{S} \coloneqq \sum_{x_L:A_L,x_R:A_R} S(x_L,x_R)$$

◆ **(8.1.2∗3)** *A representation invariant for queues.* Above we might think of A_L, A_R as two different representations for an abstract type, with S a representation invariant. For instance A_L might be the type of list-queues and A_R might be the type of batch-queues, and $S(x_L, x_R)$ would be defined to be the invariant that x_L and x_R represent the same queue.

$$A_L \coloneqq \mathsf{ListQueue.t}$$
$$A_R \coloneqq \mathsf{BatchQueue.t}$$
$$S(x_L, x_R) \coloneqq (x_L = x_R.1 \mathbin{@} \mathsf{rev}(x_R.2))$$

Showing that S is a structure-respecting correspondence between implementations of the queue abstract data type is as simple as defining a third queue implementation ListBatchQueue where $\tilde{S} := \sum_{x_L,x_R} S(x_L,x_R)$ is the representation type, such that all queue operations restrict under $\bigcirc_L, \bigcirc_R$ to the ones given for A_L, A_R respectively.

※ **(8.1.2∗4)** *Relational action of type constructors.* The essence of Reynolds' relational parametricity is that it provides a way to substitute a relation for a type in the interpretation, *i.e.* every family of types or terms carries a relational action. In the synthetic setting, the relational action is given by ordinary substitution; given a term $X : \mathcal{U} \vdash b(X) : B$ and a relation (A_L, A_R, S) as above, the relational action of $b(X)$ is nothing more than the substitution instance $b(\sum_{x_L,x_R} S(x_L,x_R))$.

◆ **(8.1.2∗5)** *Observational equivalence of list and batch queues.* Continuing from our example **(8.1.2∗3)**, we may deduce a representation independence theorem: if $x : \mathsf{QUEUE} \vdash b(x) : \mathsf{bool}$ is a syntactic context, then we have $b(\mathsf{ListQueue}) = b(\mathsf{BatchQueue})$.

Proof. We consider the boolean $b(\mathsf{ListBatchQueue})$ whose left-syntactic part is $b(\mathsf{ListQueue})$ and whose right-syntactic part is $b(\mathsf{BatchQueue})$. But for any boolean, either both the left and right syntactic parts are true or they are both false; therefore either $b(\mathsf{ListQueue})$ and $b(\mathsf{BatchQueue})$ are both true or they are both false.   □

§8.2. TYPE REFINEMENTS AND PROGRAM EXTRACTION

※ **(8.2∗1)** What's the difference between a refinement type and a subtype? Sometimes conflated, these tools have very different semantics. In particular, it is not necessary that a function $\{q : \mathbb{Q} \mid q > 0\} \to \{q : \mathbb{Q} \mid q > 0\}$ have a value at $z = 0$, whereas an element of the *refined* function type $[q : \mathbb{Q} \mid q > 0] \to [q : \mathbb{Q} \mid q > 0]$ carries within it the data of an unrefined function $\mathbb{Q} \to \mathbb{Q}$ that has the additional property of taking positive rationals to positive rationals. Specializing the perspective of Melliès and Zeilberger [MZ15] of functors as type refinement systems, we find that synthetic Tait computability also leads to a modular and proof-relevant version of type refinements.

◔ **(8.2∗2)** *A phase distinction between program and specification.* Consider the extension of a type theory by a single abstract proposition ¶. Then it is possible to view the modality $\bigcirc$ as isolating the *program-level* part of a construction, whereas the ● isolates the *specification-level* part of the construction. The fact that $\bigcirc ● A \cong \mathbf{1}$ corresponds to the way that extracted programs are guaranteed not to contain any specification-level data or proofs.

◆ **(8.2∗3)** *The refinement of positive rationals.* We may define the positive refinement for rationals using the modalities of synthetic Tait computability as the dependent sum $\sum_{q:\mathbb{Q}} ●(q > 0)$ assuming that $\mathbb{Q} : \mathcal{U}_{/¶}$ is the computational type of rational numbers.

■ **(8.2∗4)** *A general refinement type.* Letting $A : \mathcal{U}_{/\P}$ be a computational type and $B : A \to \mathcal{U}_{\backslash\P}$ be a family of specification-level types indexed in A, we can define the *refinement* type $[x : A \mid B(x)]$ as the $\sum_{x:A} B(x)$. In fact, by *realignment* **(3.3∗3)** we may construct the refinement type in such a way that the following equation holds judgmentally:

$$z : \P \vdash [x : A \mid B(x)] \equiv A \qquad\qquad (8.2{*}4{*}1)$$

Given $a : A$ and $b : B(a)$, we will write $[a \, ; b] : [x : A \mid B(x)]$ for the corresponding refined element. Given $p : [x : A \mid B(x)]$, we note that we also have $p : A$ from Eq. (8.2∗4∗1) and the fact that $A : \mathcal{U}_{/\P}$; we will write $p.\mathsf{spec} : B(p)$ for the specification part of p.

■ **(8.2∗5)** *The refinement relation.* Eq. (8.2∗4∗1) suggests a *definition* of refinement. Let $P : \mathcal{U}$ be an arbitrary type and let $A : \mathcal{U}_{/\P}$ be a computational type. We say that P *refines* A, written $P \sqsubset A$, when $z : \P \vdash P \cong A$. In other words, the computational part of P is A.

◆ **(8.2∗6)** *Variance of subtyping and refinement.* Another significant difference between subtyping and refinement is that they obey very different laws concerning the function type. The subtyping law for the function type is contravariant in the domain, whereas the refinement law for the function type can be *proved* to be covariant in the domain following our definition **(8.2∗5)**:

$$\frac{A \leq A' \qquad B' \leq B}{(A' \to B') \leq (A \to B)} \qquad\qquad \frac{A' \sqsubset A \qquad B' \sqsubset B}{(A' \to B') \sqsubset (A \to B)}$$

◆ **(8.2∗7)** *Extracting the underlying program.* Let $\mathbb{Q}^+ := [x : \mathbb{Q} \mid \bullet(x > 0)] \sqsubset \mathbb{Q}$ be the refinement of positive rational numbers. We observe that any function $f : \mathbb{Q}^+ \to \mathbb{Q}^+$ carries with it the data of function $f_\circ : \mathbb{Q} \to \mathbb{Q}$:

$$\mathbb{Q} \xrightarrow{\cong} \circ\mathbb{Q}^+ \xrightarrow{\circ f} \circ\mathbb{Q}^+ \xrightarrow{\cong} \mathbb{Q}$$

$$f_\circ$$

We also have the following element witnessing that $f_\circ$ satisfies its specification:

$$f' : \textstyle\prod_{x:\mathbb{Q}} \bullet(x > 0) \to \bullet(f_\circ(x) > 0)$$
$$f' := \lambda x, p.(f[x \, ; p]).\mathsf{spec}$$

※ **(8.2∗8)** *Proof-relevance of refinements.* In proof assistants like Nuprl and Coq [Con+86; Coq16], squash/propositional truncation types $\|-\|$ are used to control extraction, preventing specification-level data from appearing in extracted programs. This has been used to great effect in both Nuprl and Coq to extract efficient code from inefficient proofs. A significant limitation of this approach is that in the absence of the axiom of choice, we will

not usually be able to reconstruct a squashed function $\|A \to B\|$ from an assumption like $A \to \|B\|$; therefore, using proof-irrelevance to hide things from extraction can in some cases obstruct important specification-level constructions. The specificational modality $\bullet$ provides a different and more practical way to hide data from extraction without destroying it. If $A : \mathcal{U}$ is an arbitrary type and $B : \mathcal{U}_{\backslash \P}$ is a specification-level type, we have the following identifications:

$$(A \to B) \cong (A \to \bullet B) \cong (\bullet A \to \bullet B) \cong \bullet(A \to B)$$

§8.3. INFORMATION-FLOW AND NONINTERFERENCE

※ **(8.3∗1)** Both the static–dynamic phase distinction (§8.1.1) and the syntactic–semantic phase distinction (this dissertation) express a noninterference property: the (static, syntactic) part of a function is independent of the (dynamic, semantic) part of its inputs; to put it more directly, every function $\bullet A \longrightarrow \bigcirc B$ is constant. The noninterference between the open and closed modalities suggests that one may profitably rephrase information flow calculi such as Abadi, Banerjee, Heintze, and Riecke's dependency core calculus [Aba+99] in terms of a lattice of phase distinctions (open-closed partitions). Then the redaction of information at a particular security level would correspond to a particular closed modality.

▨ **(8.3∗2)** Let $(\mathbb{L}, \sqsubseteq)$ be a lattice of "security levels", where $l \sqsubseteq k$ means that l is a lower security level than k. Any topos equipped with the structure of a *filter* on $\mathbb{L}$ can interpret a modal type theory for information-flow validating standard non-interference properties. A filter on $\mathbb{L}$ over a topos **E** is given by an upward-closed and downward-directed family of opens $[l] \in \mathcal{O}_{\mathbf{E}}$, in other words a finite meet-preserving homomorphism of lattices $\mathbb{L} \longrightarrow \mathcal{O}_{\mathbf{E}}$.

☯ **(8.3∗3)** Working abstractly with such a filter is the same as working concretely in the presheaf category $\mathrm{Pr}(\mathbb{L})$, a consequence of Diaconescu's theorem [Dia75].

▨ **(8.3∗4)** Viewed from the perspective of the category of sheaves $\mathbf{Set_E}$, we can phrase the structure of a filter in type theoretic language:[1]

SECURITY LEVEL	UPWARD-CLOSED	NONEMPTY	BOUNDED
$\dfrac{l \in \mathbb{L}}{[l] : \Omega}$	$\dfrac{l,k \in \mathbb{L} \qquad l \leq k}{[l] \vdash [k]}$	$\dfrac{}{\vdash \bigvee_{l \in \mathbb{L}}[l]}$	$\dfrac{l,k \in \mathbb{L}}{[l] \wedge [k] \vdash [l \sqcap k]}$

◆ **(8.3∗5)** Consider the case of the lattice $\{\mathsf{low} \sqsubset \mathsf{med} \sqsubset \mathsf{high}\}$. Then we have functions $\bigcirc_{[\mathsf{high}]}A \to \bigcirc_{[\mathsf{low}]}A$ corresponding to the fact that low-security data is visible with a high security clearance. On the other hand, the closed modalities may be used to *seal* information behind a security clearance using the derived closed modality $\blacklozenge_l A \coloneqq \bullet_{(\bigvee_{k \sqsubset l}[k])}A$.

[1] If $\mathbb{L}$ is totally ordered, then the BOUNDED rule is redundant

§8.3.1. A core calculus for dependency

(8.3.1∗1) Abadi, Banerjee, Heintze, and Riecke [Aba+99] introduced a variant of Moggi's computational λ-calculus [Mog91] equipped with a family of idempotent monadic modalities $T_l(A)$ indexed in security levels $l \in \mathbb{L}$. Abadi, Banerjee, Heintze, and Riecke incorporates an auxiliary form of judgment $\boxed{\text{A protected } @ \, l}$ to mean that A is redacted below security level l, *i.e.* the unit $A \longrightarrow T_l(A)$ is an isomorphism. The idempotence of Abadi, Banerjee, Heintze, and Riecke's modalities is seen in the following *bind* rule:

$$\frac{\Gamma \vdash M : T_l(A) \qquad \Gamma, x : A \vdash N(x) : B \qquad B \text{ protected } @ \, l}{\Gamma \vdash \text{bind } x \leftarrow M \text{ in } N(x) : B}$$

(8.3.1∗2) We may interpret Abadi, Banerjee, Heintze, and Riecke's $\boxed{\text{A protected } @ \, l}$ judgment in our setting as the assertion that A is $\bigvee_{k \sqsubset l}[k]$-connected; then the modality $T_l(A)$ is interpreted as $\blacklozenge_l A$ from **(8.3∗5)**. What's more, the data that is visible from a given security level $k \in \mathbb{L}$ may be extracted by considering the slice over $[k]$, *i.e.* applying the open modality $\bigcirc_{[k]} := ([k] \to -)$. It is easy to see that if $k \sqsubset l$, then $\bigcirc_{[k]} \blacklozenge_l A \cong \mathbf{1}$, justifying our intuition for $\blacklozenge$ as redaction.

§8.3.2. Adequacy of topo-logical semantics

(8.3.2∗1) Based on §8.3.1, we show that any topos equipped with a filter on $\mathbb{L}$ gives rise to a sheaf model of the (total) dependency core calculus [Aba+99]. Let $\mathbf{E}$ be a topos, and let $[-] : \mathbb{L} \longrightarrow \mathcal{O}_{\mathbf{E}}$ be a filter on $\mathbb{L}$ over $\mathbf{E}$. Contexts and types are interpreted as sheaves $A : \mathbf{Set}_{\mathbf{E}}$; we say that such a sheaf is *protected* at level $l \in \mathbb{L}$ when the canonical map $A \longrightarrow \blacklozenge_l A$ is an isomorphism.

$$[\![\mathsf{bool}]\!] = \mathbf{1} + \mathbf{1}$$
$$[\![A \to B]\!] = [\![A]\!] \to [\![B]\!]$$
$$[\![T_l(A)]\!] = \blacklozenge_l [\![A]\!]$$

We observe by induction that if $\vdash B$ protected $@ \, l$, then the sheaf $[\![B]\!]$ is $\blacklozenge_l$-modal. The interpretation of DCC's terms is trivial, except that of the **bind** operator. We assume morphisms $[\![\Gamma]\!] \longrightarrow \blacklozenge_l[\![A]\!]$ and $[\![\Gamma]\!] \times [\![A]\!] \longrightarrow [\![B]\!]$ such that $[\![B]\!]$ is protected at level l, and we must produce an element of $[\![B]\!]$. But we have assumed that $[\![B]\!] \cong \blacklozenge_l[\![B]\!]$, so we may employ the ordinary Kleisli extension from $\blacklozenge_l$.

(8.3.2∗2) A *filter model* of DCC is a pair $(\mathbf{E}, [-])$ where $\mathbf{E}$ is a topos and $[-] : \mathbb{L} \longrightarrow \mathcal{O}_{\mathbf{E}}$ is a filter on $\mathbb{L}$ over $\mathbf{E}$. We call a filter model *consistent* when $\mathbf{E}$ is not the empty topos, *i.e.* $\mathbf{Set}_{\mathbf{E}}$ is not the terminal category.

(8.3.2∗3) *Observational equivalence.* Two closed terms $\cdot \vdash a, b : A$ are called *observationally equivalent*, written $a \simeq b$, when for any $x : A \vdash c(x) : T_l(\mathsf{bool})$ we have $\cdot \vdash c(a) = c(b)$.

∎ **(8.3.2∗4)** *Canonicity.* Let $\cdot \vdash a : \mathsf{T}_l(\mathsf{bool})$ be a closed term; then either $a = \mathsf{ret}(\mathsf{tt})$ or $a = \mathsf{ret}(\mathsf{ff})$.

Proof. Letting $\mathcal{C}$ be the syntactic category of DCC, we instantiate synthetic Tait computability by using the *initial* point $\mathbf{0}_{\widehat{\mathcal{C}}} : \mathbb{1} \to \widehat{\mathcal{C}}$ as a figure shape to glue along; by adjointness, this is equivalent to forming the Sierpiński cone of $\widehat{\mathcal{C}}$.

$$
\begin{array}{ccc}
\mathbb{1} & \xrightarrow{\;\mathbf{0}_{\widehat{\mathcal{C}}}\;} & \widehat{\mathcal{C}} \\
{\scriptstyle \circ}\downarrow & & \downarrow \\
\mathbb{S} & \xrightarrow{\quad} & \mathbf{G}
\end{array}
\qquad\qquad
\begin{array}{ccc}
\widehat{\mathcal{C}} & \xrightarrow{\;!_{\widehat{\mathcal{C}}}\;} & \mathbb{1} \\
{\scriptstyle \widehat{\mathcal{C}}\times\bullet}\downarrow & & \downarrow \\
\widehat{\mathcal{C}} \times \mathbb{S} & \xrightarrow{\quad} & \mathbf{G}
\end{array}
$$

Then we may reproduce $\widehat{\mathcal{C}}$ as the open subtopos $\mathbf{G}_{\P}$ for some $\P \in \mathcal{O}_{\mathbf{G}}$. The direct image of the open immersion $\widehat{\mathcal{C}} \hookrightarrow \mathbf{G}$ embeds the generic model of DCC into $\mathbf{Set}_{\mathbf{G}}$. We define a new model of DCC in the STC metalanguage of $\mathbf{Set}_{\mathbf{G}}$ that restricts to the generic model on $\mathbf{G}_{\P} \simeq \widehat{\mathcal{C}}$. In defining the model, there are only two degrees of freedom: the computability structure of the booleans and of the monad, which we interpret so as to ensure that every closed computation of boolean type is pure and returns either tt or ff:

$$\llbracket \mathsf{bool} \rrbracket \cong \textstyle\sum_{x:\mathsf{bool}} \bullet (x = \mathsf{tt} + x = \mathsf{ff})$$
$$\llbracket \mathsf{T}_l(A) \rrbracket \cong \textstyle\sum_{x:\mathsf{T}_l(A)} \bullet \{a : \llbracket A \rrbracket \mid \bigcirc(x = \mathsf{ret}(a))\}$$

Under these interpretations, a closed term $\cdot \vdash a : \mathsf{T}_l(\mathsf{bool})$ corresponds in the STC model to a morphism $\mathbf{1} \to \llbracket \mathsf{T}_l(\mathsf{bool}) \rrbracket$ that restricts to a over $\widehat{\mathcal{C}}$. Unfolding further, this is exactly a witness that $a \in \{\mathsf{ret}(\mathsf{tt}), \mathsf{ret}(\mathsf{ff})\}$. $\qquad\square$

∎ **(8.3.2∗5)** *Adequacy of filter models.* Let $(\mathbf{E}, [-])$ be a consistent filter model of DCC. Then for any $\cdot \vdash a, b : A$, we have $a \simeq b$ if $\llbracket a \rrbracket = \llbracket b \rrbracket$.

Proof. Fix a level $l \in \mathbb{L}$ and $x : A \vdash c(x) : \mathsf{T}_l(\mathsf{bool})$; we need to show that $c(a) = c(b)$. Suppose to the contrary that $c(a) \neq c(b)$; by **(8.3.2∗4)**, we may assume $c(a) = \mathsf{ret}(\mathsf{tt})$ and $c(b) = \mathsf{ret}(\mathsf{ff})$ without loss of generality. Therefore we have $\llbracket c(a) \rrbracket = \eta_{\blacklozenge_l}(\mathsf{inl}(*))$ and $\llbracket c(b) \rrbracket = \eta_{\blacklozenge_l}(\mathsf{inr}(*))$. Letting $\mathrm{U} \in \mathcal{O}_{\mathbf{E}}$ be the open $\bigvee_{k<l}[k]$, we note that $\llbracket c(a) \rrbracket \neq \llbracket c(b) \rrbracket$ because these two elements restrict along the closed immersion $\mathbf{E}_{\setminus \mathrm{U}} \hookrightarrow \mathbf{E}$ to the two disjoint elements of the coproduct $(\setminus \mathrm{U})^* \blacklozenge_l (1 + 1) \cong (\setminus \mathrm{U})^* (\setminus \mathrm{U})_* (\setminus \mathrm{U})^* (1 + 1) \cong (\setminus \mathrm{U})^* (1 + 1) \cong (1 + 1)$. From our assumption, we have $\llbracket c(a) \rrbracket = \llbracket c \rrbracket \llbracket a \rrbracket = \llbracket c \rrbracket \llbracket b \rrbracket = \llbracket c(b) \rrbracket$, a contradiction. $\qquad\square$

∎ **(8.3.2∗6)** *Noninterference.* Let $x : \mathsf{T}_l(A) \vdash e(x) : \mathsf{T}_{l'}(B)$ such that $l' \sqsubset l$. Then for all closed terms $a, b : \mathsf{T}_l(A)$, we have the observational equivalence $e(a) \simeq e(b)$.

Proof. Choose an arbitrary consistent filter model $(\mathbf{E}, [-])$; for instance, one could choose the classifying topos $\widehat{\mathbb{L}}$ of $\mathbb{L}$-filters and its generic filter $\mathsf{y}_{\mathbb{L}}$. Letting $U \in \mathcal{O}_{\mathbf{E}}$ be the open $\bigvee_{k \sqsubset l}[k]$, we have an open subtopos $\mathbf{E}_U \subseteq \mathbf{E}$ and by restriction a new filter $[-]_U : \mathbb{L} \longrightarrow \mathcal{O}_{\mathbf{E}_U}$. By **(8.3.2∗5)**, it suffices to check that $e(a)$ and $e(b)$ have equal interpretations in the filter model $(\mathbf{E}_U, [-]_U)$. We observe that $[\![a]\!] = [\![b]\!]$ because $[\![\mathsf{T}_l(A)]\!] = \blacklozenge_l[\![A]\!] \cong \mathbf{1}$, observing that $\bigvee_{k \sqsubset l}[k]_U = \top$ in $\mathcal{O}_{\mathbf{E}_U}$. $\qquad\qquad\square$

REFERENCES

[Aba+99]   Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. "A Core Calculus of Dependency". In: *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '99. San Antonio, Texas, USA: Association for Computing Machinery, 1999, pp. 147–160. ISBN: 1-58113-095-3. DOI: 10.1145/292540.292555 (cit. on pp. 183, 184).

[Abe09]   Andreas Abel. "Extensional normalization in the logical framework with proof irrelevant equality". In: *2009 Workshop on Normalization by Evaluation*. 2009 (cit. on pp. 26, 106).

[Abe13]   Andreas Abel. "Normalization by Evaluation: Dependent Types and Impredicativity". Habilitation. Ludwig-Maximilians-Universität München, 2013. URL: http://www2.tcs.ifi.lmu.de/~abel/habil.pdf (cit. on pp. 26, 106).

[AAD07]   Andreas Abel, Klaus Aehlig, and Peter Dybjer. "Normalization by Evaluation for Martin-Löf Type Theory with One Universe". In: *Electronic Notes in Theoretical Computer Science* 173 (Apr. 2007), pp. 17–39. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2007.02.025 (cit. on pp. 26, 106).

[ACP09]   Andreas Abel, Thierry Coquand, and Miguel Pagano. "A Modular Type-Checking Algorithm for Type Theory with Singleton Types and Proof Irrelevance". In: *Typed Lambda Calculi and Applications*. Ed. by Pierre-Louis Curien. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 5–19. ISBN: 978-3-642-02273-9 (cit. on p. 26).

[AÖV17]   Andreas Abel, Joakim Öhman, and Andrea Vezzosi. "Decidability of Conversion for Type Theory in Type Theory". In: *Proceedings of the ACM on Programming Languages* 2 (Dec. 2017), 23:1–23:29. ISSN: 2475-1421. DOI: 10.1145/3158111 (cit. on p. 32).

[AVW17]   Andreas Abel, Andrea Vezzosi, and Theo Winterhalter. "Normalization by Evaluation for Sized Dependent Types". In: *Proceedings of the ACM on Programming Languages* 1.ICFP (Aug. 2017), 33:1–33:30. ISSN: 2475-1421. DOI: 10.1145/3110277 (cit. on pp. 26, 106).

[Acz78]   Peter Aczel. *A General Church-Rosser Theorem*. Technical Report. University of Manchester, 1978 (cit. on p. 12).

[Ahm04]   Amal Jamil Ahmed. "Semantics of Types for Mutable State". PhD thesis. Princeton University, 2004. URL: http://www.cs.indiana.edu/~amal/ahmedsthesis.pdf (cit. on p. 9).

[AMB13]   Guillaume Allais, Conor McBride, and Pierre Boutillier. "New Equations for Neutral Terms: A Sound and Complete Decision Procedure, Formalized". In: *Proceedings of the 2013 ACM SIGPLAN Workshop on Dependently-typed Programming*. DTP '13. Boston, Massachusetts, USA: Association for Computing Machinery, 2013, pp. 13–24. ISBN: 978-1-4503-2384-0. DOI: 10.1145/2502409.2502411 (cit. on p. 106).

[Alt+01]   T. Altenkirch, P. Dybjer, M. Hofmann, and P. Scott. "Normalization by Evaluation for Typed Lambda Calculus with Coproducts". In: *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*. Washington, DC, USA: IEEE Computer Society, 2001. DOI: 10.1109/LICS.2001.932506 (cit. on pp. 106, 146).

[ADK17]   Thorsten Altenkirch, Nils Anders Danielsson, and Nicolai Kraus. "Partiality, Revisited: The Partiality Monad as a Quotient Inductive-Inductive Type". In: *Foundations of Software Science and Computation Structures*. Ed. by Javier Esparza and Andrzej S. Murawski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 534–549. ISBN: 978-3-662-54458-7. DOI: 10.1007/978-3-662-54458-7_31. arXiv: 1610.09254 [cs.LO] (cit. on p. 11).

[AHS95]   Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. "Categorical reconstruction of a reduction free normalization proof". In: *Category Theory and Computer Science*. Ed. by David Pitt, David E. Rydeheard, and Peter Johnstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 182–199. ISBN: 978-3-540-44661-3. DOI: 10.1007/3-540-60164-3_27 (cit. on pp. 105, 108).

[AK16a]   Thorsten Altenkirch and Ambrus Kaposi. "Normalisation by Evaluation for Dependent Types". In: *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*. Ed. by Delia Kesner and Brigitte Pientka. Vol. 52. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 6:1–6:16. ISBN: 978-3-95977-010-1. DOI: 10.4230/LIPIcs.FSCD.2016.6. URL: http://drops.dagstuhl.de/opus/volltexte/2016/5972 (cit. on pp. 15, 31).

[AK16b]   Thorsten Altenkirch and Ambrus Kaposi. "Type Theory in Type Theory Using Quotient Inductive Types". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* POPL '16. St. Petersburg, FL, USA: Association for Computing Machinery, 2016, pp. 18–29. ISBN: 978-1-4503-3549-2. DOI: `10.1145/2837614.2837638` (cit. on pp. 13, 15).

[AJ21]   Mathieu Anel and André Joyal. "Topo-logie". In: *New Spaces in Mathematics: Formal and Conceptual Reflections.* Ed. by Mathieu Anel and Gabriel Catren. Vol. 1. Cambridge University Press, 2021. Chap. 4, pp. 155–257. DOI: `10.1017/9781108854429.007` (cit. on pp. 51, 52).

[AL20]   Mathieu Anel and Damien Lejay. *Exponentiable ∞-topoi.* Draft, version 2. 2020. URL: `http://mathieu.anel.free.fr/mat/doc/Anel-Lejay-Exponentiable-topoi.pdf` (cit. on p. 64).

[Ang19]   Carlo Angiuli. "Computational Semantics of Cartesian Cubical Type Theory". PhD thesis. Carnegie Mellon University, 2019 (cit. on pp. 30, 142, 143).

[Ang+19]   Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. *Syntax and Models of Cartesian Cubical Type Theory.* Preprint. Feb. 2019. URL: `https://github.com/dlicata335/cart-cube` (cit. on pp. 25, 133, 137, 139, 140, 142).

[AH18]   Carlo Angiuli and Robert Harper. "Meaning explanations at higher dimension". In: *Indagationes Mathematicae* 29.1 (2018). L.E.J. Brouwer, fifty years later, pp. 135–149. ISSN: 0019-3577. DOI: `10.1016/j.indag.2017.07.010` (cit. on p. 5).

[AHH17]   Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. *Computational Higher Type Theory III: Univalent Universes and Exact Equality.* 2017. arXiv: `1712.01800 [cs.LO]`. URL: `https://arxiv.org/abs/1712.01800` (cit. on pp. 18, 26, 30, 142).

[AHH18]   Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. "Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities". In: *27th EACSL Annual Conference on Computer Science Logic (CSL 2018).* Ed. by Dan Ghica and Achim Jung. Vol. 119. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 6:1–6:17. ISBN: 978-3-95977-088-0. DOI: `10.4230/LIPIcs.CSL.2018.6`. URL: `http://drops.dagstuhl.de/opus/volltexte/2018/9673` (cit. on pp. 20, 22, 25, 137, 140–142).

[AGV72]   Michael Artin, Alexander Grothendieck, and Jean-Louis Verdier. *Théorie des topos et cohomologie étale des schémas*. Séminaire de Géométrie Algébrique du Bois-Marie 1963–1964 (SGA 4), Dirigé par M. Artin, A. Grothendieck, et J.-L. Verdier. Avec la collaboration de N. Bourbaki, P. Deligne et B. Saint-Donat, Lecture Notes in Mathematics, Vol. 269, 270, 305. Berlin: Springer-Verlag, 1972 (cit. on pp. 51, 57, 68, 69, 72, 100, 179).

[AM13]   Robert Atkey and Conor McBride. "Productive Coprogramming with Guarded Recursion". In: *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*. Boston, Massachusetts, USA: Association for Computing Machinery, 2013, pp. 197–208. ISBN: 978-1-4503-2326-0. DOI: `10.1145/2500365.2500597` (cit. on p. 62).

[Awo10]   Steve Awodey. *Category Theory*. 2nd. New York, NY, USA: Oxford University Press, Inc., 2010. ISBN: 978-0-19-923718-0 (cit. on p. 3).

[Awo15]   Steve Awodey. "Notes on cubical models of type theory". 2015. URL: `http://www.github.com/awodey/math/blob/master/Cubical/cubical.pdf` (cit. on p. 133).

[Awo18]   Steve Awodey. "Natural models of homotopy type theory". In: *Mathematical Structures in Computer Science* 28.2 (2018), pp. 241–286. DOI: `10.1017/S0960129516000268`. arXiv: `1406.3219 [math.CT]` (cit. on pp. 40, 41, 43).

[Awo21]   Steve Awodey. "A Quillen model structure on the category of cartesian cubical sets". Unpublished notes. 2021. URL: `https://github.com/awodey/math/blob/e8c715cc5cb6a966e736656bbe54d0483f9650fc/QMS/qms.pdf` (cit. on p. 71).

[AB00]   Steve Awodey and Carsten Butz. "Topological Completeness for Higher-Order Logic". In: *The Journal of Symbolic Logic* 65.3 (2000), pp. 1168–1182. ISSN: 00224812. arXiv: `math/9707206 [math.LO]`. URL: `http://www.jstor.org/stable/2586693` (cit. on p. 119).

[Bau06]   Andrej Bauer. "First Steps in Synthetic Computability Theory". In: *Electronic Notes in Theoretical Computer Science* 155 (2006). Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI), pp. 5–31. ISSN: 1571-0661. DOI: `10.1016/j.entcs.2005.11.049` (cit. on p. 95).

[BBS04]   Andrej Bauer, Lars Birkedal, and Dana S. Scott. "Equilogical spaces". In: *Theoretical Computer Science* 315.1 (2004). Mathematical Foundations of Programming Semantics, pp. 35–59. ISSN: 0304-3975. DOI: `10.1016/j.tcs.2003.11.012` (cit. on p. 8).

[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 fuer Informatik, 2014, pp. 107–128. ISBN: 978-3-939897-72-9. DOI: 10.4230/LIPIcs.TYPES.2013.107. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4628 (cit. on p. 21).

[Bic16]   Mark Bickford. *Constructive analysis and experimental mathematics using the Nuprl proof assistant.* 2016. URL: http://www.nuprl.org/documents/Bickford/reals.pdf (cit. on p. 17).

[Bic18]   Mark Bickford. *Formalizing Category Theory and Presheaf Models of Type Theory in Nuprl.* 2018. arXiv: 1806.06114 [cs.LO] (cit. on p. 6).

[Bic]   Mark Bickford. *Bishop and Bridges, Constructive Analysis, Chapter 2.* Nuprl formalization of Chapter 2 of *Constructive Analysis.* URL: http://www.nuprl.org/MathLibrary/ConstructiveAnalysis/Constructive_Analysis_Ch2.html (cit. on p. 17).

[Bir+16] Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. "Guarded Cubical Type Theory: Path Equality for Guarded Recursion". In: *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*. Ed. by Jean-Marc Talbot and Laurent Regnier. Vol. 62. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 23:1–23:17. ISBN: 978-3-95977-022-4. DOI: 10.4230/LIPIcs.CSL.2016.23 (cit. on pp. 9, 71, 74).

[Bir+11a] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. "First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees". In: *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science.* Washington, DC, USA: IEEE Computer Society, 2011, pp. 55–64. ISBN: 978-0-7695-4412-0. DOI: 10.1109/LICS.2011.16. arXiv: 1208.3596 [cs.LO] (cit. on pp. 9, 61, 95).

[Bir+11b] Lars Birkedal, Bernhard Reus, Jan Schwinghammer, Kristian Støvring, Jacob Thamsborg, and Hongseok Yang. "Step-Indexed Kripke Models over Recursive Worlds". In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* Austin, Texas, USA: Association for Computing Machinery, 2011, pp. 119–132. ISBN: 978-1-4503-0490-0. DOI: 10.1145/1926385.1926401 (cit. on p. 9).

[BST10]   Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. "Realisability semantics of parametric polymorphism, general references and recursive types". In: *Mathematical Structures in Computer Science* 20.4 (2010), pp. 655–703. DOI: 10.1017/S0960129510000162 (cit. on p. 9).

[Bis67]   Errett Bishop. *Foundations of Constructive Analysis*. New York: McGraw-Hill, 1967 (cit. on p. 18).

[Bis69]   Errett Bishop. "A General Language". Unpublished manuscript. 1969 (cit. on p. 5).

[BB85]   Errett Bishop and Douglas S. Bridges. *Constructive Analysis*. Springer-Verlag Berlin Heidelberg, 1985. ISBN: 978-3-642-64905-9. DOI: 10.1007/978-3-642-61667-9 (cit. on pp. 6, 16, 17).

[Biz16]   Aleš Bizjak. "On semantics and applications of guarded recursion". PhD thesis. University of Aarhus, 2016 (cit. on p. 9).

[BBM14]   Ales Bizjak, Lars Birkedal, and Marino Miculan. "A Model of Countable Nondeterminism in Guarded Type Theory". In: *Rewriting and Typed Lambda Calculi – Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*. Ed. by Gilles Dowek. Vol. 8560. Lecture Notes in Computer Science. Springer, 2014, pp. 108–123. ISBN: 978-3-319-08917-1. DOI: 10.1007/978-3-319-08918-8_8 (cit. on p. 9).

[BB18]   Aleš Bizjak and Lars Birkedal. "On Models of Higher-Order Separation Logic". In: *Electronic Notes in Theoretical Computer Science* 336 (2018), pp. 57–78. DOI: 10.1016/j.entcs.2018.03.016 (cit. on p. 9).

[Biz+16]   Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus E. Møgelberg, and Lars Birkedal. "Guarded Dependent Type Theory with Coinductive Types". In: *Foundations of Software Science and Computation Structures: 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*. Ed. by Bart Jacobs and Christof Löding. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 20–35. ISBN: 978-3-662-49630-5. DOI: 10.1007/978-3-662-49630-5_2. arXiv: 1601.01586 [cs.LO] (cit. on p. 9).

[BM15]   Aleš Bizjak and Rasmus Ejlers Møgelberg. "A Model of Guarded Recursion With Clock Synchronisation". In: *Electronic Notes in Theoretical Computer Science* 319.C (Dec. 2015), pp. 83–101. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2015.12.007 (cit. on pp. 9, 62).

[BM20]   Aleš Bizjak and Rasmus Ejlers Møgelberg. "Denotational semantics for guarded dependent type theory". In: *Mathematical Structures in Computer Science* 30.4 (2020), pp. 342–378. DOI: 10.1017/S0960129520000080 (cit. on pp. 9, 62–64).

[Ble17]   Ingo Blechschmidt. "Using the internal language of toposes in algebraic geometry". PhD thesis. Universität Augsberg, 2017 (cit. on p. 95).

[Blu67]   Manuel Blum. "On the size of machines". In: *Information and Control* 11.3 (1967), pp. 257–265. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(67)90546-3 (cit. on p. 10).

[BC05]   Ana Bove and Venanzio Capretta. "Modelling general recursion in type theory". In: *Mathematical Structures in Computer Science* 15.4 (2005), pp. 671–708. DOI: 10.1017/S0960129505004822 (cit. on pp. 10, 11).

[Bra05]   Edwin Brady. "Practical Implementation of a Dependently Typed Functional Programming Language". PhD thesis. Durham University, 2005 (cit. on pp. 7, 11).

[Bra13]   Edwin Brady. "Idris, a general-purpose dependently typed programming language: Design and implementation". In: *Journal of Functional Programming* 23.5 (Sept. 2013), pp. 552–593. DOI: 10.1017/S095679681300018X (cit. on pp. 7, 26).

[Bra21]   Edwin Brady. "Idris 2: Quantitative Type Theory in Practice". In: *35th European Conference on Object-Oriented Programming (ECOOP 2021)*. Ed. by Anders Møller and Manu Sridharan. Vol. 194. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 9:1–9:26. ISBN: 978-3-95977-190-0. DOI: 10.4230/LIPIcs.ECOOP.2021.9. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14052 (cit. on p. 7).

[Bra+11] Edwin Brady, James Chapman, Pierre-Évariste Dagand, Adam Gundry, Conor McBride, Peter Morris, Ulf Norell, and Nicolas Oury. *An Epigram Implementation*. Feb. 2011 (cit. on pp. 7, 26).

[BMM03]   Edwin C. Brady, Conor McBride, and James McKinna. "Inductive Families Need Not Store Their Indices". In: *Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers*. Ed. by Stefano Berardi, Mario Coppo, and Ferruccio Damiani. Vol. 3085. Lecture Notes in Computer Science. Springer, 2003, pp. 115–129. DOI: 10.1007/978-3-540-24849-1_8 (cit. on p. 11).

[BM17]   Ulrik Buchholtz and Edward Morehouse. "Varieties of Cubical Sets". In: *Relational and Algebraic Methods in Computer Science*. Ed. by Peter Höfner, Damien Pous, and Georg Struth. Cham: Springer International Publishing, 2017, pp. 77–92. ISBN: 978-3-319-57418-9 (cit. on p. 133).

[BGS18]   Marta Bunge, Felipe Gago, and Ana María San Luis. *Synthetic Differential Topology*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2018. DOI: 10.1017/9781108553490 (cit. on p. 95).

[BCM20]   Kevin Buzzard, Johan Commelin, and Patrick Massot. "Formalising Perfectoid Spaces". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 299–312. ISBN: 978-1-4503-7097-4. DOI: 10.1145/3372885.3373830 (cit. on p. 6).

[Car78]   John Cartmell. "Generalised Algebraic Theories and Contextual Categories". PhD thesis. Oxford University, Jan. 1978 (cit. on pp. 27, 28, 36).

[Car86]   John Cartmell. "Generalised Algebraic Theories and Contextual Categories". In: *Annals of Pure and Applied Logic* 32 (1986), pp. 209–243. ISSN: 0168-0072 (cit. on p. 13).

[CCD17]   Simon Castellan, Pierre Clairambault, and Peter Dybjer. "Undecidability of Equality in the Free Locally Cartesian Closed Category (Extended version)". In: *Logical Methods in Computer Science* 13.4 (2017) (cit. on pp. 38, 85).

[Cav21]   Evan Cavallo. "Higher Inductive Types and Internal Parametricity for Cubical Type Theory". PhD thesis. Carnegie Mellon University, 2021 (cit. on pp. 17, 18, 159).

[CH18]   Evan Cavallo and Robert Harper. *Computational Higher Type Theory IV: Inductive Types*. 2018. arXiv: 1801.01568 [cs.LO] (cit. on p. 18).

[CH19]   Evan Cavallo and Robert Harper. "Higher Inductive Types in Cubical Computational Type Theory". In: *Proceedings of the ACM on Programming Languages* 3.POPL (Jan. 2019), 1:1–1:27. ISSN: 2475-1421. DOI: 10.1145/3290314 (cit. on pp. 18, 159).

[CB16]   David Christiansen and Edwin Brady. "Elaborator Reflection: Extending Idris in Idris". In: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*. Nara, Japan: Association for Computing Machinery, Sept. 2016, pp. 284–297. ISBN: 978-1-4503-4219-3. DOI: 10.1145/2951913.2951932 (cit. on p. 10).

[CGM11]   Carl von Clausewitz, J. J. Graham, and Federic Nautusch Maude. *On War*. Trans. by J. J. Graham. New & rev. ed. / with introduction and notes by F.N. Maude. Kegan Paul, Trench, Trubner London, 1911 (cit. on p. 4).

[Clo+15]   Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. "Programming and Reasoning with Guarded Recursion for Coinductive Types". In: *Foundations of Software Science and Computation Structures*. Ed. by Andrew Pitts. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 407–421. ISBN: 978-3-662-46678-0. DOI: 10.1007/978-3-662-46678-0_26. arXiv: 1501.02925 [cs.PL] (cit. on p. 9).

[Coh+17]   Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical Type Theory: a constructive interpretation of the univalence axiom". In: *IfCoLog Journal of Logics and their Applications* 4.10 (Nov. 2017), pp. 3127–3169. arXiv: 1611.02108 [cs.LO]. URL: http://www.collegepublications.co.uk/journals/ifcolog/?00019 (cit. on pp. 25, 77, 133, 140).

[Con+86]   R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986. ISBN: 0-13-451832-2 (cit. on pp. 6, 53, 182).

[CS87]   Robert L. Constable and Scott F. Smith. "Partial Objects In Constructive Type Theory". In: *Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987*. IEEE Computer Society, 1987, pp. 183–193 (cit. on p. 8).

[CZ84]   Robert L. Constable and Daniel R. Zlatin. "The Type Theory of PL/CV3". In: *ACM Transactions on Programming Languages and Systems* 6.1 (Jan. 1984), pp. 94–117. ISSN: 0164-0925. DOI: 10.1145/357233.357238 (cit. on p. 6).

[Coq16]   The Coq Development Team. *The Coq Proof Assistant Reference Manual*. 2016 (cit. on pp. 7, 182).

[CMR17]   T. Coquand, B. Mannaa, and F. Ruch. "Stack semantics of type theory". In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. June 2017, pp. 1–11. DOI: 10.1109/LICS.2017.8005130 (cit. on p. 76).

[Coq18]   Thierry Coquand. *Canonicity and normalisation for Dependent Type Theory*. Oct. 2018. arXiv: 1810.09367 [cs.PL]. URL: https://arxiv.org/abs/1810.09367 (cit. on pp. 30, 31).

[Coq19]   Thierry Coquand. "Canonicity and normalization for dependent type theory". In: *Theoretical Computer Science* 777 (2019). In memory of Maurice Nivat, a founding father of Theoretical Computer Science - Part I, pp. 184–191. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2019.01.015. arXiv: 1810.09367 [cs.PL] (cit. on pp. 27, 31, 99, 105, 108, 143).

[CHS19]   Thierry Coquand, Simon Huber, and Christian Sattler. "Homotopy canonicity for cubical type theory". In: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman Geuvers. Vol. 131. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. ISBN: 978-3-95977-107-8 (cit. on pp. 31, 140).

[Cra98]   Karl Crary. "Type-Theoretic Methodology for Practical Programming Languages". PhD thesis. Ithaca, NY: Cornell University, Aug. 1998 (cit. on p. 8).

[CH09]   Karl Crary and Robert Harper. *Mechanized Definition of Standard ML (alpha release)*. 2009. URL: https://www.cs.cmu.edu/~crary/papers/2009/mldef-alpha.tar.gz (cit. on pp. 12, 26, 33).

[Cro93]   R. L. Crole. *Categories for Types*. Cambridge Mathematical Textbooks. New York: Cambridge University Press, 1993. ISBN: 978-0-521-45701-9 (cit. on pp. 28, 87).

[de 70]   N. G. de Bruijn. "The mathematical language AUTOMATH, its usage, and some of its extensions". In: *Symposium on Automatic Demonstration*. Ed. by M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger. Berlin, Heidelberg: Springer Berlin Heidelberg, 1970, pp. 29–61. ISBN: 978-3-540-36262-3 (cit. on p. 5).

[DU21]   Leonardo De Moura and Sebastian Ullrich. "The Lean 4 Theorem Prover and Programming Language (System Description)". To appear in the proceedings of the 28th International Conference on Automated Deduction. 2021 (cit. on p. 7).

[Dia75]   Radu Diaconescu. "Change of base for toposes with generators". In: *Journal of Pure and Applied Algebra* 6.3 (1975), pp. 191–218. ISSN: 0022-4049. DOI: 10.1016/0022-4049(75)90015-8 (cit. on pp. 55, 65, 66, 119, 183).

[DCH03]   Derek Dreyer, Karl Crary, and Robert Harper. "A Type System for Higher-Order Modules". In: *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '03. New Orleans, Louisiana, USA: Association for Computing Machinery, 2003, pp. 236–249. ISBN: 1-58113-628-5. DOI: 10.1145/604131.604151 (cit. on p. 179).

[Dyb96]   Peter Dybjer. "Internal type theory". In: *Types for Proofs and Programs: International Workshop, TYPES '95 Torino, Italy, June 5–8, 1995 Selected Papers*. Ed. by Stefano Berardi and Mario Coppo. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 120–134. ISBN: 978-3-540-70722-6 (cit. on p. 40).

[Fio02]   Marcelo Fiore. "Semantic Analysis of Normalisation by Evaluation for Typed Lambda Calculus". In: *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*. PPDP '02. Pittsburgh, PA, USA: Association for Computing Machinery, 2002, pp. 26–37. ISBN: 1-58113-528-9. DOI: 10.1145/571157.571161 (cit. on pp. 27, 28, 30, 105, 126).

[FH10]   Marcelo Fiore and Chung-Kil Hur. "Second-Order Equational Logic (Extended Abstract)". English. In: *Computer Science Logic*. Ed. by Anuj Dawar and Helmut Veith. Vol. 6247. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 320–335. ISBN: 978-3-642-15204-7 (cit. on pp. 27, 58).

[FM10]   Marcelo Fiore and Ola Mahmoud. "Second-Order Algebraic Theories". In: *Mathematical Foundations of Computer Science 2010: 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*. Ed. by Petr Hliněný and Antonín Kučera. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 368–380. ISBN: 978-3-642-15155-2 (cit. on pp. 12, 27, 58).

[FPT99]   Marcelo Fiore, Gordon Plotkin, and Daniele Turi. "Abstract syntax and variable binding". In: *Proceedings of the 14th Symposium on Logic in Computer Science*. 1999, pp. 193–202 (cit. on pp. 12, 58).

[FR97]   Marcelo P. Fiore and Giuseppe Rosolini. "Two models of synthetic domain theory". In: *Journal of Pure and Applied Algebra* 116.1 (1997), pp. 151–162. ISSN: 0022-4049. DOI: 10.1016/S0022-4049(96)00164-8 (cit. on p. 7).

[Fre78]   Peter Freyd. "On proving that **1** is an indecomposable projective in various free categories". Unpublished manuscript. 1978 (cit. on pp. 30, 68, 102).

[Fre+92] Peter Freyd, Philip Mulry, Giuseppe Rosolini, and Dana Scott. "Extensional PERs". In: *Information and Computation* 98.2 (June 1992), pp. 211–227. ISSN: 0890-5401. DOI: 10.1016/0890-5401(92)90019-C (cit. on p. 7).

[Gac08]   Andrew Gacek. "The Abella Interactive Theorem Prover (System Description)". In: *Proceedings of IJCAR 2008*. Ed. by A. Armando, P. Baumgartner, and G. Dowek. Vol. 5195. Lecture Notes in Artificial Intelligence. Springer, Aug. 2008, pp. 154–161. arXiv: 0803.2305 [cs.LO] (cit. on p. 15).

[Gon08]   G. Gonthier. "Formal Proof — The Four-Color Theorem". In: *Notices of the AMS* 55.11 (2008). URL: https://www.ams.org/notices/200811/tx081101382p.pdf (cit. on pp. 6, 16).

[Gon+13]   Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. "A Machine-Checked Proof of the Odd Order Theorem". In: *ITP 2013, 4th Conference on Interactive Theorem Proving*. Ed. by Sandrine Blazy, Christine Paulin, and David Pichardie. Vol. 7998. LNCS. Rennes, France: Springer, July 2013, pp. 163–179. DOI: 10.1007/978-3-642-39634-2_14. URL: https://hal.inria.fr/hal-00816699 (cit. on pp. 6, 16).

[Gra21]   Daniel Gratzer. *Normalization for Multimodal Type Theory*. 2021. arXiv: 2106.01414 [cs.LO] (cit. on p. 31).

[GSS21]   Daniel Gratzer, Michael Shulman, and Jonathan Sterling. "Strict universes for Grothendieck topoi". In preparation. 2021 (cit. on pp. 71, 74, 76).

[GS20]   Daniel Gratzer and Jonathan Sterling. *Syntactic categories for dependent type theory: sketching and adequacy*. 2020. arXiv: 2012.10783 [cs.LO] (cit. on pp. 13, 14, 29).

[GSB19a]   Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. "Implementing a Modal Dependent Type Theory". In: *Proceedings of the ACM on Programming Languages* 3.ICFP (July 2019), 107:1–107:29. ISSN: 2475-1421. DOI: 10.1145/3341711 (cit. on pp. 26, 32, 106).

[GSB19b]   Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. *Normalization by Evaluation for Modal Dependent Type Theory*. Technical report. July 2019. URL: http://www.jonmsterling.com/pdfs/modal-mltt-tr.pdf (cit. on p. 32).

[Gro60]   Alexander Grothendieck. "Éléments de géométrie algébrique : I. Le langage des schémas". fr. In: *Publications Mathématiques de l'IHÉS* 4 (1960), pp. 5–228. URL: http://www.numdam.org/item/PMIHES_1960__4__5_0 (cit. on p. 54).

[Gro]   The PRL Group. *Constructive Algebra*. Formalization of some constructive algebra in Nuprl. URL: http://www.nuprl.org/wip/Mathematics/constructive!algebra/index.html (cit. on p. 17).

[Har16]   Robert Harper. *Practical Foundations for Programming Languages*. Second. New York, NY, USA: Cambridge University Press, 2016 (cit. on pp. 12, 22, 23, 87).

[Har21]   Robert Harper. *An Equational Logical Framework for Type Theories*. 2021. arXiv: 2106.01484 [math.LO] (cit. on pp. 13, 36).

[HHP93]   Robert Harper, Furio Honsell, and Gordon Plotkin. "A Framework for Defining Logics". In: *Journal of the ACM* 40.1 (Jan. 1993), pp. 143–184. ISSN: 0004-5411. DOI: 10.1145/138027.138060 (cit. on pp. 12, 28, 36).

[HL07]   Robert Harper and Daniel R. Licata. "Mechanizing Metatheory in a Logical Framework". In: *Journal of Functional Programming* 17.4-5 (July 2007), pp. 613–673. ISSN: 0956-7968. DOI: 10.1017/S0956796807006430 (cit. on pp. 36, 86).

[HMM90]   Robert Harper, John C. Mitchell, and Eugenio Moggi. "Higher-Order Modules and the Phase Distinction". In: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. San Francisco, California, USA: Association for Computing Machinery, 1990, pp. 341–354. ISBN: 0-89791-343-4. DOI: 10.1145/96709.96744 (cit. on pp. 24, 177, 178).

[HP05]   Robert Harper and Frank Pfenning. "On Equivalence and Canonical Forms in the LF Type Theory". In: *ACM Transactions on Computational Logic* 6.1 (Jan. 2005), pp. 61–101. ISSN: 1529-3785. DOI: 10.1145/1042038.1042041 (cit. on pp. 13, 32).

[HS00]   Robert Harper and Christopher Stone. "A Type-Theoretic Interpretation of Standard ML". In: *Proof, Language, and Interaction*. Ed. by Gordon Plotkin, Colin Stirling, and Mads Tofte. Cambridge, MA, USA: MIT Press, 2000, pp. 341–387. ISBN: 0-262-16188-5. DOI: 10.5555/345868.345906 (cit. on pp. 26, 33).

[Heg69]   Georg Wilhelm Friedrich Hegel. *Science of Logic*. Trans. by A. V. Miller. London: Allen & Unwin, 1969 (cit. on p. 26).

[Hen50]   Leon Henkin. "Completeness in the Theory of Types". In: *The Journal of Symbolic Logic* 15.2 (1950), pp. 81–91. ISSN: 00224812. DOI: 10.2307/2266967. URL: http://www.jstor.org/stable/2266967 (cit. on p. 119).

[Hil91]   David Hilbert. "Über the vollen Invariantensysteme". In: *Mathematische Annalen* 42 (3 1891), pp. 313–373. DOI: 10.1007/BF01444162 (cit. on p. 54).

[HS98]   Martin Hofmann and Thomas Streicher. "The groupoid interpretation of type theory". In: *Twenty-five years of constructive type theory (Venice, 1995)*. Vol. 36. Oxford Logic Guides. New York: Oxford Univ. Press, 1998, pp. 83–111. DOI: 10.1093/oso/9780198501275.001.0001 (cit. on p. 19).

[Hub18]   Simon Huber. "Canonicity for Cubical Type Theory". In: *Journal of Automated Reasoning* (June 13, 2018). ISSN: 1573-0670. DOI: 10.1007/s10817-018-9469-1 (cit. on pp. 20, 22, 26, 30, 87, 140, 142, 143).

[Hyl91]   J. M. E. Hyland. "First steps in synthetic domain theory". In: *Category Theory*. Ed. by Aurelio Carboni, Maria Cristina Pedicchio, and Guiseppe Rosolini. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 131–156. ISBN: 978-3-540-46435-8 (cit. on pp. 7, 9, 95).

[Ike19]   Mirai Ikebuchi. "A Lower Bound of the Number of Rewrite Rules Obtained by Homological Methods". In: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman Geuvers. Vol. 131. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 24:1–24:17. ISBN: 978-3-95977-107-8. DOI: 10.4230/LIPIcs.FSCD.2019.24. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10531 (cit. on p. 28).

[Jac99]   Bart Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics 141. Amsterdam: North Holland, 1999 (cit. on p. 9).

[Joh82]   Peter T. Johnstone. *Stone Spaces*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982. ISBN: 978-0-521-33779-3 (cit. on p. 54).

[Joh02]   Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium: Volumes 1 and 2*. Oxford Logical Guides 43. Oxford Science Publications, 2002 (cit. on pp. 28, 51, 63, 65, 70).

[JT93]   Achim Jung and Jerzy Tiuryn. "A new characterization of lambda definability". In: *Typed Lambda Calculi and Applications*. Ed. by Marc Bezem and Jan Friso Groote. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 245–257. ISBN: 978-3-540-47586-6 (cit. on p. 87).

[Jun+18]   Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. "Iris from the ground up: A modular foundation for higher-order concurrent separation logic". In: *Journal of Functional Programming* 28 (2018), e20. DOI: 10.1017/S0956796818000151 (cit. on p. 9).

[Jun+15]   Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. "Iris: Monoids and Invariants As an Orthogonal Basis for Concurrent Reasoning". In: *POPL '15: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Mumbai, India: Association for Computing Machinery, 2015, pp. 637–650. ISBN: 978-1-4503-3300-9. DOI: 10.1145/2676726.2676980 (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. 126).

[KHS19]   Ambrus Kaposi, Simon Huber, and Christian Sattler. "Gluing for type theory". In: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman Geuvers. Vol. 131. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. ISBN: 978-3-95977-107-8 (cit. on pp. 28, 31, 119).

[KL21]   Chris Kapulkin and Peter LeFanu Lumsdaine. "The Simplicial Model of Univalent Foundations (after Voevodsky)". In: *Journal of the European Mathematical Society* 23 (6 Mar. 8, 2021), pp. 2071–2126. DOI: 10.4171/JEMS/1050. arXiv: 1211.2851 [math.LO] (cit. on pp. 71, 74).

[KS19]   Chris Kapulkin and Christian Sattler. *Homotopy canonicity of homotopy type theory*. Slides from a talk given at the International Conference on Homotopy Type Theory (HoTT 2019). Aug. 2019. URL: https://hott.github.io/HoTT-2019/conf-slides/Sattler.pdf (cit. on p. 31).

[KBC19]   Ariel Kellison, Mark Bickford, and Robert Constable. "Implementing Euclid's straightedge and compass constructions in type theory". In: *Annals of Mathematics and Artificial Intelligence* 85.2 (2019), pp. 175–192. DOI: 10.1007/s10472-018-9603-0 (cit. on pp. 17, 18).

[Koc06]   Anders Kock. *Synthetic Differential Geometry*. 2nd ed. London Mathematical Society Lecture Note Series. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511550812 (cit. on p. 95).

[Koc09]   Anders Kock. *Synthetic Geometry of Manifolds*. Cambridge Tracts in Mathematics. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511691690 (cit. on p. 95).

[Laf88]   Yves Lafont. "Logiques, catégories & machines : implantation de langages de programmation guidée par la logique catégorique". PhD thesis. Université Paris 7, 1988 (cit. on p. 14).

[LS86]   J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. New York, NY, USA: Cambridge University Press, 1986. ISBN: 0-521-24665-2 (cit. on p. 16).

[Law63]   F. William Lawvere. "Functorial Semantics of Algebraic Theories". PhD thesis. Columbia University, 1963 (cit. on pp. 27, 29, 35, 39, 57).

[Law94]   F. William Lawvere. "Tools for the advancement of objective logic: closed categories and toposes". In: *The logical foundations of cognition*. Ed. by John Macnamara and Gonzalo Reyes. 4. Oxford University Press on Demand, 1994 (cit. on p. 26).

[LS09]   F. William Lawvere and Stephen H. Schanuel. *Conceptual Mathematics: A First Introduction to Categories.* 2nd. New York, NY, USA: Cambridge University Press, 2009. ISBN: 0-521-89485-9 (cit. on p. 26).

[LCH07]   Daniel K. Lee, Karl Crary, and Robert Harper. "Towards a Mechanized Metatheory of Standard ML". In: *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* Nice, France: Association for Computing Machinery, 2007, pp. 173–184. ISBN: 1-59593-575-4. DOI: 10.1145/1190216.1190245 (cit. on pp. 12, 33).

[LH12]   Daniel R. Licata and Robert Harper. "Canonicity for 2-Dimensional Type Theory". In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* Philadelphia, PA, USA: Association for Computing Machinery, 2012, pp. 337–348. ISBN: 978-1-4503-1083-3. DOI: 10.1145/2103656.2103697 (cit. on p. 26).

[Lic+18]   Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. "Internal Universes in Models of Homotopy Type Theory". In: *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK.* 2018, 22:1–22:17. DOI: 10.4230/LIPIcs.FSCD.2018.22 (cit. on p. 156).

[Luo97]   Zhaohui Luo. "Coercive subtyping in type theory". In: *Computer Science Logic.* Ed. by Dirk van Dalen and Marc Bezem. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 275–296. ISBN: 978-3-540-69201-0 (cit. on p. 6).

[MM92]   Saunders Mac Lane and Ieke Moerdijk. *Sheaves in geometry and logic: a first introduction to topos theory.* Universitext. New York: Springer, 1992. ISBN: 0-387-97710-4 (cit. on pp. 51, 65).

[MN94]   Lena Magnusson and Bengt Nordström. "The Alf proof editor and its proof engine". In: *Types for Proofs and Programs.* Ed. by Henk Barendregt and Tobias Nipkow. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 213–237. ISBN: 978-3-540-48440-0 (cit. on p. 6).

[MM16]   Philippe Malbos and Samuel Mimram. "Homological computations for term rewriting systems". In: *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).* Ed. by Delia Kesner and Brigitte Pientka. Vol. 52. Leibniz International Proceedings in Informatics (LIPIcs). Porto, Portugal: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 27:1–27:17. ISBN: 978-3-95977-010-1. URL: https://hal.archives-ouvertes.fr/hal-01678175 (cit. on p. 28).

[Man16]   Bassel Mannaa. "Sheaf Semantics in Constructive Algebra and Type Theory". PhD thesis. University of Gothenburg, 2016. URL: https://gupea.ub.gu.se/handle/2077/48250 (cit. on p. 76).

[Mar71]   Per Martin-Löf. "A Theory of Types". 1971 (cit. on p. 5).

[Mar75a]   Per Martin-Löf. "About Models for Intuitionistic Type Theories and the Notion of Definitional Equality". In: *Proceedings of the Third Scandinavian Logic Symposium*. Ed. by Stig Kanger. Vol. 82. Studies in Logic and the Foundations of Mathematics. Elsevier, 1975, pp. 81–109 (cit. on p. 30).

[Mar75b]   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. DOI: 10.1016/S0049-237X(08)71945-1 (cit. on p. 5).

[Mar79]   Per Martin-Löf. "Constructive Mathematics and Computer Programming". In: *6th International Congress for Logic, Methodology and Philosophy of Science*. Published by North Holland, Amsterdam. 1982. Hanover, Aug. 1979, pp. 153–175 (cit. on pp. 6, 20).

[Mar84]   Per Martin-Löf. *Intuitionistic type theory. Notes by Giovanni Sambin*. Vol. 1. Studies in Proof Theory. Bibliopolis, 1984, pp. iv+91. ISBN: 88-7088-105-9 (cit. on pp. 6, 12, 20).

[Mar87a]   Per Martin-Löf. *The Logic of Judgements*. Workshop on General Logic, Laboratory for Foundations of Computer Science. Feb. 22, 1987 (cit. on pp. 35, 36).

[Mar87b]   Per Martin-Löf. "Truth of a Proposition, Evidence of a Judgement, Validity of a Proof". In: *Synthese* 73.3 (1987), pp. 407–420 (cit. on p. 35).

[Mar90]   Per Martin-Löf. "Mathematics of infinity". In: *COLOG-88*. Ed. by Per Martin-Löf and Grigori Mints. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 146–197. ISBN: 978-3-540-46963-6 (cit. on p. 10).

[Mar96]   Per Martin-Löf. "On the meanings of the logical constants and the justifications of the logical laws". In: *Nordic Journal of Philosophical Logic* 1.1 (1996), pp. 11–60 (cit. on pp. 20, 35, 36).

[MMS21]   Cristina Matache, Sean Moss, and Sam Staton. "Recursion and Sequentiality in Categories of Sheaves". In: *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*. Ed. by Naoki Kobayashi. Vol. 195. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 25:1–25:22. ISBN: 978-3-95977-191-7. DOI: 10.4230/LIPIcs.FSCD.2021.25. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14263 (cit. on p. 8).

[McB99]   Conor McBride. "Dependently typed functional programs and their proofs". PhD thesis. University of Edinburgh, 1999 (cit. on pp. 7, 26).

[McB12]   Conor McBride. "Totality versus Turing Completeness?" Unpublished note. 2012. URL: https://personal.cis.strath.ac.uk/conor.mcbride/pub/Totality.pdf (cit. on p. 11).

[MM04]   Conor McBride and James McKinna. "The View from the Left". In: *Journal of Functional Programming* 14.1 (Jan. 2004), pp. 69–111. ISSN: 0956-7968. DOI: 10.1017/S0956796803004829 (cit. on p. 7).

[MZ15]   Paul-André Melliès and Noam Zeilberger. "Functors are Type Refinement Systems". In: *POPL '15: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Mumbai, India: Association for Computing Machinery, 2015. ISBN: 978-1-4503-3300-9. URL: https://hal.inria.fr/hal-01096910 (cit. on p. 181).

[Mil+97]   Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997 (cit. on pp. 26, 178).

[MS93]   John C. Mitchell and Andre Scedrov. "Notes on sconing and relators". In: *Computer Science Logic*. Ed. by E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. M. Richter. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 352–378. ISBN: 978-3-540-47890-4 (cit. on p. 28).

[MP16]   Rasmus Ejlers Møgelberg and Marco Paviotti. "Denotational Semantics of Recursive Types in Synthetic Guarded Domain Theory". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 317–326. ISBN: 978-1-4503-4391-6. DOI: 10.1145/2933575.2934516 (cit. on p. 9).

[MV19]   Rasmus Ejlers Møgelberg and Niccolò Veltri. "Bisimulation as Path Type for Guarded Recursive Types". In: *Proceedings of the ACM on Programming Languages* 3.POPL (Jan. 2019). DOI: 10.1145/3290317 (cit. on p. 9).

[MV21]   Rasmus Ejlers Møgelberg and Andrea Vezzosi. "Two Guarded Recursive Powerdomains for Applicative Simulation". In: *MFPS37: 37th Conference on Mathematical Foundations of Programming Semantics*. 2021 (cit. on p. 9).

[Mog89]   Eugenio Moggi. "A Category-Theoretic Account of Program Modules". In: *Category Theory and Computer Science*. Berlin, Heidelberg: Springer-Verlag, 1989, pp. 101–117. ISBN: 3-540-51662-X (cit. on p. 177).

[Mog91]   Eugenio Moggi. "Notions of computation and monads". In: *Information and Computation* 93.1 (1991). Selections from 1989 IEEE Symposium on Logic in Computer Science, pp. 55–92. ISSN: 0890-5401. DOI: 10.1016/0890-5401(91)90052-4 (cit. on p. 184).

[Mou+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. "The Lean Theorem Prover (System Description)". In: *Automated Deduction - CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings.* Ed. by Amy P. Felty and Aart Middeldorp. Cham: Springer International Publishing, 2015, pp. 378–388. ISBN: 978-3-319-21401-6 (cit. on p. 7).

[New18] Clive Newstead. "Algebraic Models of Dependent Type Theory". PhD thesis. Carnegie Mellon University, 2018. arXiv: `2103.06155` `[math.CT]` (cit. on p. 119).

[Niu+21] Yue Niu, Jonathan Sterling, Harrison Grodin, and Robert Harper. *A cost-aware logical framework.* Conditionally accepted to POPL '22. 2021. arXiv: `2107.04663` `[cs.PL]` (cit. on p. 11).

[Nog02] Aleksey Nogin. "Quotient Types: A Modular Approach". In: *Theorem Proving in Higher Order Logics, 15th International Conference, TPHOLs 2002, Hampton, VA, USA, August 20-23, 2002, Proceedings.* Ed. by Victor Carreño, César A. Muñoz, and Sofiène Tahar. Vol. 2410. Lecture Notes in Computer Science. Springer, 2002, pp. 263–280. DOI: `10.1007/3-540-45685-6\_18` (cit. on p. 17).

[Nor88] Bengt Nordström. "Terminating general recursion". In: *BIT Numerical Mathematics* 28.3 (1988), pp. 605–619. DOI: `10.1007/BF01941137` (cit. on p. 10).

[NPS90] Bengt Nordström, Kent Peterson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory.* Vol. 7. International Series of Monographs on Computer Science. NY: Oxford University Press, 1990 (cit. on pp. 13, 36).

[Nor09] Ulf Norell. "Dependently Typed Programming in Agda". In: *Proceedings of the 4th International Workshop on Types in Language Design and Implementation.* TLDI '09. Savannah, GA, USA: Association for Computing Machinery, 2009, pp. 1–2. ISBN: 978-1-60558-420-1 (cit. on p. 6).

[OP16] Ian Orton and Andrew M. Pitts. "Axioms for Modelling Cubical Type Theory in a Topos". In: *25th EACSL Annual Conference on Computer Science Logic (CSL 2016).* Ed. by Jean-Marc Talbot and Laurent Regnier. Vol. 62. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 24:1–24:19. ISBN: 978-3-95977-022-4. DOI: `10.4230/LIPIcs.CSL.2016.24` (cit. on pp. 71, 74, 77).

[PV07] E. Palmgren and S. J. Vickers. "Partial Horn logic and cartesian categories". In: *Annals of Pure and Applied Logic* 145.3 (2007), pp. 314–353. ISSN: 0168-0072. DOI: `10.1016/j.apal.2006.10.001` (cit. on p. 27).

[Pal93]   Erik Palmgren. "A note on *Mathematics of infinity*". In: *Journal of Symbolic Logic* 58.4 (1993), pp. 1195–1200. DOI: `10.2307/2275138` (cit. on p. 10).

[Pav16]   Marco Paviotti. "Denotational semantics in Synthetic Guarded Domain Theory". English. PhD thesis. Denmark: IT-Universitetet i København, 2016. ISBN: 978-87-7949-345-2 (cit. on p. 9).

[PMB15]   Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. "A Model of PCF in Guarded Type Theory". In: *Electronic Notes in Theoretical Computer Science* 319.Supplement C (2015). The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI)., pp. 333–349. ISSN: 1571-0661. DOI: `10.1016/j.entcs.2015.12.020` (cit. on p. 9).

[Péd21]   Pierre-Marie Pédrot. *Debunking Sheaves*. Feb. 8, 2021. URL: `https://www.xn-pdrot-bsa.fr/drafts/sheaftt.pdf` (cit. on p. 76).

[PS99]   Frank Pfenning and Carsten Schürmann. "System Description: Twelf — A Meta-Logical Framework for Deductive Systems". In: *Automated Deduction — CADE-16*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 202–206. ISBN: 978-3-540-48660-2 (cit. on p. 15).

[Pho91]   Wesley Phoa. "Domain Theory in Realizability Toposes". PhD thesis. University of Edinburgh, July 1991 (cit. on pp. 7, 9).

[PD10]   Brigitte Pientka and Jana Dunfield. "Beluga: A Framework for Programming and Reasoning with Deductive Systems (System Description)". In: *Automated Reasoning*. Ed. by Jürgen Giesl and Reiner Hähnle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–21. ISBN: 978-3-642-14203-1 (cit. on p. 15).

[Reu95]   Bernhard Reus. "Program Verification in Synthetic Domain Theory". PhD thesis. München: Ludwig-Maximilians-Universität München, Nov. 1995 (cit. on p. 7).

[Reu96]   Bernhard Reus. "Synthetic domain theory in type theory: Another logic of computable functions". In: *Theorem Proving in Higher Order Logics*. Ed. by Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Joakim von Wright, Jim Grundy, and John Harrison. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 363–380. ISBN: 978-3-540-70641-0. DOI: `10.1007/BFb0105416` (cit. on p. 7).

[Reu99]   Bernhard Reus. "Formalizing Synthetic Domain Theory". In: *Journal of Automated Reasoning* 23.3 (1999), pp. 411–444. DOI: `10.1023/A:1006258506 401` (cit. on p. 7).

[RS93]   Bernhard Reus and Thomas Streicher. "Naïve Synthetic Domain Theory — A Logical Approach". Unpublished draft. Sept. 1993 (cit. on p. 7).

[RS99]   Bernhard Reus and Thomas Streicher. "General synthetic domain theory —
            a logical approach". In: *Mathematical Structures in Computer Science* 9.2
            (1999), pp. 177–223. DOI: `10.1017/S096012959900273X` (cit. on p. 7).

[Rey80]   John C. Reynolds. "Using category theory to design implicit conversions and
            generic operators". In: *Semantics-Directed Compiler Generation*. Ed. by Neil
            D. Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 211–258.
            ISBN: 978-3-540-38339-0 (cit. on p. 6).

[Rey83]   John C. Reynolds. "Types, Abstraction, and Parametric Polymorphism". In:
            Information Processing. 1983 (cit. on p. 180).

[RS17]   Emily Riehl and Michael Shulman. "A type theory for synthetic ∞-categories".
            In: *Higher Structures* 1 (1 2017), pp. 147–224. arXiv: `1705.07442 [math.CT]`.
            URL: `https://journals.mq.edu.au/index.php/higher_structures/
            article/view/36` (cit. on p. 77).

[RSS20]   Egbert Rijke, Michael Shulman, and Bas Spitters. "Modalities in homotopy
            type theory". In: *Logical Methods in Computer Science* Volume 16, Issue 1 (Jan.
            2020). DOI: `10.23638/LMCS-16(1:2)2020`. arXiv: `1706.07526 [math.CT]`.
            URL: `https://lmcs.episciences.org/6015` (cit. on p. 78).

[Ros86]   Guiseppe Rosolini. "Continuity and effectiveness in topoi". PhD thesis. Uni-
            versity of Oxford, 1986 (cit. on p. 7).

[Sch00]   Stephen H. Schanuel. "Objective number theory and the retract chain condi-
            tion". In: *Journal of Pure and Applied Algebra* 154.1 (2000). Category Theory
            and its Applications, pp. 295–298. ISSN: 0022-4049. DOI: `10.1016/S0022-
            4049(99)00185-1` (cit. on p. 26).

[Sch87]   Peter Schroeder-Heister. "Structural Frameworks with Higher-level Rules:
            Philosophical Investigations on the Foundations of Formal Reasoning". Habil-
            itation. Fachgruppe Philosophie, Universität Konstanz, 1987 (cit. on p. 36).

[Sch50]   Kurt Schütte. "Beweistheoretische Erfassung der unendlichen Induktion in
            der Zahlentheorie". In: *Mathematische Annalen* 122.5 (1950), pp. 369–389.
            DOI: `10.1007/BF01342849` (cit. on p. 106).

[Sco70]   Dana Scott. "Constructive validity". In: *Symposium on Automatic Demon-
            stration*. Ed. by M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger.
            Berlin, Heidelberg: Springer Berlin Heidelberg, 1970, pp. 237–275. ISBN: 978-
            3-540-36262-3 (cit. on p. 5).

[Shu15a]   Michael Shulman. "The Univalence Axiom for Elegant Reedy Presheaves".
            In: *Homology, Homotopy and Applications* 17 (2 2015), pp. 81–106. DOI:
            `10.4310/HHA.2015.v17.n2.a6`. arXiv: `1307.6248 [math.AT]` (cit. on pp. 71,
            74, 76).

[Shu15b]   Michael Shulman. "Univalence for inverse diagrams and homotopy canonicity".
            In: *Mathematical Structures in Computer Science* 25.5 (2015), pp. 1203–1277.
            DOI: 10.1017/S0960129514000565 (cit. on p. 30).

[Spi+21]   Simon Spies, Lennard Gäher, Daniel Gratzer, Joseph Tassarotti, Robbert
            Krebbers, Derek Dreyer, and Lars Birkedal. "Transfinite Iris: Resolving an
            Existential Dilemma of Step-Indexed Separation Logic". In: *Proceedings of
            the ACM SIGPLAN 2-21 Conference on Programming Language Design and
            Implementation*. Association for Computing Machinery, 2021. DOI: 10.1145/
            3453483.3454031 (cit. on p. 9).

[Ste18]   Jonathan Sterling. *Algebraic Type Theory and Universe Hierarchies*. Dec.
            2018. arXiv: 1902.08848 [math.LO] (cit. on p. 31).

[Ste20]   Jonathan Sterling. *Objective Metatheory of (Cubical) Type Theories*. Thesis
            Proposal. 2020. URL: http://www.jonmsterling.com/pdfs/proposal-
            slides.pdf (cit. on p. 144).

[SA21]   Jonathan Sterling and Carlo Angiuli. "Normalization for Cubical Type The-
            ory". In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer
            Science (LICS)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2021,
            pp. 1–15. DOI: 10.1109/LICS52264.2021.9470719. arXiv: 2101.11479
            [cs.LO] (cit. on pp. 31, 100, 147).

[SAG19]   Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. "Cubical Syntax for
            Reflection-Free Extensional Equality". In: *4th International Conference on For-
            mal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman
            Geuvers. Vol. 131. Leibniz International Proceedings in Informatics (LIPIcs).
            Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019,
            31:1–31:25. ISBN: 978-3-95977-107-8. DOI: 10.4230/LIPIcs.FSCD.2019.31.
            arXiv: 1904.08562 [cs.LO]. URL: http://drops.dagstuhl.de/opus/
            volltexte/2019/10538 (cit. on pp. 30, 31, 87, 143).

[SAG20]   Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. *A cubical language for
            Bishop sets*. Under review. 2020. arXiv: 2003.01491 [cs.LO] (cit. on pp. 30,
            31, 74, 143).

[SH18]   Jonathan Sterling and Robert Harper. "Guarded Computational Type Theory".
            In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in
            Computer Science*. Oxford, United Kingdom: Association for Computing
            Machinery, 2018. ISBN: 978-1-4503-5583-4. arXiv: 1804.09098 [cs.LO] (cit.
            on pp. 9, 62, 64).

[SH21]   Jonathan Sterling and Robert Harper. "Logical Relations as Types: Proof-Relevant Parametricity for Program Modules". In: *Journal of the ACM* 68.6 (Oct. 2021). ISSN: 0004-5411. DOI: 10.1145/3474834. arXiv: 2010.08599 [cs.PL] (cit. on pp. iii, 24, 31, 85, 100, 102, 177, 178, 180).

[SS18]   Jonathan Sterling and Bas Spitters. *Normalization by gluing for free λ-theories.* Sept. 2018. arXiv: 1809.08646 [cs.LO] (cit. on p. 28).

[Str91]   Thomas Streicher. *Semantics of Type Theory: Correctness, Completeness, and Independence Results.* Cambridge, MA, USA: Birkhauser Boston Inc., 1991. ISBN: 0-8176-3594-7 (cit. on p. 26).

[Str14a]   Thomas Streicher. "A model of type theory in simplicial sets: A brief introduction to Voevodsky's homotopy type theory". In: *Journal of Applied Logic* 12.1 (2014), pp. 45–49. DOI: 10.1016/j.jal.2013.04.001 (cit. on pp. 71, 74).

[Str14b]   Thomas Streicher. *Semantics of Type Theory Formulated in Terms of Representability.* Feb. 2014. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/FIBR/natmod.pdf (cit. on p. 76).

[Tai67]   W. W. Tait. "Intensional Interpretations of Functionals of Finite Type I". In: *The Journal of Symbolic Logic* 32.2 (1967), pp. 198–212. ISSN: 00224812. URL: http://www.jstor.org/stable/2271658 (cit. on pp. 31, 108).

[Tay91]   Paul Taylor. "The fixed point property in synthetic domain theory". In: *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science.* 1991, pp. 152–160. DOI: 10.1109/LICS.1991.151640 (cit. on pp. 7, 8).

[Tay99]   Paul Taylor. *Practical Foundations of Mathematics.* Cambridge studies in advanced mathematics. Cambridge, New York (N. Y.), Melbourne: Cambridge University Press, 1999. ISBN: 0-521-63107-6 (cit. on p. 28).

[Uem19]   Taichi Uemura. *A General Framework for the Semantics of Type Theory.* 2019. arXiv: 1904.04097 [math.CT] (cit. on pp. 13, 14, 27–29, 36).

[Uem21]   Taichi Uemura. "Abstract and Concrete Type Theories". PhD thesis. Amsterdam: Universiteit van Amsterdam, 2021 (cit. on pp. 13, 14).

[Uni13]   The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* Institute for Advanced Study: https://homotopytypetheory.org/book, 2013 (cit. on p. 21).

[VV20]   Niccolò Veltri and Andrea Vezzosi. "Formalizing π-Calculus in Guarded Cubical Agda". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs.* New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 270–283. ISBN: 978-1-4503-7097-4. DOI: 10.1145/3372885.3373814 (cit. on p. 9).

[Vic92]   S. Vickers. "Geometric theories and databases". In: *Applications of Categories in Computer Science: Proceedings of the London Mathematical Society Symposium, Durham 1991*. Ed. by M. P. Fourman, P. T. Johnstone, and A. M. Editors Pitts. London Mathematical Society Lecture Note Series. Cambridge University Press, 1992, pp. 288–314. DOI: 10.1017/CBO9780511525902.017 (cit. on p. 63).

[Vic07]   Steven Vickers. "Locales and Toposes as Spaces". In: *Handbook of Spatial Logics*. Ed. by Marco Aiello, Ian Pratt-Hartmann, and Johan Van Benthem. Dordrecht: Springer Netherlands, 2007, pp. 429–496. ISBN: 978-1-4020-5587-4. DOI: 10.1007/978-1-4020-5587-4_8 (cit. on p. 51).

[Voe06]   Vladimir Voevodsky. "A very short note on homotopy λ-calculus". Unpublished note. Sept. 2006. URL: http://www.math.ias.edu/vladimir/files/2006_09_Hlambda.pdf (cit. on p. 19).

[Wat+04]   Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. "A Concurrent Logical Framework: The Propositional Fragment". In: *Types for Proofs and Programs*. Ed. by Stefano Berardi, Mario Coppo, and Ferruccio Damiani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 355–377. ISBN: 978-3-540-24849-1 (cit. on p. 13).

[WB18]   Paweł Wieczorek and Dariusz Biernacki. "A Coq Formalization of Normalization by Evaluation for Martin-Löf Type Theory". In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. Los Angeles, CA, USA: Association for Computing Machinery, 2018, pp. 266–279. ISBN: 978-1-4503-5586-5. DOI: 10.1145/3167091 (cit. on pp. 32, 106).

[Xu15]   Chuangjie Xu. "A continuous computational interpretation of type theories". PhD thesis. University of Birmingham, July 2015. URL: http://etheses.bham.ac.uk/5967/ (cit. on p. 76).

[XE16]   Chuangjie Xu and Martín Escardó. *Universes in sheaf models*. Unpublished note. 2016. URL: https://cj-xu.github.io/notes/sheaf_universe.pdf (cit. on p. 76).

[Yet87]   David Yetter. "On right adjoints to exponential functors". In: *Journal of Pure and Applied Algebra* 45.3 (1987), pp. 287–304. ISSN: 0022-4049. DOI: 10.1016/0022-4049(87)90077-6. URL: http://www.sciencedirect.com/science/article/pii/0022404987900776 (cit. on p. 69).

[Zei08]   Noam Zeilberger. "Focusing and Higher-Order Abstract Syntax". In: *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. San Francisco, California, USA: Association for Computing Machinery, 2008, pp. 359–369. ISBN: 978-1-59593-689-9. DOI: 10.1145/1328438.1328482 (cit. on p. 106).

[Zei09]   Noam Zeilberger. "The logical basis of evaluation order and pattern-matching". PhD thesis. Carnegie Mellon University, 2009 (cit. on p. 106).

LIST OF SYMBOLS

NAMES OF TOPOI

$\mathbb{S}$	the Sierpiński topos	**(2.2.4∗1)**
T	the classifying topos of Henkin models of Martin-Löf type theory	**(5.5.2∗1)**
T$_\square$	the classifying topos of Henkin models of Cartesian cubical type theory	**(7.5.1∗3)**
A	the topos of atomic terms	**(5.5.2∗8)**
A$_\square$	the topos of cubical atomic terms	**(7.5.1∗3)**
G	the glued topos used to prove canonicity or normalization of Martin-Löf type theory	**(5.5.3∗1)**
G$_\square$	the glued topos used to prove normalization for Cartesian cubical type theory	**(7.5.2∗1)**

NAMES OF CATEGORIES

$\mathcal{T}$	the category of judgments of Martin-Löf type theory	**(1.4∗8)**
$\mathcal{T}_\square$	the category of judgments of Cartesian cubical type theory	**(4.1.1∗1)**
$\mathcal{A}$	the category of atomic contexts and substitutions	**(5.5.1∗7)**
$\mathcal{A}_\square$	the category of cubical atomic contexts and substitutions	**(7.5.1∗3)**

INTERNAL NOTIONS

$\{\phi\}\,A$	the type of partial elements of A with support ϕ	**(3.5∗3)**
$\{A \mid \phi \hookrightarrow a\}$	the extent of a partial element $a : \{\phi\}\,A$	**(3.5∗5)**
$\bigcirc_\phi$	the open modality associated to a proposition ϕ	**(3.6∗7)**
$\bullet_\phi$	the closed modality associated to a proposition ϕ	**(3.6∗7)**
$\mathcal{U}_{\backslash\phi}$	the closed subuniverse of $\mathcal{U}$ determined by a proposition ϕ	**(3.6∗6)**
$\mathcal{U}_\phi$	the closed subuniverse of $\mathcal{U}$ determined by a proposition ϕ	**(3.6∗4)**

213

¶   the "syntactic open", a proposition under which the semantic   **(4.4∗1)**
                     part of a computability structure is zeroed out

○/●   the open/closed modality associated to ¶   **(4.4∗1)**

$X \wr_\phi Y$ the syntactic stabilization of X by Y along ϕ **(7.1∗6)**

$[x \mid \phi \hookrightarrow y]$ an element of the the syntactic stabilization $X \wr_\phi Y$ **(7.1∗6)**