STRUCTURE AND LANGUAGE OF HIGHER-ORDER ALGEBRAIC EFFECTS

ZHIXUAN YANG

A thesis submitted in partial fulfilment of the requirements for
the degree of Doctor of Philosophy at
Department of Computing
Imperial College London
September
2024

Abstract

The research programme of *algebraic effects* studies non-pure features in programming languages, commonly referred to as computational effects, through the lens of algebraic theories of effectful operations. The free algebra monad induced by an algebraic theory provides the necessary structure for interpreting the computational effect in a programming language. Moreover, the universal property of free algebras can be internalised in programming languages as a useful programming construct known as *effect handlers*.

This thesis presents a generalisation of algebraic effects, called *higher-order algebraic effects* in this thesis, widening the range of effects that can be treated algebraically. The central idea is to shift focus from algebraic theories of operations to algebraic theories of *monads equipped with operations*, or more generally, theories of monoids equipped with operations in a monoidal category.

Part I of the thesis first develops a convenient categorical framework and a syntactic language for presenting algebraic theories of operations on monoids, and categorical properties of such theories are studied within this framework. Additionally, a formal theory of modular constructions of algebraic structures is proposed, based on lifting functors along fibrations.

In Part II of the thesis, a programming language with higher-order impredicative polymorphism and a restricted form of higher-order algebraic effects is presented. The consistency and the computational interpretation of the language are given by a realizability model. The canonicity of closed terms is proven using synthetic Tait computability. An extension of the language with general recursion is considered and is modelled using synthetic domain theory.

Examination Committee

Paul Kelly
Imperial College London

Jonathan Sterling
University of Cambridge

Tarmo Uustalu
Reykjavik University
Tallinn University of Technology

Statement of Originality

The author certifies that the thesis presented here is solely my own work other than where I have clearly indicated that it is the work of others.

Open Access

Acknowledgements

I'm forever indebted to my supervisor Nick Wu, who took me under his wings when I was at the lowest point of my life, and since then has been unreservedly supportive to me, helping me become a better researcher and a better person in many ways. Also, if it weren't Nick's previous work on scoped effects, I wouldn't be writing a thesis about higher-order algebraic effects at all. Having Nick as my mentor is one of the luckiest things that ever happened to me.

My second biggest source of knowledge during the last four years has been the online programming language theory community: Jon Sterling, Ende Jin, Max New, Yufeng Li, Tesla Zhang, Chris Grossack, Nathanael Arkor, Joey Eremondi, Nathan Corbyn, Danielle Marshall, Zanzi, and many many other names that I can't fully enumerate here. Special thanks go to Nathanael for his excellent commutative diagram editor QUIVER, without which typesetting the diagrams in this thesis would be insurmountable nightmare. I can't overstate how much I learnt from them by just reading their tweets/toots. Without this vibrant and friendly community, I wouldn't have even heard of some of the mathematics that was used critically in this thesis.

I'm especially indebted to Jon Sterling – his insights into type theory and many other things that he shared online and in his writings completely reshaped my understanding of programming language theory. If Nick is the person who taught me the most through direct guidance, Jon is then the person who taught me the most through his writings. It isn't exaggerating to say that Part II of this thesis is my naive fan fiction of Jon's work.

I'm also very grateful to Jon and my two other thesis examiners Tarmo Uustalu and Paul Kelly for examining this thesis in a very short time, travelling all the way to London for my viva, and giving me a ton of helpful feedback and warm encouragement. Having them as my examiners is one of my biggest honours.

My longtime friend Josh Ko, also my de facto supervisor when we were in Japan, was the person who introduced me to real programming language theory and serious scientific work in general. Without him, I might end up writing ridiculous software engineering papers using large language models. When my academic life as a graduate student was falling apart in Japan, it was Josh and Josh's former PhD supervisor Jeremy Gibbons who encouraged and helped me to come to the UK for a second chance. Josh was also the person who advised me:

'if you want to work on algebraic effects, you have to learn category theory and figure out what these scary integration symbols mean' when I knew absolutely nothing about category theory in 2019 – I'm glad I took his advice. Although we've been working on different research topics now, Josh is still a role model to me of pursuing a high standard of research and writing.

I am grateful to my coauthors during my PhD: Marco Paviotti, Birthe van den Berg, Tom Schrijvers, Donnacha Oisín Kidney Sam Lindley, Cristina Matache, Sean Moss, Sam Staton. They have taught me so much about computational effects, which was essential for me to write this thesis.

I cherish my friendships with the people that I met at Imperial College London: Jamie Willis, Marco Paviotti, Csongor Kiss, Matthew Pickering, Donnacha Oisín Kidney, David Davies, Niels Bunkenburg, Omar Tahir, Shing Hin Ho, Paulo Emílio de Vilhena, Jacob Yu, Alyssa Renata. I am especially thankful to Jamie for always being patient to teach me how to live in the UK. I'm also very grateful to our departmental PhD programme manager Dr Amani El-Kholy for consistently providing me with the most prompt help in administrative matters.

Lastly, I couldn't be more grateful to my girlfriend Sijia Chen and my parents Jubo Yang and Xumei Chen for their love and unlimited support. I hope I will be a better partner and a better son in the future.

Zhixuan Yang
Canterbury
September 2024

Contents

Chapter 1

Prologue

1∗1. Natural languages evolved to describe things around us. Logical languages are formulated to denote mathematical objects. Programming languages are invented to compose computational procedures. But since the late 1960s, programming language theorists have known that it is fruitful to not just use programming languages for denoting computations, but to study the *general class of mathematical objects* that a programming language can denote, in the same tradition of how logicians study the model theory of logical languages.

The field of studying what mathematical objects programming languages denote is known as *denotational semantics*, which is the area that this thesis belongs to. This thesis focuses on one specific topic: *higher-order algebraic effects*. What are higher-order algebraic effects? And what are they good for? To answer these questions, I beg the reader's indulgence as we take a step back to discuss a bit of the history of denotational semantics and computational effects.

1.1 Denotational Semantics in the 70s and 80s

1.1∗1. The starting point of denotational semantics of programming languages was Dana Scott's 1969 report *A Type-Theoretical Alternative to ISWIM, CUCH, OWHY*, which was widely circulated but unpublished until 1993. In this report, Scott advocated *typed* programming languages that have clear mathematical denotations over untyped programming languages, which the acronyms in the title refer to, that *by then* had been purely syntactic/formalistic.

The programming language and program logic in Scott's paper would later be known as PCF (Programming Computable Functions) and LCF (Logic for Computable functions). The denotational semantics of a context Γ or a type τ of PCF is a partial order $[\![\Gamma]\!]$ or $[\![\tau]\!]$ that has an bottom element and a join for all ω-chains, now called a *pointed ω-cpo*. The semantics of every program $\Gamma \vdash t : \tau$ of PCF is then a monotone function $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$ that preserves joins of ω-chains. Crucially, the existence of least elements and joins of ω-chains allows one to interpret general recursion at all types.

1.1∗2. Scott's denotational approach to programming language gained great momentum in the 70s and 80s. A spectacular achievement was Scott's discovery of models of *untyped* λ-calculus by constructing *continuous lattices* D that are isomorphic to the function space D^D [Scott 1972] and subsequently a universal model in the powerset of natural numbers [Scott 1976]. The study of these kinds of special partial orders became a whole new field of mathematics, *domain theory*, and solving recursive equations of mixed variance like $D \cong D^D$ thereafter remained a central topic in domain theory, as they model not just untyped languages but also recursively defined datatypes in typed languages.

1.1∗3. Apart from general recursion, the semantics of *concurrency* was another notable topic studied from the 1970s onwards. Several constructions of *power domains*, analogues of *power sets* for domains, were introduced for modelling nondeterminism [Plotkin 1976; Smyth 1978; Winskel 1985].

1.1∗4. Another influential piece of work in the 70s was Plotkin's [1977] treatment on the connection between the denotational semantics and operational semantics of PCF. Two questions that Plotkin studied for PCF would later be repeatedly asked for all sorts of programming languages by computer scientists in the following decades: *adequacy* (whether the denotation of a program determines its operational behaviour) and *full abstraction* (whether two programs have the same denotation if and only if they are operationally equivalent).

1.1∗5. The 1970s and 80s also saw the adoption of *category theory* in theoretical computer science. Notable contributions are made by Goguen et al. [1977] (more commonly known as the ADJ group), who promoted the view of abstract syntax as initial algebras of endofunctors, Arbib and Manes [1975], and Adámek [1974], who initiated studying automata as algebras of endofunctors.

During this period, *categorical logic*, pioneered by William Lawvere, was also under rapid development; see Marquis and Reyes [2004] for a detailed historical account. The methodology of categorical logic – organising the syntax of a logic as a category with certain structure and semantic models as *structure-preserving functors* – would be quickly embraced by programming language theorists. For instance, the denotational semantics of *polymorphism* [Asperti and Martini 1992; Moggi 1987; Pitts 1987] developed in the 80s had been using category theory way more extensively than denotational semantics from the 70s.

1.1∗6. Why are we interested in denotational semantics? The motivation is multifold. Firstly, it makes it possible to manipulate some complex mathematical objects using simpler, usually mechanisable, formal languages, thus making the tools developed by programming language theorists useful to more branches of mathematics. This approach is sometimes known as *synthetic mathematics*, and a

notable example is the development of *homotopy type theory*, which enables one to manipulate *homotopy types* (i.e. ∞-*groupoids*) using a relatively simple language, namely, Martin-Löf type theory with a univalence axiom [Hou (Favonia) 2017; Kapulkin and Lumsdaine 2021; Univalent Foundations Program 2013].

Secondly, even if only programming itself is concerned, programming language theorists may be interested in *meta-theoretic properties* of their languages, for example, whether programs of a language always terminate, whether a program is definable in a language, whether two programs have the same behaviour, whether programs of a language never leak critical information, and so on. Denotational methods proved to be an important, if not *the* most important approach to answering such questions, by constructing models of the programming language that prove or disprove the property in question. Moreover, the syntax and semantics of programming languages are dialectic: it is common that studying the underlying semantic structure suggests how a language can be designed or extended. (This is also why the title of this thesis is *Structure and Language of* [...] rather than *Syntax and Semantics of* [...].)

Thirdly, even in practical programming, software with customisable behaviours usually implements some *domain-specific languages*, such as languages for automating a workflow, languages for configuring a system, and languages for querying a certain kind of data. As the joke 'Greenspun's Tenth Rule of Programming' [Greenspun 2003] goes:

> Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified bug-ridden slow implementation of half of Common Lisp.

Implementing such domain-specific languages in *functional programming* can then take direct inspiration from the study of denotational semantics, since functional programming languages are close to the mathematical language.

1.2 Computational Effects as Monads

1.2∗1. By the end of the 1980s, programming language theorists had studied a variety of languages that are a pure language, usually some kind of λ-calculus, extended with some computational features: partiality, exceptions, side-effect, non-determinism, continuation, user input/output, etc. These features are collectively referred to as *computational effects*, generalising the term *side-effects*, which is reserved for the effect of reading and writing memory.

1.2∗2. A unified semantic account of all these computational effects was proposed by Moggi [1989a,b, 1991], whose idea was that *values* of a type τ and *computations* of a type τ in a programming language should be given different semantics.

More precisely, to interpret a language L with some computational effect in a category $\mathscr{C}$, if values of type τ are interpreted as an object $[\![\tau]\!] \in \mathscr{C}$, computations of type τ should be interpreted as another object $T[\![\tau]\!] \in \mathscr{C}$, where $T : \mathscr{C} \to \mathscr{C}$ is some mapping from values to effectful computations.

The structure that Moggi demanded for T was that of *monads*, a basic concept in category theory. A monad on a category $\mathscr{C}$, in the so-called Kleisli-triple formulation, consists of the following data (subject to certain laws):

1. an object $TA \in \textsc{Obj}\,\mathscr{C}$ for every object $A \in \textsc{Obj}\,\mathscr{C}$,
2. a morphism $\eta_A : A \to TA$ in $\mathscr{C}$ for every $A \in \textsc{Obj}\,\mathscr{C}$, and
3. a morphism $f^* : TA \to TB$ for every $A, B \in \textsc{Obj}\,\mathscr{C}$ and $f : A \to TB$.

As mentioned earlier, the object TA is used for interpreting computations producing A-values. The morphism $\eta_A : A \to TA$ is used for embedding values as computations that perform no effects. The morphism f^* is used for *sequential composition* of two computations: given $f : A \to TB$ and $g : B \to TC$, we have $g^* \cdot f : A \to TC$. When the programming language to be interpreted has structural contexts of variables, an additional piece of data called a *strength* $s_{\Gamma,B} : \Gamma \times TA \to T(\Gamma \times A)$ is needed for the interaction of contexts and effects, but let us leave them out in this introduction for the sake of simplicity.

The structure of monads is what Moggi used as the 'backbone' for interpreting computational effects. For specific effects, we also need 'flesh' – additional operations on the monad T that give semantics to the specific effectful operations. For example, to interpret side-effect, we would also need the monad T to come with operations that interpret memory reading and writing.

1.2∗3. The following are a few by now standard examples of monads on the category of sets that respectively model a computational effect. For brevity we only show the mapping on objects for each monad (c.f. Moggi [1989a,b, 1991]).

- The state monad $A \mapsto (A \times S)^S$ for a fixed set S models the effect of a single mutable state storing a value of S.
- The continuation monad $A \mapsto R^{(R^A)}$ for a fixed set R models the effect of *call with current continuation* (call/cc).
- The finite distribution monad $A \mapsto DA$ models the effect of probabilistic choice, where the set DA contains all functions $p : A \to [0, 1]$ such that $p(x) \neq 0$ for finitely many $x \in A$ and $\sum_{x \in A} p(x) = 1$.
- The exception monad $A \mapsto E + A$ for a fixed set E models the effect of throwing and catching exceptions from the set E.
- The powerset monad $A \mapsto \mathscr{P}A$ models non-deterministic choice.

In addition to these simple ones, there has been a large body of research on monads for more realistic computational effects since Moggi's proposal. Two notable lines of work are the monads for mutable state that supports memory allocation [Plotkin and Power 2002; Staton 2010], storing pointers [Kammar et al. 2017], storing higher-order functions [Levy 2002; Sterling et al. 2023, 2024] and the *huge* family of monads for probabilistic/statistic/randomised programming languages that we cannot enumerate here (see nLab [2024c] for a table).

1.3 Computational Effects Post-Moggi

1.3∗1. Aside from the prolific work on monads for specific computational effects, there have also been two conceptual developments of Moggi's framework of computational effects as monads, both of which are directly related to this thesis.

1.3∗2. The first development is that categorical structures that are similar to monads but not exactly monads have been suggested for modelling computational effects in ways that are either more general or more precise than using monads: Hugh's arrows [Hughes 2000; Jacobs et al. 2009; Lindley et al. 2011], applicative functors [McBride and Paterson 2008; Paterson 2012], graded monads [Katsumata 2014; Katsumata et al. 2022; McDermott and Uustalu 2022a], and parameterised monads [Atkey 2009; Orchard et al. 2020]. This diversity of categorical structure is then re-unified by the concept of *monoids in monoidal categories* [Rivas and Jaskelioff 2017], a concept that encompasses monads and all the concepts mentioned above.

1.3∗3. The second development is the research programme commonly known as *algebraic effects*, initiated by Plotkin and Power [2002, 2004], who advocated another uniform framework of modelling computational effects by studying *algebraic theories* (to be technically precise, enriched algebraic theories) of the operations of computational effects. For example, a possible theory of nondeterminism is that of a binary operation $x + y$ for nondeterministic choice and the equations $(x + y) + z = x + (y + z)$, $x + y = y + x$, and $x + x = x$.

Unlike the generalisation to monoids in 1.3∗2, the point of using algebraic theories instead of monads *is not* about generality or expressivity. In fact, algebraic theories and monads are roughly equivalent mathematically: every algebraic theory generates a monad by the free algebra construction (which can then be used for giving semantics to programming languages with effects) and every monad determines a possibly infinitary algebraic theory [Linton 1966].

The point of using algebraic theories to model computational effects is instead conceptual simplicity and naturality, acknowledging our intuition that it is the effectful operations rather than the structure of monads (sequential composition and returning a pure value) that are the 'flesh' of a computational effect. For many

monads used for modelling computational effects, Plotkin and Power [2002] showed that they can be generated from some very simple algebraic theories. For example, in the category of directed cpos, the theory of nondeterminism above generates the *convex power domain monad*. Therefore, using algebraic theories, we can give semantics to a computational effect by simply listing the operations and equations that we wish to hold. Having an concise explicit formula of the corresponding monad *is* useful for mathematical calculation and efficient implementation, but they *logically come after* the algebraic theories.

1.3∗4. While the research programme of algebraic effects initially focused solely on the semantics of computational effects, as previously mentioned, the study of semantics can influence the design of syntax too. This proved true for algebraic effects after Plotkin and Pretnar [2009, 2013] proposed a new programming language feature, *effect handlers*, internalising the universal property of monads generated from algebraic theories in a programming language.

From a programming perspective, effect handlers make a programming language *extensible*: the programmer can on the fly add new effectful operations to the language, using the new operations to write programs, and then give semantics to those new operations wherever needed by supplying a model (called a *handler* in this context), and possibly different models in different scenarios.

The flexibility of effect handlers has led to a quick adoption of them in the programming language theory community. There has now been a large body of research on almost every aspect of effect handlers: operational semantics, type systems for tracking effects precisely, efficient compilation, effect polymorphism, connections to other primitives such as delimited continuations, so on and so forth. It is impossible to review even a tiny fraction of this development here without omitting something equally worth mentioning, so we shall just refer the reader to the community-maintained online bibliography at `https://github.com/yallop/effects-bibliography`.

1.4 Higher-Order Algebraic Effects

1.4∗1. However, there is limitation in Plotkin and Pretnar's [2009; 2013] framework on *what kind of operations* can be added to the programming language and interpreted by handlers. To explain limitation more precisely, let us suppose that $\mathscr{C}$ is some suitable category that we use for interpreting types and pure programs of the programming language; for example, $\mathscr{C}$ may be Set, ωCpo, or some topos. Under some conditions, every ($\mathscr{C}$-enriched) algebraic theory T over $\mathscr{C}$ determines a strong monad M_T on $\mathscr{C}$ such that every operation o of the theory T induces a natural transformation $o_X : P \times (M_T X)^N \to M_T X$ for some objects

$P, N \in \mathscr{C}$ making the following diagram commute:

$$\begin{array}{ccc} P \times (M_T(M_T X))^N & \xrightarrow{o_{M_T X}} & M_T(M_T X) \\ {\scriptstyle P \times \mu_X^N}\downarrow & & \downarrow{\scriptstyle \mu_X} \\ P \times (M_T X)^N & \xrightarrow[o_X]{} & M_T X \end{array}$$

where μ is the multiplication of M_T. Operations $o_X : P \times (M_T X)^N \to M_T X$ satisfying this diagram is sometimes called *algebraic operations* in the literature [Plotkin and Power 2003]. Since monadic multiplication is used for interpreting *sequential composition* of computations in programming languages, what this diagram says is that the effectful operation o commutes with sequential composition. In more conventional programming language syntax, this means

$$\big(\textsc{do}\ x \leftarrow o(p, \lambda n.\ a);\ k\big) = o\big(p, \lambda n.\ (\textsc{do}\ x \leftarrow a;\ k)\big) \tag{1.1}$$

where o is the effectful operation taking a parameter p of type P and N-many computations as arguments, represented as a λ-function with a bound variable n of type N, and k is a computation with a free variable x. The computation $(\textsc{do}\ x \leftarrow a; k)$ first does a, binds its return value to x, then does k.

Equation (1.1) is reasonable for *some* effectful operations in practice. For example, if o is the operation that makes a non-deterministic choice of continuing as one of its n arguments, then both sides of (1.1) are operationally the same: making a non-deterministic choice of $i \in \{1, \ldots n\}$, doing a_i, and then doing k.

However, there is no reason to expect all effectful operations to take the form of $o_X : P \times (M_T X)^N \to M_T X$ and satisfies equation (1.1). For example, the operation of *call/cc* on a monad M may be formulated as a family of morphisms

$$c_{X,Y} : ((X \Rightarrow MY) \Rightarrow MX) \to MX$$

natural in $X, Y \in \mathscr{C}$, where the notation $X \Rightarrow Y$ means the exponential Y^X, so c does not fit in the form $o_X : P \times (M_T X)^N \to M_T X$.

Moreover, even when an effectful operation o is of the form $P \times (MX)^N \to MX$, it does not necessarily satisfy equation (1.1). For example, if o is the operation that runs its n arguments in parallel (and combines their return values in some way), then the two sides of (1.1) behave differently: the left-hand side runs $a_1, \ldots, a_n$ in parallel, combines their return values, and then does k once, whereas k is run n-times in parallel on the right-hand side.

1.4∗2. Natural transformations $o : P \times (M-)^N \to M$ on a monad M for some $P, N \in \mathscr{C}$ that do *not* satisfy (1.1) are called *scoped operations* on the monad M by Piróg et al. [2018]. This name comes from the intuition that such an operation $o(p, a)$ for $p : P$ and $a : (MX)^N$ delimits a meaningful boundary between the

computation argument a and future continuation after the scoped operation. For readers whose eyes are trained to parse $\{\cdots\}$ as markers of scopes, it is probably more suggestive to write a scoped operation as $o\ p\ \{n.\ a\}$ or $o\ p\ \{a_1\}\ \cdots\{a_n\}$ when N is $1 + \cdots + 1$, so for scoped operations o,

$$\textsc{do}\ x \leftarrow o\ p\ \{n.\ a\};\ k \quad \neq \quad \textsc{do}\ x \leftarrow o\ p\ \{n.\ a;\ k\}.$$

This is in contrast with algebraic operations $P \times (MX)^N \to MX$ that satisfy equation (1.1), for which the boundary between the argument a and the continuation k is irrelevant. Indeed, ($\mathscr{C}$-enriched, same below) natural transformations $P \times (M-)^N \to M$ satisfying (1.1) are in bijection with natural transformations $P \times (-)^N \to M$, which are further in bijection with $\mathscr{C}$-morphisms $P \to MN$, revealing the fact that algebraic operations do not really take computations as arguments but are actually 'atomic computations' that take in a parameter $p : P$ and returns a result $n : N$. This distinction is exhibited by comparing (binary) nondeterministic choice and parallel composition of processes: they could both be given as natural transformations $+, \| : (M-)^2 \to M$, while nondeterministic choice can also be given as a morphism $1 \to M2$ and $\|$ cannot.

1.4∗3. To extend algebraic effects to encompass non-algebraic operations such as scoped operations, we develop *higher-order algebraic effects* in this thesis. The idea is simple: instead of considering algebraic theories over the base category $\mathscr{C}$ (for interpreting types of the programming language), we consider algebraic theories in the endofunctor category $\textsc{Endo}(\mathscr{C})$, in particular, theories of *monads with operations*. Here an operation on a monad M means a morphism $\Sigma M \to M$ in $\textsc{Endo}(\mathscr{C})$, where Σ is a functor $\textsc{Endo}(\mathscr{C}) \to \textsc{Endo}(\mathscr{C})$. The special case of $\Sigma M = P \times (M-)^N$ recovers algebraic operations, and of course a general $\Sigma : \textsc{Endo}(\mathscr{C}) \to \textsc{Endo}(\mathscr{C})$ allows a richer class of operations to be modelled. Under some conditions, every such a theory T of monads with operations has an initial algebra, which is a monad equipped with T-operations, and we can use this monad for interpreting effectful programming languages as usual.

1.4∗4. An additional advantage of shifting our attention from $\mathscr{C}$ to $\textsc{Endo}(\mathscr{C})$ is that the latter readily suggests a further generalisation to consider algebraic theories of *monoids with operations* in an arbitrary monoidal category $\mathscr{E}$. This gives us a unified and generalised account of the development of *computational effects as monoids* (1.3∗2) and *computational effects as algebraic theories* (1.3∗3).

1.5 Structure of This Thesis

1.5∗1. This thesis has two parts. In Part I, we study the categorical foundation of higher-order algebraic effects. In Part II, we study programming languages for

higher-order algebraic effects and their meta-theoretic properties.

1.5∗2 (Part I). In Chapter 2, we first develop a convenient way of presenting algebraic theories over monoidal categories by using *equational systems* originally proposed by Fiore and Hur [2009] and the internal language of monoidal categories, which we call *monoidal algebraic theories*. Then we study some subcategories of algebraic theories of monoids equipped with additional operations.

In Chapter 3, we propose a general theory for *modularity in algebraic structures*. The main idea is to define a *modular model* M of an algebraic theory Σ to be a *lifting* of the functor $(- + \Sigma) : \mathscr{T} \to \mathscr{T}$ along the fibration $P : \mathscr{A} \to \mathscr{T}$

$$\begin{array}{ccc} \mathscr{A} & \xrightarrow{M} & \mathscr{A} \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P} \\ \mathscr{T} & \xrightarrow[-+\Sigma]{} & \mathscr{T} \end{array}$$

where $\mathscr{T}$ is the category of algebraic theories in question and $P : \mathscr{A} \to \mathscr{T}$ is the fibration given by the Grothendieck construction for $(-)$-Alg $: \mathscr{T}^{\mathrm{op}} \to \mathrm{CAT}$, so $\mathscr{A}$ is the category contains all models of all theories in $\mathscr{T}$.

This idea is then generalised to considering liftings $M : \mathscr{A} \to \mathscr{A}'$ of an arbitrary functor $T : \mathscr{T} \to \mathscr{T}'$ along two possibly different fibrations $P : \mathscr{A} \to \mathscr{T}$ and $P' : \mathscr{A}' \to \mathscr{T}'$. The informal intuition is that the functor T is a theory $T\Gamma$ parameterised by a generic 'future extension' Γ to the theory, and the functor $M : \mathscr{A} \to \mathscr{A}'$ is then a model of T that parameterised by a generic model of Γ, so we call the lifting M a *model transformers* of T. A number of universal constructions and concrete constructions of model transformers are then given.

The theory of this chapter applies to not just higher-order algebraic effects, but to modularity of denotational semantics in general. Although the results in this chapter are preliminary, the author finds them interesting and encouraging.

1.5∗3 (Part II). In Chapter 4, we first study the meta-theory of a logical framework originally due to Sterling [2021] in some detail, which will be a powerful tool for us to study programming languages for higher-order algebraic effects later chapters. The main result of this chapter is a categorical equivalence

$$S\text{-Mod}(\mathscr{C}) \cong \mathrm{LCCC}_{\cong}(\text{Jdg}\, S, \mathscr{C})$$

for every signature S defined in this logical framework, where S-Mod$(\mathscr{C})$ is the category of models of S in a locally cartesian closed (LCC) category $\mathscr{C}$ and $\mathrm{LCCC}_{\cong}(\text{Jdg}\, S, \mathscr{C})$ is the category of LCC-functors from an LCC category Jdg S to $\mathscr{C}$ and natural isomorphisms between these functors.

In Chapter 5, we present $\mathrm{F}_{\omega}^{\mathrm{ha}}$, an extension of Girard's [1972] System F_{ω} with (a special case of) higher-order algebraic effects. The language is intended to be an

idealised model of functional programming languages nowadays extended with higher-order algebraic effects, so there is no full dependent types, in particular equality types, in the language, forcing us to consider only *lawless* algebraic theories in this language. This degeneracy makes the language somewhat detached from the categorical foundations developed in previous chapters, but it also makes this language interesting: even if the handlers in this language are only lawless 'raw monads', the computation judgements still satisfy the monadic laws strictly. For its meta-theoretic properties, we show its consistency by a realizability model and its closed-term canonicity using *synthetic Tait computability*. A consequence of these two results is that for every closed term $t : \mathit{bool}$ in $\mathrm{F}_{\omega}^{\mathrm{ha}}$, there is a halting Turing machine that halts with 0 or 1 iff t is judgementally equal to tt or ff in $\mathrm{F}_{\omega}^{\mathrm{ha}}$. Moreover, we show how $\mathrm{F}_{\omega}^{\mathrm{ha}}$ can be extended with *general recursion* and give it a denotational model using *synthetic domain theory*.

1.5∗4. Chapter 2 and 3 are a revision and generalisation of the materials in the paper [Yang and Wu 2023]. The other chapters are new materials not published elsewhere. A complete list of papers by the author on (higher-order) algebraic effects published during his PhD is as follows:

1. Sam Lindley, Cristina Matache, Sean Moss, Sam Staton, Nicolas Wu, and Zhixuan Yang. *Scoped Effects as Parameterized Algebraic Theories*. ESOP 2024. doi:10.1007/978-3-031-57262-31.
2. Donnacha Oisín Kidney, Zhixuan Yang, and Nicolas Wu. *Algebraic Effects Meet Hoare Logic in Cubical Agda*. POPL 2024. doi:10.1145/3632898.
3. Zhixuan Yang and Nicolas Wu. *Modular Models of Monoids with Operations*. ICFP 2023. doi:10.1145/3607850.
4. Zhixuan Yang, Marco Paviotti, Nicolas Wu, Birthe van den Berg, and Tom Schrijvers. *Structured Handling of Scoped Effects*. ESOP 2022. doi:10.1007/978-3-030-99336-817.
5. Zhixuan Yang and Nicolas Wu. *Fantastic Morphisms and Where to Find Them: A Guide to Recursion Schemes*. MPC 2022. doi:10.1007/978-3-031-16912-09.
6. Zhixuan Yang and Nicolas Wu. *Reasoning about Effect Interaction by Fusion*. ICFP 2021. doi:10.1145/3473578.

1.6 How to Read This Thesis

1.6∗1. As the reader must have noticed, the paragraphs in this thesis are arranged into numbered blocks, so that their visual structure matches their semantic structure. There are exactly two counters used throughout the thesis: numbers that do not contain '∗', such as (1.1), are used exclusively for equations and

are always surrounded by parentheses. Numbers containing '$*$' are used for everything else, such as paragraphs, lemmas, theorems, and a number $x.y{*}z$ means the z-th block in Section $x.y$.

1.6∗2. This thesis is written with the assumption that the reader is comfortable with basic category theory: categories, functors, natural transformations, (co)limits, exponentials, adjunctions, monads, (co)ends. Any reasonable introductory textbook on category theory covers all these concepts; the author's favourites are Mac Lane's [1998] classic textbook and Borceux's [1994a; 1994b; 1994c] comprehensive and extremely readable handbook. When we occasionally use some not-so-basic categorical concepts, I will try to be self-contained – I apologise in advance if I failed to do so.

It is also assumed that the reader is knowledgeable in programming language theory, including λ-calculus and its connection with cartesian closed categories, dependent type theories [Angiuli and Gratzer 2024; Nordström et al. 1990], and their interpretation in presheaf categories [Hofmann 1997; Jacobs 1999]. Some basic familiarity with topos theory will also be valuable for Chapter 5.

1.6∗3 (Foundation). We will use a traditional set-theoretic language to discuss category theory, so when we say 'a category $\mathscr{C}$', we mean a *set* of objects $\mathrm{Obj}\,\mathscr{C}$ and a family of *sets* $\mathscr{C}(a,b)$ of morphisms for all $a,b \in \mathrm{Obj}\,\mathscr{C}$ equipped with identity morphisms $id_a \in \mathscr{C}(a,a)$ and composition $(- \cdot -) : \mathscr{C}(b,c) \times \mathscr{C}(a,b) \to \mathscr{C}(a,c)$. Size issues are dealt with using *Grothendieck universes*, which are roughly sets of (smaller) sets and closed under the usual set-forming operations [Shulman 2008]. Given any Grothendieck universe U, a set A, and a category $\mathscr{C}$,

- A is called *U-small* if $A \in U$,
- $\mathscr{C}$ is called *locally U-small* if $\mathscr{C}(a,b) \in U$ for all $a,b \in \mathrm{Obj}\,\mathscr{C}$,
- $\mathscr{C}$ is called *globally U-small* if $\mathrm{Obj}\,\mathscr{C} \in U$, and
- $\mathscr{C}$ is called *U-small* if it is both locally and globally U-small.

We assume that there are two (arbitrary) Grothendieck universes U_1 and U_2 such that $U_1 \in U_2$. Moreover, U_1-small sets will just be called *small sets*, and the category of small sets is denoted by Set; U_1-small categories will be called *small categories*, and their category will be denoted by Cat. Similarly, U_2-small categories will be called *large categories*, and their category is denoted by CAT.

Part I

Structure

Chapter 2

Algebraic Theories of Monoids with Operations

2∗1. In this chapter we carry out the plan of *higher-order algebraic effects*, or more accurately but less memorably *computational effects as algebraic theories of monoids with operations* that we motivated in Section 1.4. Concretely, we develop a categorical framework, as well as a syntactic language, for presenting algebraic theories over monoidal categories. The structure of this chapter is as follows:

- In Section 2.1, we first review the concept of *monoids in monoidal categories* and some examples relevant in the treatment of computational effects.
- In Section 2.2, we develop Fiore and Hur's [2009] *equational systems*, a convenient framework for presenting algebraic theories. In particular, we study morphisms between equational systems and their colimits, allowing us to construct equational systems by 'gluing' smaller ones.
- In Section 2.3, we develop a type theory for describing constructions and equational systems over monoidal categories.
- With all the machinery in hand, in Section 2.4 we come back to theories of monoids with operations and see some concrete examples.
- Finally, in Section 2.5 we classify operations on monoids into finer families and study the connections between these families.

2∗2 Remark. Although the framework that we will develop in this chapter does allow us to present strictly more monads than the classical framework of algebraic effects (enriched algebraic theories over locally κ-presentable categories), this is not the main point of higher-order algebraic effects. Instead, the point is to present *more operations*: even if (the initial algebra of) a theory (of monads with operations) over $\textsc{Endo}(\mathscr{C})$ and (the free algebra monad of) a theory over $\mathscr{C}$ may generate the same monad, the associated operations on the monad can be different, and crucially, the notion of models/handlers can be different.

2∗3 (Bibliographical note). The idea of considering theories of *monads with operations* was already hinted in Moggi's [1989a] technical report, and he showed how *algebraic operations* can be lifted along monad transformers. Inspired by

Moggi's work, Liang et al. [1995] implemented theories of monads with operations as typeclasses in a functional programming language, which later inspired the widely used Haskell library MTL. On the theory side, there does not seem to be a lot of work after Moggi, except that Jaskelioff's [2009] construction of lifting *first-order operations* (called *scoped operations* in this thesis) along a special class of monad transformers. Jaskelioff's construction was clarified by Jaskelioff and Moggi [2010] and generalised to monoid transformers.

Although equations of operations are sometimes considered in the above line of work, it was Plotkin and Power's [2002] insight that many monads that we use to model computational effects can be generated from some algebraic theories of operations and equations. In the same way, several examples of monad transformers in practice are determined by certain combinations of algebraic theories [Hyland et al. 2006]. In this line of work, non-algebraic operations, such as exception catching and parallel composition, are treated as fundamentally different from algebraic operations. Quoting Plotkin and Power [2003],

> Of the various operations, *handle* is of a different computational character and, although natural, it is not algebraic. Andrzej Filinski (personal communication) describes *handle* as a deconstructor, whereas the other operations are constructors (of effects).

This view naturally led to Plotkin and Pretnar's [2009; 2013] treatment of exception handling as a model of the algebraic theory of exception throwing, rather than an operation in the theory of exception, and led to their introduction of effect handlers, which internalises homomorphisms out of free models of algebraic theories as a programming language construct.

Later, Wu et al. [2014] argues that the treatment of non-algebraic operations as handlers poses certain non-composability problems in programming, which we will also discuss in 2.4∗12. They proposed two solutions in Haskell: *the bracketing approach* and the more general *higher-order syntax approach*.

The bracketing approaches uses two algebraic operations *begin* and *end* to delimit the scope of an operation, much like how the commands `\begin` and `\end` of LaTeX work under the hood. The category theory of this solution is later developed by Piróg et al. [2018] and is shown to be related to a variant of parameterised algebraic theories [Lindley et al. 2024].

On the other hand, the second solution of Wu et al. [2014] directly constructs the monad of programs with scoped operations as the initial algebra of a higher-order endofunctor. This solution is later developed by Yang et al. [2022], Yang and Wu [2023], and van den Berg and Schrijvers [2024].

The present chapter is a revised version of Sections 2-4 of the paper [Yang and Wu 2023], which aims to generalise and develop a formal categorical foundation for Wu et al.'s [2014] idea of constructing the monad of effectful programs as the

initial algebra of a higher-order endofunctor. The resulting theory is what we call *higher-order algebraic effects*. I apologise to the reader that this chapter somewhat lacks concrete examples of higher-order algebraic effects. The reader can find more examples in the cited papers in the last paragraph, especially the paper by van den Berg and Schrijvers [2024].

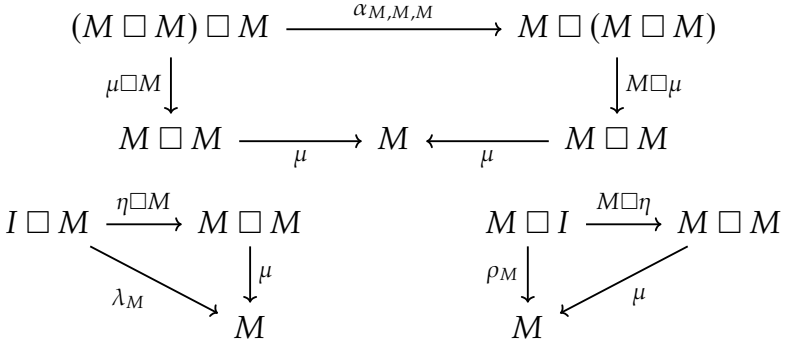
2.1 Notions of Computation as Monoids in Monoidal Categories

2.1∗1. A *monoidal category* is a category $\mathscr{C}$ equipped with a functor $\Box : \mathscr{C} \times \mathscr{C} \to \mathscr{C}$, called the *monoidal product* (and sometimes the *tensor product*), an object $I \in \mathscr{C}$, called the *monoidal unit*, and three natural isomorphisms

$$\alpha_{A,B,C} : A \Box (B \Box C) \cong (A \Box B) \Box C, \qquad \lambda_A : I \Box A \cong A, \qquad \rho_A : A \Box I \cong A$$

satisfying some coherence axioms [Mac Lane 1998, §VII.1] that guarantee there is a unique way using α, λ, and ρ to re-bracket an expression involving monoidal products to another [Mac Lane 1998, §VII.2] . A monoidal category is *strict* if the isomorphisms are identity morphisms. A monoidal category is *(left) closed* if all functors $- \Box A$ have right adjoints $-/A : \mathscr{C} \to \mathscr{C}$.

2.1∗2. A *monoid* $\langle M, \mu, \eta \rangle$ in a monoidal category $\mathscr{C}$ is an object $M \in \mathscr{C}$ equipped with two morphisms: a *multiplication* $\mu : M \Box M \to M$ and a *unit* $\eta : I \to M$ making the following diagrams commute:



2.1∗3. In the following until Section 2.3, we present a collection of examples of monoidal categories, in which monoids model different flavours of *notions of computations*, and will serve as the main application of the theory developed in this thesis. The main messages are that

1. monoids in monoidal categories are an expressive abstraction that unifies different notions of computations;

2. by imposing suitable conditions, these monoidal categories for notions of computations can be made closed and (co)complete, so are suitable for doing algebraic theories over them later.

Some of the following examples are quite technical and my description is sketchy, but the reader does not need to fully understand the details in the examples to proceed to Section 2.3.

2.1.1 Monads and Size Issues

2.1.1∗1 (Monads). The category $\textsc{Endo}(\mathscr{C})$ of endofunctors $\mathscr{C} \to \mathscr{C}$ on a category $\mathscr{C}$ can be turned into a monoidal category by equipping it with functor composition $F \circ G$ as the monoidal product and the identity functor $\mathrm{Id} : \mathscr{C} \to \mathscr{C}$ as the unit. Monoids in this category are called *monads* on $\mathscr{C}$, and they are used to model *computational effects*, also called *notions of computation*, in programming languages [Moggi 1989a, 1991], where the unit $\eta : \mathrm{Id} \to M$ is understood as embedding *pure values* into computations, and the multiplication $\mu : M \circ M \to M$ is understood as flattening *computations of computations* into computations by sequentially executing them. The understanding of μ as sequential composition is better exhibited by the following co-Yoneda isomorphism:

$$(F \circ G)A = F(GA) \cong \int^{X \in \mathscr{C}} \coprod_{\mathscr{C}(X,GA)} FX$$

where $\int^X$ denotes a coend and $\coprod_{\mathscr{C}(X,GA)}$ denotes a $\mathscr{C}(X, GA)$-fold coproduct. The informal reading of the coend is that FX is the first computation, returning a value of type X, and the second computation is determined by the result of FX, given as a function $X \to GA$. So $\mu : M \circ M \to M$ is sequential composition of two computations in which the second is determined by the first one.

2.1.1∗2. However, the category $\textsc{Endo}(\mathscr{C})$ is usually not as well behaved as we would like for doing algebraic theories in it, even when $\mathscr{C}$ itself is a very nice category such as $\textsc{Set}$. In particular, $\textsc{Endo}(\textsc{Set})$ is not closed with respect to either cartesian products or functor composition. Moreover, $\textsc{Endo}(\textsc{Set})$ is not a nice category for doing algebraic theories; for example, some objects in $\textsc{Endo}(\textsc{Set})$, such as the covariant powerset functor $\mathscr{P}$, do not have free monads over them.

These issues about $\textsc{Endo}(\textsc{Set})$ are fundamentally related to *sizes* of sets. For instance, if $\mathscr{P} : \textsc{Set} \to \textsc{Set}$ were to have a free monad F, then F would also be *algebraically free* [nLab 2024a, Theorem 3.2], so $F\emptyset$ would carry the initial $\mathscr{P}$-algebra. By Lambek's lemma [Lambek 1968], we would then have a set $F\emptyset$ satisfying $F\emptyset \cong \mathscr{P}(F\emptyset)$, contradicting Cantor's theorem [nLab 2024a].

There are two ways to rectify the size issues: (1) we can consider some 'extremely nice' categories $\mathscr{C}$, namely, *(small-) complete small categories*, or (2) we can restrict our attention to 'reasonably nice endofunctors' on a 'reasonably nice' category $\mathscr{C}$, namely, *κ-accessible functors* on *locally κ-presentable categories*. We will describe both approaches below.

2.1.1∗3. When $\mathscr{C}$ is a small category that is also small-complete, the monoidal structure $\langle \circ, \mathrm{Id} \rangle$ on $\textsc{Endo}(\mathscr{C})$ is closed, with the right adjoint $-/G$ to $- \circ G$ for every $G : \mathscr{C} \to \mathscr{C}$ given by *right Kan extension* [Mac Lane 1998, §X]:

$$(F/G)A = \int_{B \in \mathscr{C}} \prod_{\textsc{Set}(A,GB)} FB, \tag{2.1}$$

which exists since the 'bound of the end' $B \in \mathscr{C}$ is small and $\mathscr{C}$ is small-complete. The unit of the adjunction $- \circ G \dashv -/G$ is given by the natural transformations $\eta_F : F \to (F \circ G)/G$ whose component at every $A \in \mathscr{C}$

$$\eta_{F,A} : FA \to \int_{B \in \mathscr{C}} \prod_{\textsc{Set}(A,GB)} F(GB)$$

is the mediating morphism induced by the wedge

$$\eta_{F,A,B} := \langle Fk \rangle_{k \in \textsc{Set}(A,GB)} : FA \to \prod_{\textsc{Set}(A,GB)} F(GB)$$

for all $B \in \mathscr{C}$. The counit ϵ of the adjunction $- \circ G \dashv -/G$ is given by the composite

$$((F/G) \circ G)A = \int_{B \in \mathscr{C}} \prod_{\textsc{Set}(GA,GB)} FB \xrightarrow{\pi_A} \prod_{\textsc{Set}(GA,GA)} FA \xrightarrow{\pi_{id:GA \to GA}} FA$$

for all $F \in \textsc{Endo}(\mathscr{C})$ and $A \in \mathscr{C}$.

Moreover, $\textsc{Endo}(\mathscr{C})$ is also a small-complete small category: it is small because its domain and codomain categories $\mathscr{C}$ are both small; it is small-complete because limits in functor categories are computed pointwise and $\mathscr{C}$ is small-complete. We will see later in Theorem 2.2.1∗19 that it implies that all objects in $\textsc{Endo}(\mathscr{C})$ have free monads (and many other free algebraic structures).

2.1.1∗4. However, it is long known that *in classical logic* the only examples of small-complete small categories $\mathscr{C}$ are *complete preorders* [nLab 2024b]. Therefore $\mathscr{C}$ being small and small-complete is classically too strong a requirement for the purpose of modelling computational effects.

However, a stunning result in categorical logic is that if we carry out category theory internally in the *effective topos* Eff, or more generally realizability toposes [Hyland 1988; Oosten 2008], the category Mset of *modest sets*, also known as *partial equivalence relations* (PERs), is a non-trivial small-complete small-categories, internally. The effective topos and modest sets find many application in programming language theory: for one, they provide semantics for type theories with *impredicative* polymorphism, such as System F [Girard 1986] and the calculus of inductive constructions [Coquand and Huet 1988; Luo 1994] with an impredicative universe of sets, which is the type theory underlying the Rocq (previously known as Coq) proof assistant with the `-impredicative-set` option. Later in Chapter 5, we will also use realizability to model programming languages with higher-order algebraic effects and impredicative polymorphism.

2.1.1∗5 Remark. Mathematics carried out internally in a category can also be

externalised to the ambient meta-theory [Jacobs 1999; Phoa 1992; Streicher 2023], telling us what an internal construction 'really means' from the external point of view. In particular, if we externalise the internal category $\textsc{Endo}(\textsc{Mset})$, we obtain a fibration $[\textsc{Mset}] \to \textsc{Eff}$; taking its fiber over $1 \in \textsc{Eff}$, we obtain an ordinary category of *realizable endofunctors* $\textsc{Endo}_r(\textsc{Mset})$, which are roughly endofunctors whose mapping on morphisms are realised by Turing machines; see Jaskelioff and Moggi [2010, Example 2.20] or Bainbridge et al. [1990] for more information.

2.1.2 Finitary and Accessible Monads

2.1.2∗1. We have seen the approach of rectifying the size issues in $\textsc{Endo}(\mathscr{C})$ by assuming an 'extremely nice' $\mathscr{C}$ that exists only in non-classical settings. Now let us describe an alternative, classically valid, approach – restricting our attention to 'reasonably nice' endofunctors.

First we observe that the reason the formula (2.1) does not work for arbitrary endofunctors $F, G : \textsc{Set} \to \textsc{Set}$ is that the bound of the end $B \in \textsc{Set}$ would be large while $\textsc{Set}$ is only small-complete. Therefore the idea is to consider functors that allow us to cut $B \in \textsc{Set}$ to a small bound.

An endofunctor $F \in \textsc{Endo}(\textsc{Set})$ is called *finitary* if it preserves *filtered colimits*. An informal description of finitariness is that *we lose no information if we restrict F to finite sets*. Precisely, F is finitary if we first restrict F to $F \circ V : \textsc{Fin} \to \textsc{Set}$ on the full subcategory of finite sets, where $V : \textsc{Fin} \to \textsc{Set}$ is the inclusion functor, and then take the left Kan extension of $F \circ V$ along V, the resulting functor is still isomorphic to F. In other words, we have the following equivalence, where $\textsc{Endo}_f(\textsc{Set}) \subseteq \textsc{Endo}(\textsc{Set})$ denotes the full subcategory of finitary endofunctors:

$$\textsc{Endo}_f(\textsc{Set}) \underset{-\circ V}{\overset{\textsc{Lan}_V - \;\cong}{\leftarrow\!\!\rightarrow}} \textsc{Set}^{\textsc{Fin}} \tag{2.2}$$

2.1.2∗2. The category $\textsc{Endo}_f(\textsc{Set})$ inherits the monoidal structure $\langle \circ, \mathrm{Id} \rangle$ of $\textsc{Endo}(\textsc{Set})$, which under (2.2) is equivalent to $\langle \bullet, V \rangle$ on $\textsc{Set}^{\textsc{Fin}}$ [Fiore et al. 1999; Kelly and Power 1993] where

$$Vn = n \qquad \text{and} \qquad (F \bullet G)n = \int^{m \in \textsc{Fin}} Fm \times (Gn)^m. \tag{2.3}$$

For every $G \in \textsc{Set}^{\textsc{Fin}}$, the functor $- \bullet G$ has a right adjoint

$$(F/G)n = \int_{m \in \textsc{Fin}} \prod_{\textsc{Set}(n, Gm)} Fm$$

with unit and counit similar to those in 2.1.1∗3. The end $\int_{m \in \textsc{Fin}}$ exists because the category $\textsc{Fin}$ of finite sets is essentially small and $\textsc{Set}$ is small complete.

Moreover, the functor $\circ : \textsc{Endo}_f(\textsc{Set}) \times \textsc{Endo}_f(\textsc{Set}) \to \textsc{Endo}_f(\textsc{Set})$ is also finitary, i.e. preserving filtered colimits in $\textsc{Endo}_f(\textsc{Set}) \times \textsc{Endo}_f(\textsc{Set})$. This is

equivalent to preserving filtered colimits in both of its arguments separately: functor composition preserves (all) colimits in the first argument

$$((\mathrm{colim}_i F_i) \circ G)n = (\mathrm{colim}_i F_i)(Gn) \cong \mathrm{colim}_i (F_i(Gn))$$

because colimits of functors can be computed pointwise. It preserves filtered colimits in the second argument by the finitariness of its first argument:

$$(F \circ (\mathrm{colim}_i G_i))n = F((\mathrm{colim}_i G_i)n) \cong F(\mathrm{colim}_i G_i n) \cong \mathrm{colim}_i F(G_i n)$$

In particular, the diagram of an ω-chain is filtered, so the functor $\circ$ preserves colimits of ω-chains, a property we will use to construct free monads over finitary endofunctors in Section 2.2.1.

2.1.2∗3. Monoids in $\langle \textsc{Endo}_f(\textsc{Set}), \circ, \mathrm{Id} \rangle$ are called *finitary monads* on Set, and they are equivalent to (finitary) *Lawvere theories* [Power 1999, Theorem 4.2; Adamek et al. 2010, Theorem A.38] and the closely related *abstract clones* [Cohn 1981, page 132; Fiore et al. 1999, Section 3]. Computational effects modelled by Lawvere theories are usually called *algebraic effects* [Plotkin and Power 2002, 2004].

Apart from modelling computational effects, a related but slightly different application of monoids in $\textsc{Endo}_f(\textsc{Set})$, or equivalently $\textsc{Set}^{\textsc{Fin}}$, is modelling *abstract syntax with variable binding* [Fiore and Szamozvancev 2022; Fiore et al. 1999]. In this case, a monoid $M \in \textsc{Set}^{\textsc{Fin}}$ is understood as a family of sets of terms indexed by the number of *variables* in the context. The monoid unit $V \to M$ is then embedding *variables* as M-terms, and the monoid multiplication $M \bullet M \to M$ is *simultaneous substitution* of terms for variables.

Yet another interesting reading of monoids of $\langle \bullet, V \rangle$ due to Fiore and Staton [2014] is computations supporting (i) binding a piece of code to a code pointer and (ii) jumping to a code pointer. Based on this reading, the monoidal category $\langle \textsc{Set}^{\textsc{Fin}}, \bullet, V \rangle$ provides an adequate denotational semantics of a calculus of substitution/jumps, on which algebraic effects can be encoded.

2.1.2∗4. The adjunction $\textsc{Endo}_f(\textsc{Set}) \cong \textsc{Set}^{\textsc{Fin}}$ can be generalised to endofunctors on categories other than Set: we can replace (1) Set with any *locally κ-presentable* (lκp) category $\mathscr{C}$ for a regular cardinal κ, (2) Fin with the subcategory $\mathscr{C}_\kappa$ of *κ-presentable objects* in $\mathscr{C}$, and (3) finitary functors with *κ-accessible* functors $\textsc{Endo}_\kappa(\mathscr{C})$ [Adámek and Rosicky 1994]. This results in a cocomplete closed monoidal category $\langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$, on which $\circ$ is κ-accessible. The κ-accessibility $\circ$ implies that there is a large enough limit ordinal α such that $\circ$ preserves colimits of all α-chains, a property we will later use for constructing free monoids.

2.1.2∗5. All lκp categories are necessarily (small-) complete and cocomplete, and they cover a wide range of categories that are used for modelling programming

languages. Examples of lκp categories include:

* all presheaf categories $\textsc{Set}^{\mathscr{D}}$ for small categories $\mathscr{D}$ and $\kappa = \aleph_0$ (an lκp category is called locally *finitely* presentable when $\kappa = \aleph_0$), and more generally all Grothendieck toposes for some κ [Borceux 1994c, 3.4.16];
* categories of models for *essentially algebraic theories* [Adámek and Rosicky 1994, §3.D], which include the category $\textsc{Cat}$ of small categories for $\kappa = \aleph_0$, the category $\omega\textsc{Cpo}$ of ω-complete partial orders for $\kappa = \aleph_1$ (however, the category of *directed* complete partial orders is not lκp for any κ);
* functor categories $\mathscr{C}^{\mathscr{D}}$ for small categories $\mathscr{D}$ and lκp $\mathscr{C}$;
* moreover, when $\mathscr{C}$ is lκp, it is automatically lλp for any $\lambda > \kappa$.

Therefore, lκp categories provide a nice setting for algebraic theories in the context of programming language semantics.

2.1.2∗6 Remark. Let $\kappa < \lambda$ be two regular cardinals and $\mathscr{C}$ be an lκp category. As we mentioned above $\mathscr{C}$ is also lλp, and every κ-accessible endofunctor on $\mathscr{C}$ is also going to be λ-accessible because every λ-filtered colimit is also κ-filtered. The monoidal structure $\langle \circ, \mathrm{Id} \rangle$ of functor composition in $\textsc{Endo}_\kappa(\mathscr{C})$ coincides with $\langle \circ, \mathrm{Id} \rangle$ in $\textsc{Endo}_\lambda(\mathscr{C})$, so they can be both denoted by $\circ$ unproblematically. However, the closed structure F/G in $\textsc{Endo}_\kappa(\mathscr{C})$, given by

$$(F/G)n = \int_{x \in \mathscr{C}_\kappa} \prod \mathscr{C}(n, Gx).\, Fx$$

in fact depends on κ and has its universal property with respect to only κ-accessible functors, so it may not coincide with the closed structure in $\textsc{Endo}_\lambda(\mathscr{C})$. In the same way, F/G computed in $\textsc{Endo}_\kappa(\mathscr{C})$ does not have to be the right Kan extension of F along G among the bigger category $\textsc{Endo}(\mathscr{C})$ of all endofunctors or the category $\textsc{Endo}_{acc}(\mathscr{C})$ of all accessible endofunctors, in which every object may have a different choice of κ. Therefore if we were pedantic about the notation, we should write $F/_\kappa G$ instead of just F/G.

2.1.3 Strong Monads

2.1.3∗1. The multiplication $\mu : M \circ M \to M$ of a monad $M : \mathscr{C} \to \mathscr{C}$ allows one to compose two effectful computations $f : A \to MB$ and $g : B \to MC$ by

$$A \xrightarrow{f} MB \xrightarrow{Mg} M(MC) \xrightarrow{\mu_C} MC.$$

However, to give semantics to effectful programming languages with a *structural* context of variables as done by Moggi [1989a, 1991], what we need is slightly stronger: for all objects $\Gamma \in \mathscr{C}$ (thought of as variable contexts) and morphisms

$f : \Gamma \times A \to MB$ and $g : \Gamma \times B \to MC$ (two computations under a context Γ), we would like to have a morphism $\Gamma \times A \to MC$ (the sequential composition of f and g). The structure of monads $\langle M, \mu, \eta \rangle$ is not sufficient for doing this, and we need additionally a natural transformation

$$s_{\Gamma,B} : \Gamma \times MB \to M(\Gamma \times B),$$

with which we have the composite

$$\Gamma \times A \xrightarrow{\langle \pi_1, f \rangle} \Gamma \times MB \xrightarrow{s_{\Gamma,B}} M(\Gamma \times B) \xrightarrow{Mg} M(MC) \xrightarrow{\mu_C} MC.$$

To make this way of composing effectful computations associative and the pure computation $(\eta_A \cdot \pi_2) : \Gamma \times A \to MA$ an identity, the natural transformation s must satisfy certain coherence conditions (see [Moggi 1991, Definition 3.2]). The natural transformation s is called a *strength* for the monad M, and the tuple $\langle M, \mu, \eta, s \rangle$ is called a *strong monad*.

2.1.3∗2. Strong monads are monoids in the monoidal category of *strong endofunctors* and composition. A strong endofunctor $\langle F, s \rangle$ on a category $\mathscr{C}$ with finite products is a functor $F : \mathscr{C} \to \mathscr{C}$ with a natural transformation $s_{\Gamma,B} : \Gamma \times FB \to F(\Gamma \times B)$ making the following diagrams commute:

$$\begin{array}{ccc} 1 \times FB & & \\ {\scriptstyle s_{1,B}}\downarrow & \searrow^{\lambda_{FB}} & \\ F(1 \times B) & \xrightarrow[F\lambda_B]{} & FB \end{array} \tag{2.4}$$

$$\begin{array}{ccccc} (\Gamma' \times \Gamma) \times FB & \xrightarrow{\alpha_{\Gamma',\Gamma,FB}} & \Gamma' \times (\Gamma \times FB) & \xrightarrow{\Gamma' \times s_{\Gamma,B}} & \Gamma' \times F(\Gamma \times B) \\ {\scriptstyle s_{\Gamma' \times \Gamma, B}}\downarrow & & & & \downarrow{\scriptstyle s_{\Gamma',\Gamma \times B}} \\ F((\Gamma' \times \Gamma) \times B) & & \xrightarrow[F\alpha_{\Gamma',\Gamma,B}]{} & & F(\Gamma' \times (\Gamma \times B)) \end{array} \tag{2.5}$$

where λ and α are the left unitor and associator for the cartesian monoidal structure $\langle \times, 1 \rangle$. Moreover, *strong natural transformations* between $\langle F, s^F \rangle$ and $\langle G, s^G \rangle$ are natural transformations $\tau : F \to G$ making the following commute:

$$\begin{array}{ccc} \Gamma \times FB & \xrightarrow{\Gamma \times \tau_B} & \Gamma \times GB \\ {\scriptstyle s^F_{\Gamma,B}}\downarrow & & \downarrow{\scriptstyle s^G_{\Gamma,B}} \\ F(\Gamma \times B) & \xrightarrow[\tau_{\Gamma \times B}]{} & G(\Gamma \times B) \end{array}$$

2.1.3∗3. Strong endofunctors on $\mathscr{C}$ and strong natural transformations can be collected into a category $\textsc{Endo}_s(\mathscr{C})$. The category $\textsc{Endo}_s(\mathscr{C})$ has a monoidal structure $\langle \circ_s, \mathrm{Id}_s \rangle$ where Id_s is the identity functor equipped with the identity

strength, and $\circ_s$ is the composition of strong functors:

$$\langle F, s^G\rangle \circ_s \langle G, s^F\rangle = \langle F \circ G,\ (Fs^G_{\Gamma,B} \cdot s^F_{\Gamma,GB})_{\Gamma,B\in\mathscr{C}}\rangle.$$

Strong monads on $\mathscr{C}$ are precisely monoids in this monoidal category. When $\mathscr{C}$ is cartesian closed, strong functors are the same as $\mathscr{C}$*-enriched* functors [Kock 1972; McDermott and Uustalu 2022b].

2.1.3∗4. When $\mathscr{C}$ is Set, or slightly more generally a full subcategory of Set closed under finite products of Set, every $F : \mathscr{C} \to \mathscr{C}$ has a strength:

$$\begin{aligned} &s_{\Gamma,B} : \Gamma \times FB \to F(\Gamma \times B) \\ &s_{\Gamma,B}\ \langle \gamma, f\rangle = F\ (\lambda b.\ \langle \gamma, b\rangle)\ f \end{aligned} \tag{2.6}$$

which can be readily checked to satisfy the laws of strengths (2.4, 2.5). In fact, this is the only strength for F: let $t_{\Gamma,B} : \Gamma \times FB \to F(\Gamma \times B)$ be any strength for F, for all $\gamma \in \Gamma$ and $f \in FB$, the naturality of t implies the commutativity of

$$\begin{array}{ccc} 1 \times FB & \xrightarrow{t_{1,B}} & F(1 \times B) \\ {\scriptstyle \gamma\times FB}\downarrow & & \downarrow{\scriptstyle F(\gamma\times B)} \\ \Gamma \times FB & \xrightarrow[t_{\Gamma,B}]{} & F(\Gamma \times B) \end{array}$$

Evaluating the two paths at $\langle *, f\rangle \in 1 \times FB$, we have

$$t_{\Gamma,B}\langle \gamma, f\rangle = F\ (\gamma \times B)\ (t_{1,B}\ \langle *, f\rangle).$$

However, by the law (2.4), $t_{1,B}\ \langle *, f\rangle$ has to be $F\lambda_B^{-1}\langle *, f\rangle$. Hence $t_{\Gamma,B}\langle \gamma, f\rangle$ is equal to the canonical strength s (2.6) above. However, functors on a general category $\mathscr{C}$ may have no or non-unique strengths; see McDermott and Uustalu [2022b] for some counter-examples.

2.1.3∗5. As a side remark to readers who know fibred category theory, the fact that functors on sets are canonically strong is not so much about sets, but more about the ability to use the value γ in the current 'context of variables' when defining $(\lambda b.\ \langle \gamma, b\rangle) : B \to \Gamma \times B$ in (2.6). Indeed, for every $\mathscr{C}$ with finite limits and a fibred endofunctor $F : \mathscr{C}^{\to} \to \mathscr{C}^{\to}$ on the fundamental fibration $\mathrm{Cod} : \mathscr{C}^{\to} \to \mathscr{C}$, the restriction of F to the fiber over $1 \in \mathscr{C}$ induces a functor $\overline{F} : \mathscr{C} \cong \mathscr{C}_1^{\to} \to \mathscr{C}_1^{\to} \cong \mathscr{C}$. The functor $\overline{F}$ has a strength similar to (2.6): firstly we have the following morphism in the fiber over every $\Gamma \in \mathscr{C}$,

$$\begin{array}{ccc} \Gamma \times B & \xrightarrow{\langle \pi_1, id\rangle} & \Gamma \times (\Gamma \times B) \\ & {\scriptstyle \pi_1}\searrow \quad \swarrow{\scriptstyle \pi_1} & \\ & \Gamma & \end{array}$$

which plays the same role as $(\lambda b.\ \langle \gamma, b\rangle) : B \to \Gamma \times B$ in (2.6), and then this morphism is mapped by the fibred functor F to a morphism still over Γ:

$$\begin{array}{ccc} F(\Gamma \times B) & \xrightarrow{F\langle \pi_1, id\rangle} & F(\Gamma \times (\Gamma \times B)) \\ & \searrow \quad \swarrow & \\ & \Gamma & \end{array} \tag{2.7}$$

Because $\pi_1 : \Gamma \times B \to \Gamma$ is the pullback of $B \to 1$ along $\Gamma \to 1$, and F as a fibred functor preserves pullbacks, $F(\Gamma \times B) \to \Gamma$ is also a pullback of $FB \to 1$ (i.e. $\overline{F}B \to 1$) along $\Gamma \to 1$. Another choice of the pullback of $FB \to 1$ along $\Gamma \to 1$ is just $\pi_1 : \Gamma \times \overline{F}B \to \Gamma$. Hence there is a canonical isomorphism $F(\Gamma \times B) \cong \Gamma \times \overline{F}B$, and similarly $F(\Gamma \times (\Gamma \times B)) \cong \Gamma \times \overline{F}(\Gamma \times B)$, so the morphism $F\langle \pi_1, id\rangle$ in (2.7) gives us a morphism $\Gamma \times \overline{F}B \to \overline{F}(\Gamma \times B)$. This can be checked to be a strength for $\overline{F}$. However, from the other direction we cannot necessarily obtain a fibred endofunctor on $\textsc{Cod} : \mathscr{C}^{\to} \to \mathscr{C}$ from a strong functor on $\mathscr{C}$ in general. Instead, strong functors on $\mathscr{C}$ are equivalent to fibred endofunctors on the *simple fibration* over $\mathscr{C}$ [Jacobs 1999, Proposition 2.6.9] (and fibred endofunctors over the fundamental fibration restrict to fibred endofunctors over the simple fibration).

2.1.3∗6. Similar to the setting of ordinary monads, to have a 'nice' monoidal category of strong monads, we need to either (1) consider small-complete small categories $\mathscr{C}$ or (2) κ-accessible strong monads. More precisely, denote by $\textsc{Endo}_{s\kappa}(\mathscr{C})$ the full subcategory of $\textsc{Endo}_s(\mathscr{C})$ that contains strong functors whose underlying functors are κ-accessible.

When $\mathscr{C}$ is *lκp as a cartesian closed category*, which means that $\mathscr{C}$ is lκp and cartesian closed, and that the κ-presentable objects of $\mathscr{C}$ are closed under finite products, the category $\textsc{Endo}_{s\kappa}(\mathscr{C})$ is also lκp and has a closed monoidal structure of functor composition and the identity functor. The primary example of such setting is $\mathscr{C} = \omega\textsc{Cpo}$ for modelling general recursion. We refer the reader to Kelly and Power [1993, §4] and Kelly [1982] for details.

2.1.4 Graded Monads

2.1.4∗1. Another generalisation of monads is to index the monad with some *grades* that track quantitative information about the effects performed by a computation [Katsumata 2014; Katsumata et al. 2022; McDermott and Uustalu 2022a]. Precisely, let $\langle \mathscr{G}, \cdot, 1\rangle$ be any small strict monoidal category, whose objects we call grades. A *$\mathscr{G}$-graded monad* on a category $\mathscr{C}$ is a functor $M : \mathscr{G} \to \textsc{Endo}(\mathscr{C})$ equipped with natural transformations

$$\eta : \mathrm{Id} \to M1 \qquad\qquad \mu_{a,b} : (Ma) \circ (Mb) \to M(a \cdot b) \tag{2.8}$$

natural in $a, b \in \mathcal{G}$, satisfying laws similar to those of monads [Katsumata 2014].

2.1.4∗2. For example, for tracking operations performed by a computation, $\mathcal{G}$ can be the poset of sets of operation names, ordered by inclusion, with the monoidal structure $1 = \emptyset$ and $a \cdot b = a \cup b$. And for tracking the number of nondeterministic choices made by a computation, $\mathcal{G}$ can be the poset $\langle \mathbb{N}, \leqslant \rangle$ with monoidal structure $\langle 0, + \rangle$.

2.1.4∗3. Again, to avoid the size issues when doing algebraic theories (2.1.1∗2), we can either require $\mathcal{C}$ to be small and small-complete or consider only *κ-accessible $\mathcal{G}$-graded monads* $M : \mathcal{G} \to \textsc{Endo}_\kappa(\mathcal{C})$ for $\mathrm{l}\kappa\mathrm{p}\ \mathcal{C}$.

2.1.4∗4. Similar to the ungraded situation, κ-accessible graded monads are equivalent to monoids in the functor category $\textsc{Endo}_\kappa(\mathcal{C})^{\mathcal{G}}$ equipped with the following monoidal structure similar to the Day tensor:

$$I = \coprod_{\mathcal{G}(1,-)} \mathrm{Id} \qquad\qquad F * G = \int^{a,b \in \mathcal{G}} \coprod_{\mathcal{G}(a \cdot b, -)} (Fa \circ Gb). \tag{2.9}$$

This monoidal product has right adjoints given by

$$(G \multimap F)a = \int_{b \in \mathcal{G}} \int_{m \in \mathcal{C}_\kappa} \prod_{\mathcal{C}(-,(Gb)m)} F(a \cdot b)m.$$

2.1.4∗5. The equivalence between κ-accessible graded monads (2.8) and monoids for the monoidal structure (2.9) seems to be unwritten folklore, so we here show a sketch of the proof using (co)end calculus [Loregian 2021]. The correspondence between $\eta : I \to M$ for the monoidal structure (2.9) and $\eta : \mathrm{Id} \to M1$ (2.8) is

$$\begin{aligned}
&\textsc{Hom}(I, M) \\
\cong\ &\{\text{We write } [A, B] \text{ for } \mathcal{C}(A, B) \text{ below}\} \\
&\textstyle\int_{c \in \mathcal{G}} \int_{x \in \mathcal{C}_\kappa} [\coprod_{\mathcal{G}(1,c)} x,\ (Mc)x] \\
\cong\ &\textstyle\int_{c \in \mathcal{G}} \int_{x \in \mathcal{C}_\kappa} [\mathcal{G}(1, c) \times x,\ (Mc)x] \\
\cong\ &\textstyle\int_{c \in \mathcal{G}} \int_{x \in \mathcal{C}_\kappa} [\mathcal{G}(1, c), [x,\ (Mc)x]] \\
\cong\ &\{[\mathcal{G}(1, c), -] \text{ preserves limits}\} \\
&\textstyle\int_{c \in \mathcal{G}} [\mathcal{G}(1, c), \int_{x \in \mathcal{C}_\kappa} [x,\ (Mc)x]] \\
\cong\ &\{\text{ Yoneda lemma }\} \\
&\textstyle\int_{x \in \mathcal{C}_\kappa} [x,\ (M1)x] \\
\cong\ &\textsc{Hom}(\mathrm{Id}, M1)
\end{aligned}$$

The correspondence between $\mu : M * M \to M$ (2.8) and families of natural transformations $\mu_{a,b} : (Ma) \circ (Mb) \to M(a \cdot b)$ (2.9) natural in $a, b \in \mathcal{G}$ is

$$\textsc{Hom}(M * M, M)$$

$$
\begin{aligned}
&\cong \int_{c\in\mathcal{G}} \textsc{Hom}((M * M)c, Mc) \\
&\cong \int_{c\in\mathcal{G}} \int_{x\in\mathcal{C}_\kappa} [\int^{a,b\in\mathcal{G}} \amalg_{\mathcal{G}(a\cdot b,-)}(Ma(Mbx)), (Mc)x] \\
&\cong \int_{c\in\mathcal{G}} \int_{x\in\mathcal{C}_\kappa} \int_{a,b\in\mathcal{G}} [\mathcal{G}(a\cdot b,-), [(Ma(Mbx)), (Mc)x]] \\
&\cong \int_{x\in\mathcal{C}_\kappa} \int_{a,b\in\mathcal{G}} \int_{c\in\mathcal{G}} [\mathcal{G}(a\cdot b,-), [(Ma(Mbx)), (Mc)x]] \\
&\cong \int_{x\in\mathcal{C}_\kappa} \int_{a,b\in\mathcal{G}} [Ma(Mbx), (M(a\cdot b))x] \\
&\cong \int_{a,b\in\mathcal{G}} \textsc{Hom}(Ma \circ Mb, M(a\cdot b))
\end{aligned}
$$

We will not torture the reader with checking the correspondence of the two versions of the laws here.

2.1.4∗6. We have seen monads and several variations – realizable, accessible, strong, graded – all formulated as monoids in monoidal categories. Despite their differences, they all model notions computations that support sequential compositions in a sense. In the following, let us have a look at some other monoids that are conceptually quite different from monads.

2.1.5 Cartesian Monoids

2.1.5∗1. Every cartesian category $\mathcal{C}$, i.e. a category with finite products, can be equipped with the product $\times$ as the monoidal product and the terminal object $1 \in \mathcal{C}$ as the monoidal unit. When $\mathcal{C}$ has all exponentials B^A, $\mathcal{C}$ is then a cartesian closed category. Particularly, the category $\textsc{Set}$ is a closed monoidal category in this way. Monoids in $\textsc{Set}$ are precisely the usual notion of monoids, such as the set of lists with concatenation and empty list.

2.1.5∗2. For $\mathrm{l}\kappa\mathrm{p}$ and cartesian closed $\mathcal{C}$, the category $\textsc{Endo}_\kappa(\mathcal{C})$ is cartesian closed. The cartesian unit and product in $\textsc{Endo}_\kappa(\mathcal{C})$ are defined pointwise:

$$1n = 1_{\mathcal{C}} \qquad\qquad (F \times G)n = Fn \times Gn$$

The exponential is given by

$$(F^G)n = \int_{m\in\mathcal{C}_\kappa} \prod_{\mathcal{C}(n,m)} (Fm)^{Gm}.$$

A computational interpretation of cartesian monoids in $\textsc{Endo}_\kappa(\mathcal{C})$ is that they model *notions of independent computations*: cartesian multiplication $M \times M \to M$ composes two computations that have no dependency and return the same type of values, whereas monad multiplication $M \circ M \to M$ composes a computation with another that depends on the result of the former.

2.1.6 Applicative Functors

2.1.6∗1. Between the two extremes of $M \circ M$ and $M \times M$, there are monoidal structures on $\textsc{Endo}_\kappa(\mathscr{C})$ that allow computations to have restricted dependency. One of them is the Day convolution [Day 1970] induced by cartesian products: the Day monoidal structure on $\textsc{Endo}_\kappa(\textsc{Set})$ has as unit the identity functor, and the monoidal product is given by the following coend formula:

$$(F * G)n = \int^{m,k \in \textsc{Set}_\kappa \times \textsc{Set}_\kappa} Fm \times Gk \times n^{(m \times k)}. \tag{2.10}$$

Informally, $F * G$ models two computations Fn and Gm that are *almost independent* except that their return values are combined by a pure function $n^{(m \times k)}$. This structure is symmetric and closed, with the right adjoint to $- * G$ given by

$$G \multimap F = \int_{n \in \textsc{Set}_\kappa} (F(- \times n))^{Gn}.$$

More generally, we can replace $\textsc{Set}$ with any *lκp as a cartesian closed* category $\mathscr{V}$ [Kelly 1982] and also the coend in (2.10) with a $\mathscr{V}$-enriched coend, which allows us to give a more accurate formulation of applicatives in functional languages with general recursion by setting $\mathscr{V} = \omega\textsc{Cpo}$. Alternatively, we can consider small-complete small $\mathscr{C}$ to work around the size issues as in 2.1.1∗3.

2.1.6∗2. Monoids for $\langle *, \mathrm{Id} \rangle$ are called *applicative functors* or simply *applicatives*, and they are equivalent to *lax monoidal functors* $\langle \textsc{Set}, \times, 1 \rangle \to \langle \textsc{Set}, \times, 1 \rangle$ [McBride and Paterson 2008; Paterson 2012]. A practical application of them is in *build systems* [Mokhov et al. 2018], since usually the result of a building task does not affect what the next building task is.

2.1.6∗3. Applicatives are a weaker notion than strong monads, as intuitively independence is a special case of dependence. Precisely, for any two functors $F, G \in \textsc{Endo}_\kappa(\textsc{Set})$, there are canonical strengths (2.6):

$$s^F_{XY} : FX \times Y \to F(X \times Y), \qquad s^G_{XY} : GX \times Y \to G(X \times Y),$$

and then there is a natural transformation $e : F * G \to F \circ G$ as follows:

$$(F{*}G)n = \int^{mk} Fm{\times}Gk{\times}n^{(m \times k)} \to \int^{mk} F(G(m{\times}k{\times}n^{(m \times k)})) \to \int^{mk} F(Gn) \cong F(Gn)$$

where the first arrow is repeated uses of the strengths of F and G, and the second arrow is function evaluation. Consequently, for any monad $\langle M, \mu, \eta \rangle$ on $\textsc{Set}$, it induces an applicative functor with unit η and multiplication

$$M * M \xrightarrow{e} M \circ M \xrightarrow{\mu} M.$$

However, there are many applicative functors that are not obtained from monads in this way [McBride and Paterson 2008; Paterson 2012].

2.1.7 Hughes Arrows

2.1.7∗1. Between applicatives and monads, there is a middle ground of notions of computations that allows *data dependency* but not *control dependency*, known as *Hughes arrows*, or simply *arrows* [Hughes 2000; Jacobs et al. 2009; Lindley et al. 2011], or *strong promonads* [Román 2022]. Unlike monads and applicatives, arrows are not monoids in $\textsc{Endo}(\mathscr{C})$, but in the category of *strong endoprofunctors*.

An *endoprofunctor* P on a small category $\mathscr{C}$ is just a functor $P : \mathscr{C}^{\mathrm{op}} \times \mathscr{C} \to \textsc{Set}$. Informally, the set $P(a, b)$ is P-computations from type a to type b. Assuming $\mathscr{C}$ has finite products, a *strong* endoprofunctor on $\mathscr{C}$ is additionally equipped with a family s of morphisms, called a *strength*,

$$s_{abc} : P(a, b) \to P(a \times c, b \times c),$$

natural in a, b and dinatural in c satisfying certain coherence conditions [Rivas and Jaskelioff 2017, Def. 7.1]. The strength s_{abc} informally means every computation in $P(a, b)$ can also be run alongside some unused data c. A *strong natural transformation* $\tau : \langle P, s^P \rangle \to \langle Q, s^Q \rangle$ is a natural transformation $\tau : P \to Q$ making the following diagram commute:

$$\begin{array}{ccc} P(a,b) & \xrightarrow{s^P_{abc}} & P(a \times c, b \times c) \\ {\scriptstyle \tau_{a,b}}\downarrow & & \downarrow{\scriptstyle \tau_{a\times c, b\times c}} \\ Q(a,b) & \xrightarrow[s^Q_{abc}]{} & P(a \times c, b \times c) \end{array}$$

2.1.7∗2. Strong endoprofunctors and strong natural transformations between them form a category $\textsc{EndoPro}_s(\mathscr{C})$ [Rivas and Jaskelioff 2017], which can be equipped with a monoidal structure $\langle I, \otimes \rangle$:

$$I(a, b) = \mathscr{C}(a, b) \qquad (P \otimes Q)(a, b) = \int^{x \in \mathscr{C}} P(a, x) \times Q(x, b) \tag{2.11}$$

where the associated strength of I is

$$s^I_{abc} := (- \times c) : \mathscr{C}(a, b) \to \mathscr{C}(a \times c, b \times c),$$

and the strength of $P \otimes Q$ is the composite

$$\begin{aligned} \int^x P(a, x) \times Q(x, b) &\longrightarrow \int^x P(a \times c, x \times c) \times Q(x \times c, b \times c) \\ &\longrightarrow \int^y P(a \times c, y) \times Q(y, b \times c) \end{aligned}$$

where the first arrow is $\int^x s^P_{axb} \times s^Q_{xbc}$ and the second arrow is the coprojection morphism for $x \times c$ of the coend.

Informally, the product $P \otimes Q$ are two computations P and Q with some type of data x flowing from P to Q, so it allows more dependency than applicatives, but unlike $M \circ M$, it does not allow the second computation to *dynamically* depend

on the return value of P (see Pieters et al. [2020] and Lindley et al. [2011] for more detailed comparisons).

2.1.7∗3. While the category of endoprofunctors *without* strengths has a closed monoidal structure $\langle I, \otimes \rangle$ with the same definition of I and $\otimes$ as those in (2.11), with the right adjoints $P \multimap -$ to $- \otimes P$ given by

$$(P \multimap Q)(a, b) = \int_{x \in \mathscr{C}} [P(b, x), Q(a, x)],$$

I do not know whether endofunctors *with* strengths $\textsc{EndoPro}_s(\mathscr{C})$ is also closed.

2.1.7∗4 Remark. In this section, we have paid special attention to the cocompleteness and closedness of monoidal categories. However, in the rest of this thesis, cocompleteness will play a much more predominant role than closedness: we need cocompleteness to construct free algebras of algebraic theories, while closedness will only be used for a few concrete constructions (such as the construction of the free monoid over A as the list object $\mu X.\, 1 + A \,\square\, X$).

2.2 Equational Systems and Translations

2.2∗1. We have seen monoids in various monoidal categories, but if the only thing that we know about a monoid is its unit and multiplication, then it is barely interesting. Instead, monoids in practice usually come with operations: for example, the state monad $(- \times S)^S$ comes with operations for reading and writing the mutable state; the exception monad $- + E$ has operations for throwing and catching exceptions; the ordinary monoid in $\textsc{Set}$ of lists with concatenation has the operation of appending an element to a list.

Therefore, we shall have a way to talk about algebraic theories in monoidal categories systematically. In this thesis, we will use Fiore and Hur's [2007; 2009] *equational systems*, a simple but powerful framework subsuming (enriched) algebraic theories. Importantly, the model theory of equational systems is well developed: Fiore and Hur established conditions for the existence of free algebras, cocompleteness of the category of models, and monadicity.

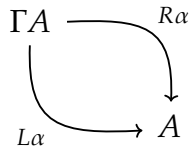
2.2∗2. In the following, we first briefly introduce equational systems and Fiore and Hur's theorem for the existence of free algebras of equational systems, and we also show a *constructively valid* proof of the theorem for small-complete small categories (Section 2.2.1). We then introduce *functorial translations* between equational systems, making them a category, and discuss some basic properties of this category (Sections 2.2.2 and 2.2.3).

2.2.1 Equational Systems

2.2.1∗1. Generally speaking, an algebraic theory consists of the *signature* and *equations* of its operations. A concise way to specify a signature over a category $\mathscr{C}$ is just a functor $\Sigma : \mathscr{C} \to \mathscr{C}$, and then a Σ-*algebra* is an object $A \in \mathscr{C}$, called the *carrier*, together with a *structure map* $\alpha : \Sigma A \to A$. For example, the signature functor of the *theory of monoids* in a monoidal category $\mathscr{E}$ with binary coproducts is $\Sigma_{\text{MON}} = (- \square -) + I$. A Σ_{MON}-algebra $\langle A, \alpha \rangle$ is an object A with a morphism $\alpha : (A \square A) + I \to A$, or equivalently two morphisms $A \square A \to A$ and $I \to A$.

2.2.1∗2. We denote the category of Σ-algebras by Σ-ALG, whose morphisms from $\langle A, \alpha \rangle$ to $\langle B, \beta \rangle$ are *algebra homomorphisms*, i.e. morphisms $h : A \to B$ in $\mathscr{C}$ such that $h \cdot \alpha = \beta \cdot \Sigma h : \Sigma A \to B$. The forgetful functor dropping the structure map is denoted by $\mathrm{U}_\Sigma : \Sigma\text{-ALG} \to \mathscr{C}$ or just U when it is not ambiguous.

2.2.1∗3. Equations on a signature $\Sigma : \mathscr{C} \to \mathscr{C}$ are usually presented as commutative diagrams, such as the diagrams in 2.1∗2 for monoids. Pictorially, a commutative diagram looks like the following:



which contains a pair of paths $L\alpha$ and $R\alpha$ from some formal object ΓA to some formal object A, parameterised by a formal morphism $\alpha : \Sigma A \to A$ of the operation. The starting node ΓA can be called the *context* of the diagram.

A categorical way to make precise such as a diagram is a functor $\Gamma : \mathscr{C} \to \mathscr{C}$ and a pair of functors $L, R : \Sigma\text{-ALG} \to \Gamma\text{-ALG}$. For example, let $\langle \mathscr{E}, \square, I \rangle$ be a monoidal category and $\Sigma = (- \square -) : \mathscr{E} \to \mathscr{E}$ be the signature of a binary operation. To encode the associativity diagram in 2.1∗2 for Σ-algebras, we let the functor $\Gamma : \mathscr{E} \to \mathscr{E}$ be $(- \square -) \square -$, and the two paths are represented by two functors $L, R : (- \square -)\text{-ALG} \to ((- \square -) \square -)\text{-ALG}$:

$$L\langle A, \mu : A \square A \to A \rangle = \langle A, \mu \cdot (\mu \square A) \rangle$$
$$R\langle A, \mu : A \square A \to A \rangle = \langle A, \mu \cdot (A \square \mu) \cdot \alpha_{A,A,A} \rangle$$

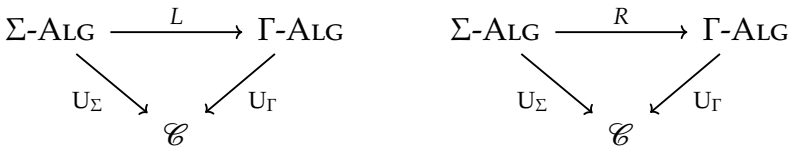
The functors $L, R : \Sigma\text{-ALG} \to \Gamma\text{-ALG}$ must satisfy $\mathrm{U}_\Gamma \circ L = \mathrm{U}_\Sigma$ and $\mathrm{U}_\Gamma \circ R = \mathrm{U}_\Sigma$.

2.2.1∗4 Definition (Fiore and Hur [2009]). An *equational system* on a category $\mathscr{C}$

$$\dot{\Sigma} := (\Sigma \triangleright \Gamma \vdash L = R)$$

consists of four functors: a *(functorial) signature* $\Sigma : \mathscr{C} \to \mathscr{C}$, a *(functorial) context* $\Gamma : \mathscr{C} \to \mathscr{C}$, and a pair of two *(functorial) terms* $L, R : \Sigma\text{-ALG} \to \Gamma\text{-ALG}$ making the

following diagrams commute:



(2.12)

An *algebra*, or a *model*, of $\dot{\Sigma}$ is a Σ-algebra $\langle A \in \mathscr{C}, \alpha : \Sigma A \to A \rangle$ such that

$$L\langle A, \alpha \rangle = R\langle A, \alpha \rangle \in \Gamma\text{-Alg}.$$

We denote by $\dot{\Sigma}$-Alg the full subcategory of Σ-Alg containing $\dot{\Sigma}$-algebras.

2.2.1∗5 Notation. In the definition above, the functors $L, R : \Sigma\text{-Alg} \to \Gamma\text{-Alg}$ always send objects $\langle A, \alpha : \Sigma A \to A \rangle$ to $\langle A, \beta : \Gamma A \to A \rangle$, keeping carriers A unchanged, so we will simply write $L\alpha : \Gamma A \to A$ to mean $\pi_2(L\langle A, \alpha \rangle)$.

2.2.1∗6 Example. Let $\mathscr{C}$ be a monoidal category with binary coproducts. The concept of monoids in $\mathscr{C}$ (2.1∗2) can be defined as an equational system on $\mathscr{C}$

$$\textsc{Mon} := (\Sigma_{\textsc{Mon}} \triangleright \Gamma_{\textsc{Mon}} \vdash L_{\textsc{Mon}} = R_{\textsc{Mon}}) \tag{2.13}$$

with functorial signature and contexts

$$\Sigma_{\textsc{Mon}}\ M = (M \square M) + I$$
$$\Gamma_{\textsc{Mon}}\ M = ((M \square M) \square M) + (I \square M) + (M \square I),$$

and $L_{\textsc{Mon}}, R_{\textsc{Mon}} : \Sigma_{\textsc{Mon}}\text{-Alg} \to \Gamma_{\textsc{Mon}}\text{-Alg}$ given by

$$L_{\textsc{Mon}}\ \beta = [L_1, L_2, L_3] \qquad R_{\textsc{Mon}}\ \beta = [R_1, R_2, R_3],$$

where we define $\mu := (\beta \cdot \iota_1) : M \square M \to M$, $\eta := (\beta \cdot \iota_2) : I \to M$ and

$$L_1 := \mu \cdot (\mu \square M) \qquad R_1 := \mu \cdot (M \square \mu) \cdot \alpha_{M,M,M}$$
$$L_2 := \mu \cdot (\eta \square M) \qquad R_2 := \lambda_M$$
$$L_3 := \mu \cdot (M \square \eta) \qquad R_3 := \rho_M$$

Each pair L_i and R_i encodes a commutative diagram for monoids in 2.1∗2. An algebra of this equational system is precisely a monoid in $\mathscr{C}$.

2.2.1∗7 Notation. From the example above we see that although the definition of equational systems allows exactly one operation and one equation, multiple operations/equations can be encoded via coproducts. Therefore we will just say *an equational system with a set of operations* $\{\Sigma_o : \mathscr{C} \to \mathscr{C}\}_{o \in O}$ and *a set of equations* $\{\Gamma_e \vdash L_e = R_e\}_{e \in E}$, where $\Gamma_e : \mathscr{C} \to \mathscr{C}$ and $L_e, R_e : (\coprod_{o \in O} \Sigma_o)\text{-Alg} \to \Gamma_e\text{-Alg}$, provided that $\mathscr{C}$ has O-indexed and E-indexed coproducts.

Moreover, given an equational system $\dot{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ on a category

$\mathscr{C}$ with binary coproducts, we denote by $\dot{\Sigma} \curlywedge \Sigma'$ the extension of $\dot{\Sigma}$ with a new operations of signature $\Sigma' : \mathscr{C} \to \mathscr{C}$:

$$\dot{\Sigma} \curlywedge \Sigma' := (\Sigma + \Sigma' \triangleright \Gamma \vdash L \circ \mathrm{U} = R \circ \mathrm{U})$$

where $\mathrm{U} : (\Sigma + \Sigma')\text{-}\textsc{Alg} \to \Sigma\text{-}\textsc{Alg}$ is the forgetful functor dropping Σ' operations. Similarly, we denote by $\dot{\Sigma} \curlywedge (\Gamma' \vdash L' = R')$ the extension of $\dot{\Sigma}$ with a new equation:

$$\dot{\Sigma} \curlywedge (\Gamma' \vdash L' = R') := (\Sigma \triangleright \Gamma + \Gamma' \vdash [L, L'] = [R, R']),$$

where $[L, L'] : \Sigma\text{-}\textsc{Alg} \to (\Gamma + \Gamma')\text{-}\textsc{Alg}$ is the functor mapping $\alpha : \Sigma X \to X$ to $[L\alpha, L'\alpha] : (\Gamma + \Gamma')X \to X$, and $[R, R']$ is similar.

2.2.1∗8 Example. The concept of Eilenberg-Moore algebras can be defined as an equational system. Let M be a monad on some $\mathscr{C}$ with coproducts. The equational system $M\text{-}\textsc{Alg}$ has as signature M itself and two equations: $\mathrm{Id}_{\mathscr{C}} \vdash L_1 = R_1$ and $M \circ M \vdash L_2 = R_2$ where for all $\langle X \in \mathscr{C}, \alpha : MX \to X \rangle$,

$$L_1\alpha = (X \xrightarrow{\eta_X} MX \xrightarrow{\alpha} X) \qquad R_1\alpha = id_X$$
$$L_2\alpha = (M(MX) \xrightarrow{\mu_X} MX \xrightarrow{\alpha} X) \qquad R_2\alpha = (M(MX) \xrightarrow{M\alpha} MX \xrightarrow{\alpha} X)$$

The category of algebras for this equational system is precisely the Eilenberg-Moore category of the monad M.

2.2.1∗9. Equational systems can also express *inequations* using a standard trick in enriched algebraic theories [Robinson 2002]. Let $\mathscr{C}$ be a category with an enrichment over $\textsc{Poset}$, which means that every hom-set $\mathscr{C}(A, B)$ is equipped with a partial order and composition of $\mathscr{C}$-morphisms $\mathscr{C}(A, B) \times \mathscr{C}(B, C) \to \mathscr{C}(A, C)$ is monotone, such that $\mathscr{C}$ is *copowered* (also called *tensored*) over $\textsc{Poset}$, which means that for every $A \in \mathscr{C}$ and $P \in \textsc{Poset}$, there is an object $P \cdot A \in \mathscr{C}$ and a natural isomorphism between posets:

$$\mathscr{C}(P \cdot A, B) \cong \textsc{Poset}(P, \mathscr{C}(A, B)).$$

Let $\mathfrak{S}$ be the two-element poset $\{\bot \sqsubseteq \top\}$. For every endofunctor $\Sigma : \mathscr{C} \to \mathscr{C}$ (not necessarily a $\textsc{Poset}$-enriched functor), an operation $\alpha : \mathfrak{S} \cdot \Sigma A \to A$ is then two operations $\alpha_\bot, \alpha_\top : \Sigma A \to A$ such that $\alpha_\bot \sqsubseteq \alpha_\top$ in the poset $\mathscr{C}(\Sigma A, A)$. Therefore in this case we can impose orders on operations of an equational system. Moreover, we can encode an inequational axiom $l(x) \sqsubseteq r(x)$ by introducing two operations $\bar{l} \sqsubseteq \bar{r}$ and equations stating that $l(x) = \bar{l}(x)$ and $r(x) = \bar{r}(x)$. This trick can also be generalised to $\textsc{Cat}$-enriched categories, i.e. 2-categories, to express lax-equations or pseudo-equations.

2.2.1∗10 Example. The category $\textsc{Poset}$ is enriched over itself with the pointwise order on $\textsc{Poset}(A, B)$, and it is tensored with $P \cdot A$ given by cartesian product

$P \times A$. The equational system Bot over the category Poset expresses posets with a bottom element: it has the signature functor $\Sigma_{\text{Bot}} := 1 + (\mathfrak{S} \cdot -)$ and two equations $\{\text{Id} \vdash L_i = R_i\}_{i=1,2}$ where for every $\alpha : 1 + \mathfrak{S} \cdot A \to A$

$$L_1\alpha = (A \xrightarrow{\iota_\perp} \mathfrak{S} \cdot A \xrightarrow{\alpha \cdot \iota_2} A) \qquad R_1\alpha = (A \to 1 \xrightarrow{\alpha \cdot \iota_1} A)$$
$$L_2\alpha = (A \xrightarrow{\iota_\top} \mathfrak{S} \cdot A \xrightarrow{\alpha \cdot \iota_2} A) \qquad R_2\alpha = (A \xrightarrow{id_A} A)$$

An algebra of Bot is therefore a preorder A with an element $b \in A$ and two monotone functions $\bar{l} : A \to A$, and $\bar{r} : A \to A$ such that $\bar{l} \sqsubseteq \bar{r}$. The two equations of Bot state that $\bar{l}(x) = b$ and $\bar{r}(x) = x$ for all $x \in A$, so an algebra of Bot is equivalently a preorder with a bottom element b.

2.2.1∗11. Among the algebras of an equational system $\dot{\Sigma}$ over $\mathscr{C}$, the *free algebras* are particularly useful since they represent *abstract syntax* of terms built from variables and operations of the theory. The abstract syntax can be *interpreted* with another model using the free-forgetful adjunction:

$$\phi : \mathscr{C}(X, A) \;\cong\; \dot{\Sigma}\text{-Alg}(\mathrm{F}\, X, \langle A, \alpha \rangle)$$

Given any model $\langle A, \alpha \rangle$ of $\dot{\Sigma}$ and $g : X \to A$, the morphism $\phi(g) : \mathrm{F}\, X \to \langle A, \alpha \rangle$ interprets the free algebra with the semantic model $\langle A, \alpha \rangle$, with generators X interpreted by $g : X \to A$. Fiore and Hur [2009] showed several sufficient conditions for the existence of free algebras, which we record below.

2.2.1∗12 Theorem (Fiore and Hur [2009]). *Let $\dot{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ be an equational system over $\mathscr{C}$. If $\mathscr{C}$ is (small-) cocomplete and one of the following holds*:

- Σ *and* Γ *preserve colimits of α-chains for a limit ordinal α;*
- Σ *preserves colimits of α-chains for a limit ordinal α, and both Σ and Γ preserve epimorphisms in $\mathscr{C}$;*
- Σ *preserves colimits of α-chains for a limit ordinal α, and Σ preserves epimorphisms, and $\mathscr{C}$ has no transfinite chains of proper epimorphisms,*

then there are left adjoints to the inclusion functor $\dot{\Sigma}$-Alg $\hookrightarrow$ Σ-Alg and the forgetful functor Σ-Alg $\to \mathscr{C}$:

$$\dot{\Sigma}\text{-Alg} \underset{\hookrightarrow}{\overset{\longleftarrow}{\perp}} \Sigma\text{-Alg} \underset{\longrightarrow}{\overset{\longleftarrow}{\perp}} \mathscr{C} \tag{2.14}$$

Moreover, $\dot{\Sigma}$-Alg is cocomplete and the composite of the adjunction is monadic.

2.2.1∗13 Notation. We denote the composite adjunction of (2.14) by $\mathrm{F}_{\dot{\Sigma}} \dashv \mathrm{U}_{\dot{\Sigma}}$, or simply $\mathrm{F} \dashv \mathrm{U}$ when $\dot{\Sigma}$ is clear from context. Moreover, the initial $\dot{\Sigma}$-algebra is denoted by $\langle \mu\dot{\Sigma}, \alpha^{\dot{\Sigma}} : \Sigma\mu\dot{\Sigma} \to \mu\dot{\Sigma} \rangle$.

2.2.1∗14. Fiore and Hur's proof of this result is quite technical, but we will not rely on the specifics of their construction. For concreteness, we provide some basic intuition here: the free Σ-algebra on some $A \in \mathscr{C}$ is first constructed by a transfinite iteration of $A + \Sigma -$ on 0 [Adámek 1974]

$$0 \xrightarrow{!} A + \Sigma 0 \xrightarrow{A+\Sigma!} A + \Sigma(A + \Sigma 0) \longrightarrow \cdots$$

and taking colimits for limit ordinals. The iteration will stop at some $X \cong A + \Sigma X$ in α steps, giving the carrier of the free Σ-algebra. Then it is quotiented by the equation $L = R$ and the congruence rule, using Fiore and Hur's *algebraic coequalisers*. The quotienting may also need to be repeated α times when Σ or Γ does not preserve epimorphisms. The result of quotienting is the free $\dot{\Sigma}$-algebra.

2.2.1∗15 Example. Let $\mathscr{C}$ be a small-cocomplete monoidal category such that $\square : \mathscr{C} \times \mathscr{C} \to \mathscr{C}$ preserves α-chains for some limit ordinal α. For example, $\mathscr{C}$ can be $\langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ for an $\mathrm{l}\kappa\mathrm{p}$ $\mathscr{C}$ from Section 2.1.2 or $\langle \textsc{Endo}_\kappa(\textsc{Set}), *, \mathrm{Id} \rangle$ from Section 2.1.6. Then the first condition of Theorem 2.2.1∗12 is applicable to the equational system Mon from Example 2.2.1∗6, so every $A \in \mathscr{C}$ has a free monoid.

Moreover, Fiore and Hur [2009] showed that when $\mathscr{C}$ is left closed, there is a simple formula for free monoids: for every $A \in \mathscr{C}$, the free monoid over A is the initial algebra $\mu X.\ I + A \square X$ (sometimes called the *list object* for A) equipped with appropriate monoid operations. This formula is useful in practice: when $\mathscr{C}$ is $\langle \textsc{Set}, \times, 1 \rangle$, it is exactly the usual definition $\mu X.\ 1 + A \times X$ of lists of A-elements; and when $\mathscr{C}$ is $\langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ or $\langle \textsc{Endo}_\kappa(\textsc{Set}), *, \mathrm{Id} \rangle$, this gives formulas for free monads and free applicatives that are suitable for implementation [Rivas and Jaskelioff 2017]. Note that this formula needs closedness to work: to define the monoid multiplication

$$(\mu X.\ I + A \square X) \square (\mu X.\ I + A \square X) \to (\mu X.\ I + A \square X),$$

we need closedness to shift one $(\mu X.\ I + A \square X)$ to the right-hand side

$$(\mu X.\ I + A \square X) \to (\mu X.\ I + A \square X)/(\mu X.\ I + A \square X)$$

and then we can use the universal property of the initial algebra.

2.2.1∗16 Remark. When $\mathscr{C}$ is not closed, the initial algebra $\mu X.\ 1 + A \square X$ may not be the free monoid over A. For a counterexample, let $\mathscr{C}$ be the cartesian monoidal category $\langle \textsc{Mon}, \times, 1 \rangle$ of monoids in $\langle \textsc{Set}, \times, 1 \rangle$. By the Eckmann-Hilton argument, a monoid object in the monoidal category $\langle \textsc{Mon}, \times, 1 \rangle$ is precisely an ordinary commutative monoid in sets (so the *structure* of a monoid object degenerates into a *property* in this case). The free monoid object over some $M \in \textsc{Mon}$ is then obtained by quotienting M with the commutativity equations.

On the other hand, the initial algebra $\mu X.\ 1 + M \times X$ in Mon is the monoid

whose elements are finite lists $\langle m_1, m_2, \ldots, m_n\rangle$ of M-elements, considered up to trailing zeros, by which we mean $\langle m_1, m_2, \ldots, m_n\rangle$ and $\langle m_1, m_2, \ldots, m_n, e, \ldots, e\rangle$ are considered the same, where e is the unit element of M. Its unit element is the empty sequence $\langle\rangle$, and its multiplication is the *pairwise* multiplication using the multiplication of M (after padding with enough trailing zeros):

$$\langle m_1, m_2, \ldots, m_n\rangle \cdot \langle m'_1, m'_2, \ldots, m'_n\rangle = \langle m_1 \cdot_M m'_1, \ldots, m_n \cdot_M m'_n\rangle. \tag{2.15}$$

The associated $(1 + M \times -)$-algebra is still given by the empty list and *cons* as usual. Now we observe that the multiplication (2.15) is not commutative when M is not commutative, so the initial algebra $\mu X.\, 1 + M \times X$ may not be a monoid object in $\langle \textsc{Mon}, \times, 1\rangle$, let alone the free monoid object over M.

2.2.1∗17. An appealing aspect of Theorem 2.2.1∗12 is that the base category $\mathscr{C}$ is only required to be cocomplete rather than locally κ-presentable. Therefore the theorem applies to the category $\textsc{dCpo}$ of *directed complete partial orders*, generalising the construction of free dcpo-algebras for operations of finite arities and (in)equations [Abramsky and Jung 1995, Theorem 6.1.2]. Allowing an endofunctor $\Sigma : \textsc{dCpo} \to \textsc{dCpo}$ as the signature comes in handy: in particular, it allows us to express operations that takes an ascending chain $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \cdots$ as arguments by choosing $\Sigma := (-)^\omega$ where $\omega := \{0 \sqsubseteq 1 \sqsubseteq 2 \sqsubseteq \cdots\}$. This functor preserves colimits of all Ω-chains, where Ω is the first uncountable ordinal.

2.2.1∗18. I do not know if any of the sufficient conditions for the existence of free algebras given by Theorem 2.2.1∗12 (or possibly some of their classically equivalent conditions) is constructively true, and in particular, whether they are applicable to small-complete small categories in realizability toposes (2.1.1∗4). Anyway, we have an alternative constructive proof for small-complete small categories using *impredicative encodings* [Awodey et al. 2018], a refinement of the well known *Church encoding* of inductive datatypes.

Remarkably, the theorem below imposes no constraints on the functorial signature Σ and context Γ, generalising the result that every endofunctor on a small-complete small category has an initial algebra [Hyland 1988, §3.1].

2.2.1∗19 Theorem. *Let $\mathscr{C}$ be a small-complete small category. For every equational system $\dot\Sigma = (\Sigma \triangleright \Gamma \vdash L = R)$ over $\mathscr{C}$, there is a monadic adjunction $\mathrm{F}_{\dot\Sigma} \dashv \mathrm{U}_{\dot\Sigma}$. Moreover, the category $\dot\Sigma$-$\textsc{Alg}$ is a small-complete-and-cocomplete small category.*

2.2.1∗20. An easy proof of the theorem goes by first showing that $\dot\Sigma$-$\textsc{Alg}$ is also a small-complete small category for all equational systems $\dot\Sigma$-$\textsc{Alg}$ over $\mathscr{C}$, and then the free algebra of $\dot\Sigma$ over an object $X \in \mathscr{C}$ can be constructed from the initial algebra of $\dot\Sigma \triangleleft \mathrm{K}_X$, where $\mathrm{K}_X : \mathscr{C} \to \mathscr{C}$ is the constant functor mapping to X. The initial algebra of $\dot\Sigma \triangleleft \mathrm{K}_X$ can be then constructed as the limit of the

identity functor $\mathrm{Id} : (\dot{\Sigma} \upharpoonleft \mathrm{K}_X)\text{-}\textsc{Alg} \to (\dot{\Sigma} \upharpoonleft \mathrm{K}_X)\text{-}\textsc{Alg}$ [Mac Lane 1998, §X.1 Lemma 1], which exists since $(\dot{\Sigma} \upharpoonleft \mathrm{K}_X)\text{-}\textsc{Alg}$ is a small-complete small category.

The proof below is essentially the same idea but with some levels of abstraction expanded out. It was written before I noticed the simple argument above, but it is kept here since its explicit character may be useful to readers interested in formalising the construction in type theories with impredicative universes.

Proof. First we notice that $\dot{\Sigma}\text{-}\textsc{Alg}$ is a small category: if $\textsc{Obj}\,\mathscr{C}$ and $\textsc{Mor}\,\mathscr{C}$ are small sets, then $\textsc{Obj}\,\dot{\Sigma}\text{-}\textsc{Alg} = \{\langle A, f\rangle \mid A \in \textsc{Obj}\,\mathscr{C}, f \in \mathscr{C}(\Sigma A, A)\}$ is also a small set, and $\dot{\Sigma}\text{-}\textsc{Alg}(\langle A, f\rangle, \langle B, g\rangle) \subseteq \mathscr{C}(A, B)$ is clearly a small set as well.

For every $X \in \mathscr{C}$, we define $Y \in \mathscr{C}$ to be the following end in $\mathscr{C}$:

$$Y := \int_{\langle A,f\rangle \in \textsc{Obj}\,\dot{\Sigma}\text{-}\textsc{Alg}} \prod_{g \in \mathscr{C}(X,A)} A$$

which exists since $\mathscr{C}$ is small-complete. We define a Σ-algebra $\alpha : \Sigma Y \to Y$ as follows: for every $\langle A, f\rangle \in \textsc{Obj}\,\dot{\Sigma}\text{-}\textsc{Alg}$ and $k : X \to A$, there is a $\mathscr{C}$-morphism:

$$\Sigma Y \xrightarrow{\Sigma\pi_{\langle A,f\rangle}} \Sigma(\textstyle\prod_{g \in \mathscr{C}(X,A)} A) \xrightarrow{\Sigma\pi_g} \Sigma A \xrightarrow{f} A$$

and it can be checked that this defines a wedge $w_{\Sigma,f} : \Sigma Y \to \Pi_{g \in \mathscr{C}(X,A)} A$, so by the universal property of the end Y, we have a morphism $\alpha : \Sigma Y \to Y$ such that

$$\begin{array}{ccccc} \Sigma Y & \xrightarrow{\Sigma\pi_{\langle A,f\rangle}} & \Sigma \prod_g A & \xrightarrow{\Sigma\pi_k} & \Sigma A \\ {\scriptstyle\alpha}\downarrow & & & & \downarrow{\scriptstyle f} \\ Y & \xrightarrow[\pi_{\langle A,f\rangle}]{} & \prod_g A & \xrightarrow[\pi_k]{} & A \end{array} \tag{2.16}$$

commutes. Notice that this square means $\pi_k \cdot \pi_{\langle A,f\rangle}$ is a Σ-algebra homomorphism.

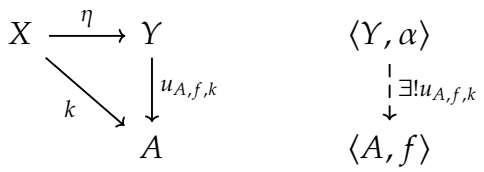
Next we show that $\langle Y, \alpha\rangle$ is in the subcategory $\dot{\Sigma}\text{-}\textsc{Alg} \subseteq \Sigma\text{-}\textsc{Alg}$, i.e. $L\alpha = R\alpha : \Gamma Y \to Y$. Again, we consider every $\langle A, f\rangle \in \textsc{Obj}\,\dot{\Sigma}\text{-}\textsc{Alg}$ and $k : X \to A$. The functors $L, R : \Sigma\text{-}\textsc{Alg} \to \Gamma\text{-}\textsc{Alg}$ maps the homomorphism square (2.16) to

$$\begin{array}{ccc} \Gamma Y & \xrightarrow{\Gamma(\pi_k \cdot \pi_{\langle A,f\rangle})} & \Gamma A \\ {\scriptstyle L\alpha}\downarrow\downarrow{\scriptstyle R\alpha} & & \downarrow{\scriptstyle Lf=Rf} \\ Y & \xrightarrow[\pi_k \cdot \pi_{\langle A,f\rangle}]{} & A \end{array}$$

Hence $\pi_k \cdot \pi_{\langle A,f\rangle} \cdot L\alpha = \pi_k \cdot \pi_{\langle A,f\rangle} \cdot R\alpha$. By the uniqueness part of the universal property of the end Y, we have $L\alpha = R\alpha$.

We have a $\dot{\Sigma}$-algebra $\langle Y, \alpha\rangle$ now, and we have a morphism $\eta : X \to Y$ defined by the wedge $\langle\langle g\rangle_{g \in \mathscr{C}(X,A)} : X \to \prod_{g \in \mathscr{C}(X,A)} A\rangle_{\langle A,f\rangle}$. We need to show that $\eta : X \to \mathrm{U}_{\dot{\Sigma}}\langle Y, \alpha\rangle$ is a universal morphism. That is to say, for every $\langle A, f\rangle \in \textsc{Obj}\,\dot{\Sigma}\text{-}\textsc{Alg}$ and $k : X \to A$, there is a unique $\dot{\Sigma}$-algebra homomorphism

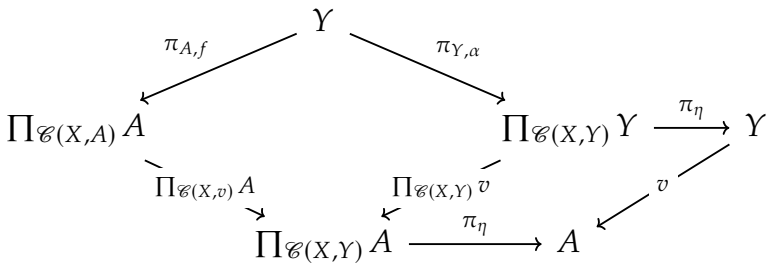
$u_{A,f,k} : \langle Y, \alpha \rangle \to \langle A, f \rangle$ such that the triangle on the left below commutes:



We let the required homomorphism $u_{A,f,k} := \pi_k \cdot \pi_{A,f} : Y \to A$ be the one in (2.16). We have $u_{A,f,k} \cdot \eta = k$ essentially by construction:

$$u_{A,f,k} \cdot \eta = (\pi_k \cdot \pi_{\langle A,f \rangle}) \cdot \langle \langle g \rangle_{g \in \mathscr{C}(X,A)} \rangle_{\langle A,f \rangle} = k.$$

The uniqueness of $u_{A,f,k}$ is slightly more involved. Given any $v : \langle Y, \alpha \rangle \to \langle A, f \rangle$ such that $v \cdot \eta = k$, by the property of Y as an end, the following commutes:



So $\pi_\eta \cdot (\prod_{\mathscr{C}(X,v)} A) \cdot \pi_{A,f} = \pi_\eta \cdot (\prod_{\mathscr{C}(X,Y)} v) \cdot \pi_{Y,\alpha}$, or equivalently

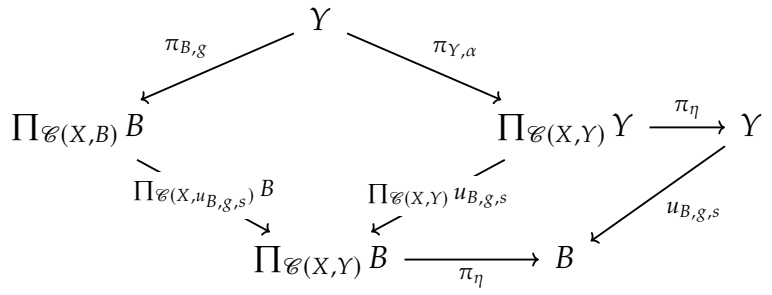
$$\pi_{v \cdot \eta} \cdot \pi_{A,f} = v \cdot \pi_\eta \cdot \pi_{Y,\alpha}$$

and therefore $u_{A,f,k} = \pi_k \cdot \pi_{A,f} = \pi_{v \cdot \eta} \cdot \pi_{A,f} = v \cdot \pi_\eta \cdot \pi_{Y,\alpha} = v \cdot u_{Y,\alpha,\eta}$. Now that $u_{A,f,k} = v \cdot u_{Y,\alpha,\eta}$, we have $v = u_{A,f,k}$ as long as $u_{Y,\alpha,\eta} = id_Y$.

It remains to show that $u_{Y,\alpha,\eta} : Y \to Y$ is the identity morphism: by the universal property of Y, it is sufficient to show that for all $\langle B, g \rangle \in \textsc{Obj}\, \dot{\Sigma}\text{-}\textsc{Alg}$ and $s : X \to B$, $(\pi_s \cdot \pi_{B,g}) \cdot u_{Y,\alpha,\eta} = (\pi_s \cdot \pi_{B,g}) \cdot id_Y$, or equivalently

$$u_{B,g,s} \cdot (\pi_\eta \cdot \pi_{Y,\alpha}) = \pi_s \cdot \pi_{B,g}. \tag{2.17}$$

Again, by the property of Y as an end and the property of products $\prod_{\mathscr{C}(X,Y)}$, the following diagram commutes:



Therefore we have $u_{B,g,s} \cdot \pi_\eta \cdot \pi_{Y,\alpha} = \pi_\eta \cdot (\prod_{\mathscr{C}(X,u_{B,g,s})} B) \cdot \pi_{B,g} = \pi_s \cdot \pi_{B,g}$. Hence

we have shown (2.17) and thus $u_{Y,\alpha,\eta} = id_Y : Y \to Y$.

We have shown that there is a free algebra over every $X \in \mathscr{C}$, and this uniquely determines an adjunction $F_{\dot\Sigma} \dashv U_{\dot\Sigma}$ as usual [Mac Lane 1998, p. 83, Theorem 2]. Using Beck's monadicity theorem, it is not difficult to show that this adjunction is monadic; see Fiore and Hur [2009, Proposition 6.4]. Similarly to how the forgetful functor of a finitary algebraic theory creates limits [Mac Lane 1998, Theorem V.1.2], it is not difficult to show that $U_{\dot\Sigma}$ creates limits too. Since $\mathscr{C}$ is small-complete, $\dot\Sigma$-Alg is also complete. Finally, a known result is that every complete small-complete is small-cocomplete [Hyland 1988, §3.1], which can be proven using an impredicative encoding similar to what we have done above. □

2.2.1∗21. Theorem 2.2.1∗12 and Theorem 2.2.1∗19 give different sufficient conditions for the existences of free algebras of equational systems, and there are some other constructive techniques such as the one by Fiore et al. [2022]. In the future, we typically only rely on the existence of free algebras but not those specific conditions guaranteeing the existence of free algebras. The following definition will be used for abstracting over those different conditions.

2.2.1∗22 Definition. Let $\mathscr{C}$ be a category. A relation $\mathscr{A} \subseteq \textsc{Endo}(\mathscr{C}) \times \textsc{Endo}(\mathscr{C})$ of endofunctors is called a *freeness condition* if every equational system $\Sigma \triangleright \Gamma \vdash L = R$ with $\langle \Sigma, \Gamma \rangle \in \mathscr{A}$ has the free-forgetful adjunction.

We denote by $\textsc{Eqs}_{\mathscr{A}}(\mathscr{C})$ the full subcategory of $\textsc{Eqs}(\mathscr{C})$ containing equational systems whose functorial signature and context are in $\mathscr{A}$.

2.2.1∗23 Example. If $\mathscr{C}$ is lκp then the relation $\textsc{Endo}_\kappa(\mathscr{C}) \times \textsc{Endo}_\kappa(\mathscr{C})$ containing all pairs of κ-accessible endofunctors (Section 2.1.2) is a freeness condition by the first item of Theorem 2.2.1∗12.

If $\mathscr{C}$ is small-complete and small, the entire relation $\textsc{Endo}(\mathscr{C}) \times \textsc{Endo}(\mathscr{C})$ is a freeness condition (2.1.1∗3) by Theorem 2.2.1∗19.

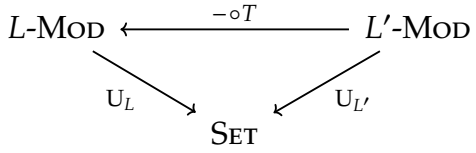
For every $\mathscr{C}$, there is always the largest freeness condition $\mathscr{F}$ containing all pairs $\langle \Sigma, \Gamma \rangle$ such that all equational systems with signature Σ and context Γ have the free-forgetful adjunction. However, $\mathscr{F}$ is less useful than it may appear: in general we do not know whether $\mathscr{F}$ has good closure properties, for example, whether $\langle \Sigma + \Sigma', \Gamma + \Gamma' \rangle$ is in $\mathscr{F}$ when $\langle \Sigma, \Gamma \rangle$ and $\langle \Sigma', \Gamma' \rangle$ are in $\mathscr{F}$.

2.2.2 Functorial Translations

2.2.2∗1. Morphisms between equational systems are not studied by Fiore and Hur [2007, 2009], but we need them later for talking about combinations of equational systems. A natural idea for morphisms from an equational system $\dot\Sigma$ to another $\dot\Psi$ is *translations*, which map operations in $\dot\Sigma$ to *terms* of $\dot\Psi$, preserving

equations in a suitable sense. However, a technical difficulty is that equational systems $\dot{\Psi}$ may not have terms, i.e. free algebras.

In the following, we avoid this by introducing a more indirect definition which we call *functorial translations* between equational systems. For some motivation, consider Lawvere theories: every morphism $T : L \to L'$ between Lawvere theories induces a functor $- \circ T : L'\text{-Mod} \to L\text{-Mod}$ between their category of models from the opposite direction. Moreover this functor commutes with the forgetful functors from models of L and L' to sets:



This mapping from morphisms T between Lawvere theories to functors $- \circ T$ between the categories of models (from the opposite direction) that commute with U_L and $\mathrm{U}_{L'}$ is in fact an equivalence of categories [Adamek et al. 2010, 11.38]. Mimicking this situation, we will define morphisms between equational systems also as functors between their categories of models from the opposite direction that commute with the forgetful functors.

From a more informal viewpoint, this is justified by that if we can 'translate' operations of an equational system Σ to terms of operations of another equational system Σ', then given a model of Σ', we can construct a model of Σ.

2.2.2∗2. In the rest of this section, we fix a category $\mathscr{C}$ and a freeness condition $\mathscr{A} \subseteq \textsc{Endo}(\mathscr{C}) \times \textsc{Endo}(\mathscr{C})$ of endofunctors (Definition 2.2.1∗22).

2.2.2∗3 Definition. A *functorial translation* of equational systems on $\mathscr{C}$ from $\dot{\Sigma} = (\Sigma \triangleright \Gamma \vdash L = R)$ to $\dot{\Sigma}' = (\Sigma' \triangleright \Gamma' \vdash L' = R')$ is a functor $T : \dot{\Sigma}'\text{-Alg} \to \dot{\Sigma}\text{-Alg}$ such that $\mathrm{U}_{\dot{\Sigma}} \circ T = \mathrm{U}_{\dot{\Sigma}'}$, where $\mathrm{U}_{\dot{\Sigma}} : \dot{\Sigma}\text{-Alg} \to \mathscr{C}$ and $\mathrm{U}_{\dot{\Sigma}'} : \dot{\Sigma}'\text{-Alg} \to \mathscr{C}$ are the forgetful functors. Equational systems on $\mathscr{C}$ and translations form a category $\textsc{Eqs}(\mathscr{C})$ whose identity morphisms are the identity functors $\dot{\Sigma}\text{-Alg} \to \dot{\Sigma}\text{-Alg}$ and composition of translations $T \circ T'$ is functor composition.

2.2.2∗4 Example. Let $\mathscr{C}$ be a category with finite coproducts and products. The theory Grp of groups over $\mathscr{C}$ is the theory Mon of monoids in $\langle \mathscr{C}, \times, 1 \rangle$ from Example 2.2.1∗6 extended with a new operation i with functorial signature $\mathrm{Id} : \mathscr{C} \to \mathscr{C}$ and a new functorial equation $\mathrm{Id} \vdash L = R$ where

$$L\langle M, \eta, \mu, i \rangle = \langle M, M \xrightarrow{\langle id_X, i \rangle} M \times M \xrightarrow{\mu} M \rangle$$
$$R\langle M, \eta, \mu, i \rangle = \langle M, M \xrightarrow{!} 1 \xrightarrow{\eta} M \rangle$$

Then there is a translation $T : \textsc{Mon} \to \textsc{Grp}$ that maps every $\langle M, \eta, \mu, i \rangle$ in

Grp-Alg to an object $\langle M, \eta, \mu \rangle$ in Mon-Alg by forgetting the newly added operation. We call translations like $T : \text{Mon} \to \text{Grp}$ that simply forget some operations and equations *inclusion translations*.

2.2.2∗5. For equational systems with free-algebras, the following lemma shows that functorial translations between them coincide with the expected notion of translations: maps from operations to terms.

2.2.2∗6 Lemma. Let $\dot{\Sigma}, \dot{\Psi} \in \text{Eqs}_{\mathscr{A}}(\mathscr{C})$. There is a bijection between functorial translations $T : \dot{\Sigma} \to \dot{\Psi}$ and monad morphisms $m : \text{U}_{\dot{\Sigma}}\text{F}_{\dot{\Sigma}} \to \text{U}_{\dot{\Psi}}\text{F}_{\dot{\Psi}}$.

Proof. By Fiore and Hur [2009, Proposition 6.4], the free-forgetful adjunction for an equational system is always monadic, so functorial translations, i.e. functors $T : \dot{\Psi}\text{-Alg} \to \dot{\Sigma}\text{-Alg}$ such that $\text{U}_{\dot{\Sigma}} \circ T = \text{U}_{\dot{\Psi}}$, are in bijection with functors $S : \mathscr{C}^{\text{U}_{\dot{\Psi}}\text{F}_{\dot{\Psi}}} \to \mathscr{C}^{\text{U}_{\dot{\Sigma}}\text{F}_{\dot{\Sigma}}}$ between the corresponding Eilenberg-Moore categories that commute with the forgetful functors. By Borceux [1994b, Proposition 4.5.9], those functors S are in bijection with monad morphisms $m : \text{U}_{\dot{\Sigma}}\text{F}_{\dot{\Sigma}} \to \text{U}_{\dot{\Psi}}\text{F}_{\dot{\Psi}}$. □

2.2.2∗7 Corollary. The two functorial terms $L, R : \Sigma\text{-Alg} \to \Gamma\text{-Alg}$ of an equational system can also be viewed as translations between the equational systems of signatures Γ and Σ without equations. Therefore when Σ and Γ have free algebras, L and R are in bijection with two monad morphisms $\text{U}_\Gamma\text{F}_\Gamma \to \text{U}_\Sigma\text{F}_\Sigma$. When $\mathscr{C}$ is locally small and small-complete, the monad $\text{U}_\Gamma\text{F}_\Gamma$ for free Γ-algebras is also the free monad over the endofunctor Γ [nLab 2024a, Theorem 3.2]. In this case, monad morphisms $\text{U}_\Gamma\text{F}_\Gamma \to \text{U}_\Sigma\text{F}_\Sigma$ are in bijection with natural transformations $\Gamma \to \text{U}_\Sigma\text{F}_\Sigma$. Moreover, it can be checked that a Σ-algebra $\alpha : \Sigma A \to A$ satisfies a functorial equation $\Gamma \vdash L = R$ if and only if the following commutes:

$$\Gamma A \underset{\tilde{R}_A}{\overset{\tilde{L}_A}{\rightrightarrows}} \text{U}_\Sigma\text{F}_\Sigma A \xrightarrow{\text{U}_\Sigma \epsilon_{\langle A,\alpha\rangle}} A$$

where $\tilde{L}$ and $\tilde{R}$ are the natural transformations corresponding to L and R.

2.2.2∗8 Theorem. *Let $\mathscr{C}$ be a category with binary coproducts and $\text{Eqs}_f(\mathscr{C}) \subseteq \text{Eqs}(\mathscr{C})$ be the full subcategory containing all equational systems admitting the free-forgetful adjunction. Then we have an equivalence of categories*

$$\text{Eqs}_f(\mathscr{C}) \quad \cong \quad \text{Mon}(\mathscr{C}),$$

where $\text{Mon}(\mathscr{C})$ is the category of monads over $\mathscr{C}$ and monad morphisms.

Proof. Example 2.2.1∗8 shows that every monad M induces an equational system M-Alg whose category of algebras is precisely the Eilenberg-Moore category

of M. Thus the equational system M-Alg is in $\mathrm{Eqs}_f(\mathscr{C})$ since free Eilenberg-Moore algebras always exist (which are simply $\langle X, \mu_X : M(MX) \to MX\rangle$ for all $X \in \mathscr{C}$). By Lemma 2.2.2∗6 above, this construction extends to a fully faithful functor $\mathrm{Mon}(\mathscr{C}) \to \mathrm{Eqs}_f(\mathscr{C})$. Moreover, the forgetful functor for every equational system $\dot{\Sigma} \in \mathrm{Eqs}_f(\mathscr{C})$ is monadic [Fiore and Hur 2009, Proposition 6.4], so $\dot{\Sigma}$ is isomorphic to the equational system $(\mathrm{F}_{\dot{\Sigma}}\mathrm{U}_{\dot{\Sigma}})$-Alg. Therefore we have an essentially surjective fully faithful functor $\mathrm{Mon}(\mathscr{C}) \to \mathrm{Eqs}_f(\mathscr{C})$, and thus an equivalence $\mathrm{Mon}(\mathscr{C}) \cong \mathrm{Eqs}_f(\mathscr{C})$. □

2.2.2∗9. The theorem above shows that equational systems subsume not just *monads with ranks* but *all monads*. Two natural questions are then

- Is the category $\mathrm{Eqs}(\mathscr{C})$ some kind of completion of $\mathrm{Mon}(\mathscr{C})$?
- Given a functor $\mathrm{U} : \mathscr{D} \to \mathscr{C}$, under what conditions U is the forgetful functor for an equational system on $\mathscr{C}$?

We leave answering these questions as future work.

2.2.2∗10. Below, we turn our attention to the property of translations $T : \dot{\Sigma} \to \dot{\Psi}$ as functors $T : \dot{\Psi}$-Alg $\to \dot{\Sigma}$-Alg, and show a condition for T to have left adjoints. We start with an interesting lemma saying that (when $\dot{\Psi}$ has free algebras) translations $T : \dot{\Sigma} \to \dot{\Psi}$ can be extended in a canonical way to translations $\tilde{T} : \Sigma \to \Psi$ between the respective equational systems *without equations*. Note, however, such an extension may not be unique: when the equation of $\dot{\Psi}$ is inconsistent in the sense that $\dot{\Psi}$ only has the trivial model 1, there is a unique trivial T, but there can still be different translations $\Sigma \to \Psi$.

2.2.2∗11 Lemma. Let $\mathscr{C}$ be a category and $\dot{\Sigma}, \dot{\Psi} \in \mathrm{Eqs}(\mathscr{C})$ such that $\dot{\Psi}$ has the free-forgetful adjunction. Every functorial translation $\dot{\Sigma} \to \dot{\Psi}$, i.e. a functor $T : \dot{\Psi}$-Alg $\to \dot{\Sigma}$-Alg such that $\mathrm{U}_{\dot{\Sigma}} \circ T = \mathrm{U}_{\dot{\Psi}}$ can be extended to a functor $\tilde{T} : \Psi$-Alg $\to \Sigma$-Alg such that $\mathrm{U}_{\Sigma} \circ T = \mathrm{U}_{\Psi}$, and $\tilde{T}$ coincides with T on $\dot{\Psi}$-Alg.

Proof. Given any Ψ-algebra $\langle A, \alpha : \Psi A \to A\rangle$, there is a free $\dot{\Psi}$-algebra over A with structure map $o : \Psi(\mathrm{F}_{\dot{\Psi}}A) \to \mathrm{F}_{\dot{\Psi}}A$. It is mapped by T to a $\dot{\Sigma}$-algebra $To : \Sigma(\mathrm{F}_{\dot{\Psi}}A) \to \mathrm{F}_{\dot{\Psi}}A$. We now define the functor $\tilde{T} : \Psi$-Alg $\to \Sigma$-Alg by

$$\tilde{T}\langle A, \alpha\rangle = \langle A,\ \Sigma A \xrightarrow{\Sigma\eta_A} \Sigma(\mathrm{F}_{\dot{\Psi}}A) \xrightarrow{To} \mathrm{F}_{\dot{\Psi}}A \xrightarrow{\epsilon_{A,\alpha}} A\rangle.$$

To see that $\tilde{T}$ restricts to T on $\dot{\Psi}$-Alg, supposing $\langle A, \alpha\rangle \in \dot{\Psi}$, T maps the counit

$\epsilon_{A,\alpha} : \langle \mathrm{F}_{\dot\Psi} A, o\rangle \to \langle A, \alpha\rangle$ to the following commutative diagram:

$$\begin{array}{ccc} \Sigma(\mathrm{F}_{\dot\Psi} A) & \xrightarrow{To} & \mathrm{F}_{\dot\Psi} A \\ {\scriptstyle \Sigma\epsilon_{A,\alpha}}\downarrow & & \downarrow{\scriptstyle \epsilon_{A,\alpha}} \\ \Sigma A & \xrightarrow[T\alpha]{} & A \end{array}$$

Therefore $\tilde{T}\alpha = \epsilon_{A,\alpha} \cdot To \cdot \Sigma\eta_A = T\alpha \cdot \Sigma\epsilon \cdot \Sigma\eta_A = T\alpha$ since $\epsilon_{A,\alpha} \cdot \eta_A = id$. □

2.2.2∗12. In the adjunction $\mathrm{F}_{\dot\Sigma} \dashv \mathrm{U}_{\dot\Sigma} : \dot\Sigma\text{-Alg} \to \mathscr{C}$ for free algebras, the category $\mathscr{C}$ can be viewed as the category $\emptyset$-Alg for the empty theory $\emptyset$ with no operations or equations, and $\mathrm{U}_{\dot\Sigma} : \dot\Sigma\text{-Alg} \to \emptyset\text{-Alg}$ is the unique translation from $\emptyset$ to $\dot\Sigma$. The fact that $\mathrm{U}_{\dot\Sigma}$ always has a left adjoint can be generalised to any functorial translations, giving us *relative free algebras*.

2.2.2∗13 Definition. Letting J be a set, we say that the freeness condition $\mathscr{A}$ is *closed under J-indexed coproducts* if whenever $\langle \Sigma_j, \Gamma_j\rangle \in \mathscr{A}$ for all $j \in J$ then $\coprod_j \Sigma_j$ and $\coprod_j \Gamma_j$ exist and are related by $\mathscr{A}$. Similarly, $\mathscr{A}$ is said to be *closed under constant functors* if $\langle \mathrm{K}_A, \mathrm{K}_B\rangle \in \mathscr{A}$ for all $A, B \in \mathscr{C}$.

2.2.2∗14 Theorem. *Assuming that $\mathscr{A}$ is closed under binary coproducts and constant functors, every functorial translation $T : \dot\Sigma \to \dot\Psi$ in $\mathrm{Eqs}_{\mathscr{A}}(\mathscr{C})$ as a functor $T : \dot\Psi\text{-Alg} \to \dot\Sigma\text{-Alg}$ has a left adjoint F and the adjunction is monadic*:

$$\begin{array}{ccc} & \overset{F}{\curvearrowright} & \\ \dot\Sigma\text{-Alg} & \dashv & \dot\Psi\text{-Alg} \\ & \underset{T}{\curvearrowleft} & \\ \mathrm{U}_{\dot\Sigma}\big\downarrow \vdash \big\uparrow \mathrm{F}_{\dot\Sigma} & & \mathrm{F}_{\dot\Psi}\big\uparrow \dashv \big\downarrow \mathrm{U}_{\dot\Psi} \\ \mathscr{C} & = & \mathscr{C} \end{array}$$

Proof. The idea of the proof is to construct the left adjoint F via the initial algebra of some other equational system, similarly to how the free algebra for a functor $\Sigma : \mathscr{C} \to \mathscr{C}$ over an object $A \in \mathscr{C}$ can be constructed via the initial algebra of the functor $A + \Sigma -$, except that we need to take equations into account.

Let $\dot\Sigma = (\Sigma \triangleright \Gamma \vdash L = R)$. To construct the free $\dot\Psi$-algebra over a $\dot\Sigma$-algebra $\langle A, \alpha : \Sigma A \to A\rangle$, we consider the equational system

$$\dot\Psi_A := \dot\Psi \upharpoonleft \mathrm{K}_A \upharpoonleft (\mathrm{K}_{\Sigma A} \vdash L' = R') \tag{2.18}$$

where K_X is the constant endofunctor mapping to $X \in \mathscr{C}$, and the two functorial

terms $L', R' : (\Psi + \mathrm{K}_A)\text{-}\textsc{Alg} \to \mathrm{K}_{\Sigma A}\text{-}\textsc{Alg}$ are

$$\begin{aligned} L'\langle B, \beta : \Psi B \to B, i : A \to B\rangle &= \langle B, \Sigma A \xrightarrow{\Sigma i} \Sigma B \xrightarrow{\tilde{T}\beta} B\rangle, \\ R'\langle B, \beta : \Psi B \to B, i : A \to B\rangle &= \langle B, \Sigma A \xrightarrow{\alpha} A \xrightarrow{i} B\rangle. \end{aligned}$$

where $\tilde{T} : \Psi\text{-}\textsc{Alg} \to \Sigma\text{-}\textsc{Alg}$ is obtained from T by Lemma 2.2.2*11. Since $\mathscr{A}$ is assumed to closed under binary coproducts and constant functors, the equational system $\dot{\Psi}_A$ is in $\mathrm{Eqs}_{\mathscr{A}}(\mathscr{C})$ and thus has an initial algebra

$$\langle B_0, \beta : \Psi B_0 \to B_0, i : A \to B_0\rangle.$$

We note that the functorial equation $L' = R'$ encodes a Σ-algebra homomorphism:

$$\begin{array}{ccc} \Sigma A & \xrightarrow{\Sigma i} & \Sigma B \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\tilde{T}\beta} \\ A & \xrightarrow{i} & B \end{array} \tag{2.19}$$

Since $\tilde{T}$ and T coincide on $\dot{\Psi}$ algebras, $\tilde{T}\beta$ is the same as $T\beta$. Therefore $i : A \to B_0$ is a $\dot{\Sigma}$-algebra homomorphism from $\langle A, \alpha\rangle$ to $\langle B_0, T\beta\rangle$.

Next we show that the arrow $i : \langle A, \alpha\rangle \to T\langle B_0, \beta\rangle$ is a universal arrow from $\langle A, \alpha\rangle$ to the functor $T : \dot{\Psi}\text{-}\textsc{Alg} \to \dot{\Sigma}\text{-}\textsc{Alg}$. For every $\langle C, \delta\rangle \in \dot{\Psi}\text{-}\textsc{Alg}$ and an arrow $f : \langle A, \alpha\rangle \to \langle C, T\delta\rangle$, we need to find a unique $\dot{\Psi}$-homomorphism $h : \langle B_0, \beta\rangle \to \langle C, \delta\rangle$ such that $Th \cdot i = f$:

$$\begin{array}{ccc} \langle A, \alpha\rangle & \xrightarrow{i} & T\langle B_0, \beta\rangle \\ & {\scriptstyle f}\searrow & \downarrow{\scriptstyle Th} \\ & & T\langle C, \delta\rangle \end{array} \qquad\qquad \begin{array}{c} \langle B_0, \beta\rangle \\ \vdots \\ \downarrow{\scriptstyle \exists! h} \\ \langle C, \delta\rangle \end{array}$$

We observe that $\langle C, \delta, f\rangle$ is a model of $\dot{\Psi}_A$ (2.18), so by the initiality of $\langle B_0, \beta, i\rangle$, there is an $h : \langle B_0, \beta, i\rangle \to \langle C, \delta, f\rangle$. Since h is a $\dot{\Psi}_A$-homomorphism, we have $h \cdot i = f$. Hence $Th \cdot i = f$ as the translation T preserves homomorphisms.

It remains to show the uniqueness of such $h : \langle B_0, \beta\rangle \to \langle C, \delta\rangle \in \dot{\Psi}\text{-}\textsc{Alg}$ with $h \cdot i = f$. Assuming there is such an h', then h' is also a $\dot{\Psi}_A$-homomorphism from $\langle B_0, \beta, i\rangle$ to $\langle C, \delta, i\rangle$. Therefore $h' = h$ by the initiality of $\langle B_0, \beta, i\rangle$.

We have shown that for every $\dot{\Sigma}$-algebra $\langle A, \alpha\rangle$, there is a universal arrow from $\langle A, \alpha\rangle$ to $T : \dot{\Psi}\text{-}\textsc{Alg} \to \dot{\Sigma}\text{-}\textsc{Alg}$. This extends to an adjunction $F \dashv T$ by [Mac Lane 1998, §IV.1 Theorem 2].

For the monadicity of the adjunction $F \dashv T$, by Beck's monadicity theorem [Borceux 1994b, Theorem 4.4.4], we need to show that (1) the functor T reflects isomorphisms and (2) for a pair of morphisms $h, g : \langle A, \alpha\rangle \to \langle B, \beta\rangle \in \dot{\Psi}\text{-}\textsc{Alg}$ such that Th and Tg have a split coequaliser in $\dot{\Sigma}\text{-}\textsc{Alg}$, h and g have a coequaliser in $\dot{\Psi}\text{-}\textsc{Alg}$ and that is preserved by T.

For (1), the image of a morphism $h : \langle A, \alpha\rangle \to \langle B, \beta\rangle \in \dot\Psi\text{-}\textsc{Alg}$ under the translation functor T is still h (as a $\dot\Sigma$-homomorphism). If h has an inverse h^{-1} in $\dot\Sigma\text{-}\textsc{Alg}$, then h^{-1} is a $\dot\Psi$-homomorphism as well:

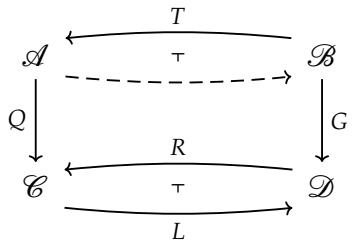
$$h \cdot \alpha = \beta \cdot \Psi h \iff h \cdot \alpha \cdot \Psi h^{-1} = \beta \iff \alpha \cdot \Psi h^{-1} = h^{-1} \cdot \beta$$

So h is also an isomorphism in $\dot\Psi\text{-}\textsc{Alg}$.

For (2), let $e : \langle B, T\beta\rangle \to \langle C, \gamma\rangle$ be the split coequaliser of Th and Tg. By the definition of split coequalisers, $\mathrm{U}_{\dot\Sigma} e$ is a split coequaliser of $h, g : A \to B$ in $\mathscr{C}$. By the monadicity of $\mathrm{F}_{\dot\Psi} \dashv \mathrm{U}_{\dot\Psi}$, there is a coequaliser $e' : \langle B, \beta\rangle \to \langle C', \gamma'\rangle$ of h and g in $\dot\Psi\text{-}\textsc{Alg}$ that is preserved by $\mathrm{U}_{\dot\Psi}$. Since $Th \cdot Te' = Tg \cdot Te'$, there is a morphism $i : \langle C, \gamma\rangle \to \langle C', T\gamma'\rangle$ such that $i \cdot e = Te'$ in $\dot\Sigma\text{-}\textsc{Alg}$. But in the category $\mathscr{C}$, both $\mathrm{U}_\Sigma e : B \to C$ and $\mathrm{U}_\Gamma e' = \mathrm{U}_\Sigma Te : B \to C'$ are coequalisers of $h, g : A \to B$, so $\mathrm{U}_\Sigma i$ is the unique isomorphism between C and C'. As a monadic functor, U_Σ reflects isomorphisms, so i is also an isomorphism in $\dot\Sigma\text{-}\textsc{Alg}$ too, and Te' is also a coequaliser of Th and Tg. □

2.2.2∗15. The two examples of freeness conditions $\mathscr{A}$ in Example 2.2.1∗23 both satisfy the assumption in the theorem. In particular, the theorem applied to the translation $\textsc{Mon} \to \textsc{Grp}$ in Example 2.2.2∗4 constructs free groups over monoids. This theorem will later be used for constructing *free modular models*.

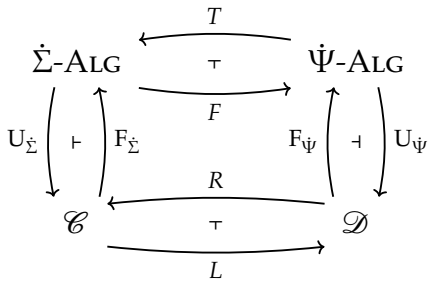
2.2.2∗16. The construction of relative free algebras is a special case of the *adjoint lifting problem* [Borceux 1994b, §4.5], which asks given functors $Q \circ T = R \circ G$ such that R has a left adjoint, whether T has a left adjoint as well?



The situation of Theorem 2.2.2∗14 is then the case where $L \dashv R$ is the identity adjunction $\mathrm{Id} \dashv \mathrm{Id} : \mathscr{C} \to \mathscr{C}$. An answer given by Borceux [1994b, Theorem 4.5.6] is that if G and P are monadic and $\mathscr{B}$ has coequalisers, then T has a left adjoint. Therefore Theorem 2.2.2∗14 can be generalised as follows.

2.2.2∗17 Theorem. *Let $L \dashv R : \mathscr{D} \to \mathscr{C}$ be an adjunction, $\mathscr{A}$ be a freeness condition on $\mathscr{D}$, $\dot\Sigma \in \mathrm{EQS}_f(\mathscr{C})$, and $\dot\Psi \in \mathrm{EQS}_{\mathscr{A}}(\mathscr{D})$. A functor $T : \dot\Psi\text{-}\textsc{Alg} \to \dot\Sigma\text{-}\textsc{Alg}$ such that $\mathrm{U}_{\dot\Psi} \circ T = R \circ \mathrm{U}_{\dot\Psi}$ has a left adjoint $F : \dot\Sigma\text{-}\textsc{Alg} \to \dot\Psi\text{-}\textsc{Alg}$ if either* (1) *$\dot\Psi\text{-}\textsc{Alg}$ has*

coequalisers or (2) $\mathscr{A}$ *is closed under binary coproducts and constant functors.*



Proof sketch. Free-forgetful adjunctions of equational systems are always monadic [Fiore and Hur 2009, Proposition 6.4], so if $\dot{\Psi}$ has coequalisers we have the left adjoint F by Borceux [1994b, Theorem 4.5.6]. Alternatively, if $\mathscr{A}$ satisfies the required property, we can construct the left adjoint in the same way as in the proof of Theorem 2.2.2∗14, inserting L and R in suitable places to go back and forth between $\mathscr{C}$ and $\mathscr{D}$. For example, the crucial diagram (2.19) in the proof of Theorem 2.2.2∗14 should be modified to

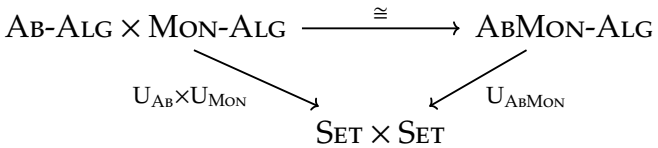
$$\begin{array}{ccc} \Sigma A & \xrightarrow{\Sigma i} & \Sigma RB \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\tilde{T}\beta} \\ A & \xrightarrow{i} & RB \end{array}$$

where $\langle A \in \mathscr{C}, \alpha : \Sigma A \to A\rangle$ and $\langle B \in \mathscr{D}, \beta : \Psi B \to B\rangle$, and the functor $\tilde{T} : \Psi\text{-Alg} \to \Sigma\text{-Alg}$ from Lemma 2.2.2∗11 is modified to

$$\tilde{T}\langle B, \beta\rangle = \langle RB,\ \Sigma RB \xrightarrow{\Sigma R\eta_B} \Sigma R(\mathrm{F}_{\dot{\Psi}}B) \xrightarrow{To} R\mathrm{F}_{\dot{\Psi}}B \xrightarrow{R\epsilon_{B,\beta}} RB\rangle.$$

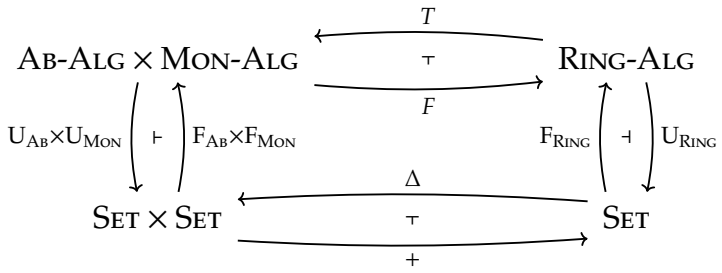
where $o : \dot{\Psi}\mathrm{F}_{\dot{\Psi}}B \to \mathrm{F}_{\dot{\Psi}}B$ is the algebra structure on the free $\dot{\Psi}$-algebra. □

2.2.2∗18 Example. Instantiate the adjunction $L \dashv R$ in Theorem 2.2.2∗17 to be $+ \dashv \Delta : \textsc{Set} \to \textsc{Set} \times \textsc{Set}$. We have an equational system $\textsc{Ring} \in \textsc{Eqs}(\textsc{Set})$ of rings, and we can define an equational system $\textsc{AbMon} \in \textsc{Eqs}(\textsc{Set} \times \textsc{Set})$ such that an algebra of AbMon is a pair $\langle A, M\rangle$ of sets with an abelian group on A and a monoid on M, so we have the following commutative triangle:



Let T : Ring-Alg → Ab-Alg × Mon-Alg be the functor that projecting out the additive group and multiplicative monoid structure of rings. It satisfies

$\mathrm{U}_{\mathrm{Ab}} \times \mathrm{U}_{\mathrm{Mon}} \circ T = \Delta \circ \mathrm{U}_{\mathrm{Ring}}$. By Theorem 2.2.2∗17, T has a left adjoint F:



The functor F generates a free ring *simultaneously* over an abelian group A and a monoid M. More generally, whenever we have a cospan of equational systems $\dot{\Sigma} \to \dot{\Psi} \leftarrow \dot{\Phi}$, we can construct an algebra of $\dot{\Psi}$ out of a $\dot{\Sigma}$-algebra and a $\dot{\Phi}$-algebra in a similar fashion. In the next chapter, we will use this technique to turn an ordinary model of an equational system to a modular model.

2.2.3 Colimits of Equational Systems

2.2.3∗1. Colimits in $\mathrm{Eqs}(\mathscr{C})$ allow us to construct bigger equational systems by 'gluing' smaller ones. For example, the equational system for rings can be obtained by first taking the coproduct of Grp and Mon and then taking a suitable coequaliser $L \rightrightarrows \textsc{Grp} + \textsc{Mon}$ encoding the interaction of operations.

2.2.3∗2 Theorem. *The category* $\mathrm{Eqs}_{\mathscr{A}}(\mathscr{C})$ *is small-cocomplete when the freeness condition* $\mathscr{A}$ *is closed under small-coproducts and if* $\langle \Psi, \Theta \rangle \in \mathscr{A}$ *then* $\langle 0, \Psi \rangle \in \mathscr{A}$.

Proof sketch. Arbitrary colimits can be constructed from coproducts and coequalisers, so it is sufficient to show that $\mathrm{Eqs}_{\mathscr{A}}(\mathscr{C})$ has small-set-indexed coproducts and coequalisers. We sketch the constructions here without proof.

(i) The coproduct of a (small) set of equational systems is obtained by taking the coproduct of signatures and equations. Precisely, if $\langle \Sigma_i \triangleright \Gamma_i \vdash L_i = R_i \rangle_{i \in I}$ is a set of equational systems with each of them in $\mathrm{Eqs}_{\mathscr{A}}(\mathscr{C})$, their coproduct is

$$\coprod_i \Sigma_i \triangleright \coprod_i \Gamma_i \vdash L = R$$

where $L, R : (\coprod_{i \in I} \Sigma_i)\text{-}\textsc{Alg} \to (\coprod_{i \in I} \Gamma_i)\text{-}\textsc{Alg}$ are

$$L\alpha = [L_i(\alpha \cdot \iota_i)]_{i \in I} \qquad\qquad R\alpha = [R_i(\alpha \cdot \iota_i)]_{i \in I}.$$

(ii) Let $T_1, T_2 : \dot{\Psi} \to \dot{\Sigma}$ be a pair of translations. Let $\dot{\Sigma}'$ be

$$\dot{\Sigma} \upharpoonleft (\Psi \vdash \tilde{T}_1 = \tilde{T}_2),$$

where $\tilde{T}_1, \tilde{T}_2 : \Sigma\text{-}\textsc{Alg} \to \Psi\text{-}\textsc{Alg}$ are obtained from T_1 and T_2 by Lemma 2.2.2∗11. By the assumption on $\mathscr{A}$, $\dot{\Sigma}'$ is still in $\mathrm{Eqs}_{\mathscr{A}}(\mathscr{C})$. The inclusion translation $\dot{\Sigma} \to \dot{\Sigma}'$ is the coequaliser of $T_1, T_2 : \dot{\Psi} \to \dot{\Sigma}$. □

2.2.3∗3 Corollary. For every lκp category $\mathscr{C}$, the freeness condition $\mathscr{A} = \textsc{Endo}_\kappa(\mathscr{C}) \times \textsc{Endo}_\kappa(\mathscr{C})$ satisfies the assumption of the theorem, so $\textsc{Eqs}_{\mathscr{A}}(\mathscr{C})$ is cocomplete. For a small-complete small category $\mathscr{C}$, the freeness condition$\mathscr{A} = \textsc{Endo}(\mathscr{C}) \times \textsc{Endo}(\mathscr{C})$ is a freeness condition so $\textsc{Eqs}(\mathscr{C})$ is cocomplete.

2.2.3∗4. Lastly, the following more description of a special case of pushouts of equational system will be convenient in the future.

2.2.3∗5 Lemma. Let $\dot{\Sigma} \in \textsc{Eqs}(\mathscr{C})$ for a category $\mathscr{C}$ with finite coproducts, and let $\Theta_i : \mathscr{C} \to \mathscr{C}$ be an endofunctor and $E_i = (\Theta_i \vdash L_i = R_i)$ be an equation for $i \in \{1, 2\}$. Let T_1 and T_2 in the diagram below be the inclusion translations, then the following is a pushout diagram of T_1 and T_2:

$$\begin{array}{ccc} \dot{\Sigma} & \xrightarrow{\quad T_2 \quad} & \dot{\Sigma} \leftthreetimes \Phi_2 \leftthreetimes E_2 \\ {\scriptstyle T_1}\downarrow & & \downarrow \\ \dot{\Sigma} \leftthreetimes \Phi_1 \leftthreetimes E_1 & \longrightarrow & \dot{\Sigma} \leftthreetimes (\Phi_1 + \Phi_2) \leftthreetimes E \end{array}$$

where $E = (\Theta_1 + \Theta_2 \vdash [L_1 \circ \alpha_1, L_2 \circ \alpha_2] = [R_1 \circ \alpha_1, R_2 \circ \alpha_2])$ and

$$\alpha_i : (\Sigma + (\Phi_1 + \Phi_2))\text{-}\textsc{Alg} \to (\Sigma + \Phi_i)\text{-}\textsc{Alg}$$

are the evident forgetful functors.

Proof. First of all, there are evident inclusion translations P_i (which are the unlabelled arrows in the pushout diagram):

$$P_i : (\dot{\Sigma} \leftthreetimes (\Phi_1 + \Phi_2) \leftthreetimes E)\text{-}\textsc{Alg} \quad \to \quad (\dot{\Sigma} \leftthreetimes \Phi_i \leftthreetimes E_i)\text{-}\textsc{Alg}$$

such that $T_1 \circ P_1 = T_2 \circ P_2$. Now for every equational system $\dot{\Psi} \in \textsc{Eqs}(\mathscr{C})$ with translations functors $Q_i : \dot{\Psi}\text{-}\textsc{Alg} \to (\dot{\Sigma} \leftthreetimes \Phi_i \leftthreetimes E_i)\text{-}\textsc{Alg}$ such that $T_1 \circ Q_1 = T_2 \circ Q_2$, we can define a translation functor

$$U : \dot{\Psi}\text{-}\textsc{Alg} \to (\dot{\Sigma} \leftthreetimes (\Phi_1 + \Phi_2) \leftthreetimes E')\text{-}\textsc{Alg}$$

by sending every $\dot{\Psi}$-algebra $\langle A, \alpha \rangle$ to the algebra on A with structure map:

$$[T_1(Q_1\dot{A}), \quad Q_1\alpha \cdot \iota_2, \quad Q_2\alpha \cdot \iota_2] : (\Sigma + \Phi_1 + \Phi_2)\, A \to A$$

It can be checked that such U is the unique one making $P_i \circ U = Q_i$. □

2.3 A Type Theory for Monoidal Categories

2.3∗1. When categorical constructions get complex, it is beneficial to use type theoretic *internal languages* to describe those constructions in a more intuitive manner [Crole 1994; Jacobs 1999; Lambek and Scott 1986]. This technique will be

prevalent in this thesis. In this section, we introduce *monoidal algebraic theories*, which allows us to present equational systems over monoidal categories and their algebras in a convenient syntactic manner.

2.3∗2. Monoidal algebraic theories are the *linear* counterpart of multi-sorted universal algebra, and the *syntactic* counterpart of *coloured PROs* (strict monoidal categories whose objects are products of a set of base objects) [MacLane 1965]. The presentation of monoidal algebraic theories below is based on the calculus of Jaskelioff and Moggi [2010], but the rules of judgemental equality is strengthened here to make terms form a monoidal category.

2.3.1 Monoidal Algebraic Theories

2.3.1∗1. A *monoidal algebraic theory* $\mathscr{L}$ is specified by three pieces of data $\langle \mathscr{B}, \mathscr{P}, \mathscr{A} \rangle$ as follows. Firstly, $\mathscr{B}$ is a set, and every element $\alpha \in \mathscr{B}$ is called a *base type*. The *types* of $\mathscr{L}$ are inductively generated by the rules

$$\frac{\alpha \in \mathscr{B}}{\vdash \alpha \text{ type}} \qquad \frac{}{\vdash I \text{ type}} \qquad \frac{\vdash A \text{ type} \qquad \vdash B \text{ type}}{\vdash A \square B \text{ type}}$$

Then $\mathscr{P}$ is a family of sets indexed by pairs of types A and B. Every element $f \in \mathscr{P}_{A,B}$ is called a *primitive operation*, and we will write $f : A \to B$ for $f \in \mathscr{P}_{A,B}$.

A *context* Γ is a finite list of variables and types $(x_1 : A_1, \ldots, x_n : A_n)$. The concatenation of two contexts is written as Γ_l, Γ_r.

The (well typed) *terms* under contexts Γ are generated by the following rules:

$$\frac{}{x : A \vdash x : A} \qquad \frac{f : A \to B \in \mathscr{P} \qquad \Gamma \vdash t : A}{\Gamma \vdash f\ t : B} \qquad \frac{\cdot}{\vdash * : I}$$

$$\frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1, \Gamma_2 \vdash (t_1, t_2) : A \square B} \qquad \frac{\Gamma \vdash t_1 : I \qquad \Gamma_l, \Gamma_r \vdash t_2 : A}{\Gamma_l, \Gamma, \Gamma_r \vdash \mathit{let} * = t_1 \mathit{\ in\ } t_2 : A}$$

$$\frac{\Gamma \vdash t_1 : A_1 \square A_2 \qquad \Gamma_l, x_1 : A_1, x_2 : A_2, \Gamma_r \vdash t_2 : B}{\Gamma_l, \Gamma, \Gamma_r \vdash \mathit{let}\ (x_1, x_2) = t_1 \mathit{\ in\ } t_2 : B}$$

Note that the type system is substructural, since the language is to be interpreted in monoidal categories rather than cartesian categories.

Lastly, $\mathscr{A}$ is a set of pairs $\langle \Gamma \vdash t_l : A,\ \Gamma \vdash t_r : A \rangle$ of terms of the same type and under the same context. Every element of $\mathscr{A}$ is called an *axiom* of $\mathscr{L}$. We will write $(\Gamma \vdash t_l = t_r : A) \in A$ when a pair $\langle \Gamma \vdash t_l : A, \Gamma \vdash t_r : A \rangle$ is in $\mathscr{A}$.

2.3.1∗2 Example. The monoidal algebraic theory of monoids has one base type M, two primitive operations $\mu : M \square M \to M$ and $\eta : I \to M$ and the following

axioms, which correspond to the laws of monoids (2.1∗2):

$$x : M \vdash \mu(\eta(*), x) = x : M \qquad x : M \vdash \mu(x, \eta(*)) = x : M$$
$$x : M, y : M, z : M \vdash \mu(\mu(x, y), z) \;=\; \mu(x, \mu(y, z)) : M \tag{2.20}$$

Although these axioms above look the same as the usual laws of monoids in Set, we will see below they can actually be interpreted in all monoidal categories, and a model of this theory will be exactly a monoid in a monoidal category.

2.3.1∗3 Lemma. The following *substitution rule* (or the 'cut rule') is admissible:

$$\frac{\Gamma_l, x : A, \Gamma_r \vdash t : B \qquad \Delta \vdash u : A}{\Gamma_l, \Delta, \Gamma_r \vdash t[u/x] : B}\ \textsc{Cut}$$

where $t[u/x]$ is substituting u for x in t.

Proof sketch. The typing rules of terms above all have substitution 'built-in' by having contexts Γ, Γ_l and Γ_r as general as possible. Thus the substitution rule can be shown by a straightforward induction on t. □

2.3.1∗4 Notation. In this thesis, when we simultaneously substitute terms $u_1, \ldots, u_n$ for all the variables in the context of a term $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$, we usually just write $t[u_1, \ldots, u_n]$ instead of $t[u_1/x_1, \ldots, u_n/x_n]$.

2.3.1∗5. The *judgemental equality* $\Gamma \vdash t_1 \equiv t_2 : A$ of terms of a theory $\mathscr{L}$ is generated by the following rules plus the usual rules for reflexivity, symmetry, transitivity, and congruence under all term formers *and substitution*:

$$\frac{(\Gamma \vdash t_l = t_r : A) \in \mathscr{A}}{\Gamma \vdash t_l \equiv t_r : A}\ \textsc{Ax} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash (\mathit{let} * = * \mathit{in}\ t : A) \equiv t : A}\ I\text{-}\beta$$

$$\frac{\Gamma \vdash t_1 : I \qquad \Gamma_l, x : I, \Gamma_r \vdash t_2 : A}{\Gamma_l, \Gamma, \Gamma_r \vdash (\mathit{let} * = t_1 \mathit{in}\ t_2[*/x]) \equiv t_2[t_1/x] : A}\ I\text{-}\eta$$

$$\frac{\Gamma_1 \vdash t_1 : A_1 \qquad \Gamma_2 \vdash t_2 : A_2 \qquad \Gamma_l, x_1 : A_1, x_2 : A_2, \Gamma_r \vdash t_3 : B}{\Gamma_l, \Gamma_1, \Gamma_2, \Gamma_r \vdash (\mathit{let}\ (x_1, x_2) = (t_1, t_2)\ \mathit{in}\ t_3) \equiv t_3[t_1/x_1, t_2/x_2] : B}\ \Box\text{-}\beta$$

$$\frac{\Gamma \vdash t_1 : A_1 \,\Box\, A_2 \qquad \Gamma_l, x : A_1 \,\Box\, A_2, \Gamma_r \vdash t_2 : B}{\Gamma_l, \Gamma, \Gamma_r \vdash (\mathit{let}\ (x_1, x_2) = t_1\ \mathit{in}\ t_2[(x_1, x_2)/x]) \equiv t_2[t_1/x] : B}\ \Box\text{-}\eta$$

The congruence rule under substitution is

$$\frac{\Gamma_l, x : A, \Gamma_r \vdash t_1 \equiv t_2 : B \qquad \Delta \vdash u : A}{\Gamma_l, \Delta, \Gamma_r \vdash t_1[u/x] \equiv t_2[u/x] : B}$$

2.3.1∗6. A *model* of a monoidal algebraic theory $\mathscr{L} = \langle \mathscr{B}, \mathscr{P}, \mathscr{A} \rangle$ in a monoidal category $\mathscr{E}$ consists of (1) an assignment of $\mathscr{E}$-objects $[\![\alpha]\!] \in \textsc{Ob}(\mathscr{E})$ to each base type $\alpha \in \mathscr{B}$, which induces the interpretation of all types and contexts:

$$[\![I]\!] = I_{\mathscr{E}} \qquad [\![A \,\square\, B]\!] = [\![A]\!] \,\square_{\mathscr{E}}\, [\![B]\!]$$
$$[\![\cdot]\!] = I_{\mathscr{E}} \qquad [\![\Gamma, x : A]\!] = [\![\Gamma]\!] \,\square_{\mathscr{E}}\, [\![A]\!]$$

and (2) an assignment of $\mathscr{E}$-morphisms $[\![f]\!] : [\![A]\!] \to [\![B]\!]$ to each primitive operation $f : A \to B \in \mathscr{P}$, which determines the interpretation of all terms:

$$[\![x]\!] = id \qquad [\![f\ t]\!] = [\![f]\!] \cdot [\![t]\!] \qquad [\![*]\!] = id \qquad [\![(t_1, t_2)]\!] = [\![t_1]\!] \,\square_{\mathscr{E}}\, [\![t_2]\!]$$
$$[\![let * = t_1\ in\ t_2]\!] = [\![t_2]\!] \cdot \_ \cdot ([\![\Gamma_1]\!] \,\square_{\mathscr{E}}\, [\![t_1]\!] \,\square_{\mathscr{E}}\, [\![\Gamma_2]\!])$$
$$[\![let\ (x_1, x_2) = t_1\ in\ t_2]\!] = [\![t_2]\!] \cdot \_ \cdot ([\![\Gamma_l]\!] \,\square_{\mathscr{E}}\, [\![t_1]\!] \,\square_{\mathscr{E}}\, [\![\Gamma_r]\!])$$

where the underscores stand for the unique isomorphisms built from associators α and unitors λ, ρ to make the domain and codomain match. Moreover, the assignments $[\![-]\!]$ of a model must make $[\![t_l]\!] = [\![t_r]\!]$ for all axioms $\langle t_l, t_r \rangle \in \mathscr{A}$.

2.3.1∗7. Let M and N be two models of a theory $\mathscr{L} = \langle \mathscr{B}, \mathscr{P}, \mathscr{A} \rangle$ in a monoidal category $\mathscr{E}$. A family of $\mathscr{E}$-morphisms $h_\alpha : [\![\alpha]\!]_M \to [\![\alpha]\!]_N$ for all base types $\alpha \in \mathscr{B}$ extends to a family of morphisms $\tilde{h}_A : [\![A]\!]_M \to [\![A]\!]_N$ for all $\mathscr{L}$-types A:

$$\tilde{h}_I = id_I \qquad \tilde{h}_{A \square B} = \tilde{h}_A \,\square_{\mathscr{E}}\, \tilde{h}_B \qquad \tilde{h}_\alpha = h_\alpha.$$

Such a family of morphisms h is called a *homomorphism* from M to N if for every primitive operation $f : A \to B \in \mathscr{P}$, the following commutes:

$$\begin{array}{ccc} [\![A]\!]_M & \xrightarrow{[\![f]\!]_M} & [\![B]\!]_M \\ {\scriptstyle \tilde{h}_A}\downarrow & & \downarrow{\scriptstyle \tilde{h}_B} \\ [\![A]\!]_N & \xrightarrow[[\![f]\!]_N]{} & [\![B]\!]_N \end{array}$$

Models of $\mathscr{L}$ in $\mathscr{E}$ and their homomorphisms assemble to a category $\mathscr{L}$-$\textsc{Mod}(\mathscr{E})$.

2.3.1∗8 Theorem (Soundness). *Let $[\![-]\!]$ be a model of a monoidal algebraic theory $\mathscr{L}$. If two $\mathscr{L}$-terms are judgementally equal, $\Gamma \vdash t_1 \equiv t_2 : A$, then $[\![t_1]\!] = [\![t_2]\!]$.*

Proof sketch. It is straightforward verification that the rules of judgemental equalities in 2.3.1∗5 is validated by the axioms of a monoidal category after we show the *substitution lemma*: for all $\Gamma_l, x : A, \Gamma_r \vdash t : B$ and $\Delta \vdash u : A$, we have

$$[\![t[u/x]]\!] \;=\; [\![t]\!] \cdot (\Gamma_l \,\square\, [\![u]\!] \,\square\, \Gamma_r) : [\![\Gamma_l, \Delta, \Gamma_r]\!] \to [\![B]\!]$$

which itself can be proven by induction on t. □

2.3.1∗9. Given a monoidal category $\mathscr{C}$, its *internal language* $\mathscr{L}(\mathscr{C})$ is a monoidal algebraic theory defined as follows.

- The set of base types of $\mathscr{L}(\mathscr{C})$ is exactly $\textsc{Obj}\,\mathscr{C}$, so we have an interpretation of all types as objects in $\mathscr{C}$ by interpreting every base type as itself.
- The set of primitive operations between two types A and B is $\mathscr{C}(\llbracket A \rrbracket, \llbracket B \rrbracket)$. Again, by interpreting every primitive operation as itself, we have an interpretation of all terms in $\mathscr{C}$ as in 2.3.1∗6.
- The set of axioms of $\mathscr{L}(\mathscr{C})$ is the maximal one containing all pairs of terms $\langle t_1, t_2 \rangle$ such that $\llbracket t_1 \rrbracket = \llbracket t_2 \rrbracket$ in $\mathscr{C}$. The canonical interpretation is a model of $\mathscr{L}(\mathscr{C})$ by construction.

The internal language $\mathscr{L}(\mathscr{C})$ of a monoidal category $\mathscr{C}$ is *sound and complete* for reasoning about $\mathscr{C}$: two terms in $\mathscr{L}(\mathscr{C})$ satisfy $t_1 \equiv t_2$ if and only if they are equal under the canonical interpretation in $\mathscr{C}$. Soundness follows from Theorem 2.3.1∗8 and completeness is by the construction of $\mathscr{L}(\mathscr{C})$.

2.3.1∗10. What we have above is enough for our purpose of using a convenient syntax to describe and reason about constructions in monoidal categories, but there are certainly more things can be said about monoidal algebraic theories. Following the standard agenda of categorical logic, we expect the following to be true but leave fleshing them out as future work:

- Types and terms of $\mathscr{L}$ underlie a *classifying monoidal category* $\mathscr{M}(\mathscr{L})$ there is a model of $\mathscr{L}$ in $\mathscr{M}(\mathscr{L})$, and there is an equivalence of categories

  $$\mathscr{L}\text{-}\textsc{Mod}(\mathscr{C}) \cong \textsc{MonCat}_s(\mathscr{M}(\mathscr{L}), \mathscr{C}),$$

  where the right-hand side is the category of strict monoidal functors $\mathscr{M}(\mathscr{L}) \to \mathscr{C}$ and monoidal natural transformations.
- There is a 2-category $\textsc{Mat}$ of monoidal algebraic theories and a 2-equivalence

  $$\textsc{Mat} \underset{\mathscr{M}}{\overset{\mathscr{L}}{\underset{\cong}{\leftarrow\!\!\rightarrow}}} \textsc{MonCat}_s$$

  between $\textsc{Mat}$ and the 2-category of monoidal categories, strict monoidal functor and monoidal natural transformations.

2.3.1∗11 Remark. We have defined the syntax and semantics of monoidal algebraic theories manually. A more modern approach would be directly define monoidal algebraic theories using the framework of *generalised algebraic theories* [Cartmell 1978, 1986] (or the closely related framework of *quotient inductive-inductive types* [Altenkirch and Kaposi 2016; Altenkirch et al. 2018; Kovács 2023]).

Using these frameworks would then directly give us a notion of models of monoidal algebraic theories and syntactic models.

2.3.2 More Type Formers for Monoidal Algebraic Theories

2.3.2∗1. Sometimes we work in monoidal categories with additional structure, and in this case we extend monoidal algebraic theories with new syntax for the additional structure. For example, when we work in left closed monoidal categories, we extend the calculus with a new type former B/A and term formers:

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : B/A} \qquad \frac{\Gamma_1 \vdash t_1 : B/A \qquad \Gamma_2 \vdash t_2 : A}{\Gamma_1, \Gamma_2 \vdash t_1\ t_2 : B}$$

whose interpretation in a model is given by the corresponding structure of the closed monoidal category:

$$[\![\lambda x : A.\, t : B/A]\!] = abst([\![t]\!]) \qquad [\![t_1\ t_2]\!] = ev \cdot ([\![t_1]\!] \square_{\mathscr{E}} [\![t_2]\!])$$

where $abst : \mathscr{E}(C \square_{\mathscr{E}} A, B) \to \mathscr{E}(C, B/_{\mathscr{E}}A)$ is the natural isomorphism associated to the adjunction $(- \square_{\mathscr{E}} A) \dashv (-/_{\mathscr{E}}A)$ and $ev : (B/_{\mathscr{E}}A) \square_{\mathscr{E}} A \to B$ is its counit. The usual β and η rules characterising the universal property of B/A are added to the equational theory routinely.

2.3.2∗2. Note that the type former B/A is *contravariant* in the position of A, so in the presence of /-types, the way in 2.3.1∗7 of extending a family of morphisms $h_\alpha : [\![\alpha]\!]_M \to [\![\alpha]\!]_N$ between interpretations of base types to a family of morphisms $\tilde{h}_A$ between interpretations of all types will not work, and we will not have a category of models and *model homomorphisms* for a theory $\mathscr{L}$ with /-types. However, we can still have a category $\mathscr{L}$-$\textsc{Mod}_{\cong}(\mathscr{E})$ of models and *model isomorphisms* by demanding that every h_α must be an $\mathscr{E}$-isomorphism.

2.3.2∗3. As a comment on the notation, Jaskelioff and Moggi [2010] used B^A for the closed structure, while we use Lambek's [1958] notation B/A to avoid the confusion with exponentials (which are right adjoints to cartesian products).

2.3.2∗4 Example. Consider the theory of monoids from Example 2.3.1∗2 and the extension of the closed structure B/A as in 2.3.2∗1. *Cayley's theorem* from elementary algebra adapted to this setting says that the monoid $\langle M, \mu, \eta\rangle$ embeds into the monoid M/M with unit $\vdash \lambda x.\, x : M/M$ and multiplication

$$f : M/M,\ g : M/M \vdash \lambda x.\, f\ (g\ x) : M/M. \tag{2.21}$$

The embedding is given by $e = (x : M \vdash \lambda y.\, \mu(x, y) : M/M)$, which is a monoid

homomorphism in the following sense:

$$e[\eta\ *] \equiv \lambda y.\ \mu(\eta\ *, y) \equiv \lambda y.\ y$$
$$e[\mu(x,y)] \equiv \lambda z.\ \mu(\mu(x,y),z) \equiv \lambda z.\ \mu(x,\mu(y,z)) \equiv \lambda z.\ e[x]\ (e[y]\ z)$$

where we write $t[-]$ for substitution when there is a unique variable in the context of t. The embedding e has a left inverse $r = (f : M/M \vdash f\ (\eta\ *) : M)$ such that

$$x : M \vdash r[e] \equiv (\lambda y.\ \mu(x,y))\ (\eta\ *) \equiv x : M$$

However, the inverse r is in general *not* a monoid homomorphism: in the context $(f : M/M, g : M/M)$, we have

$$r[\lambda x.\ f\ (g\ x)] \equiv f\ (g\ (\eta\ *)) \not\equiv \mu(f\ (\eta\ *),\ g\ (\eta\ *)) \equiv \mu(r[f], r[g])$$

This elementary result has surprisingly many applications in functional programming because the multiplication (2.21) is usually a kind of function composition with $O(1)$ time complexity, regardless of the possibly expensive multiplication μ. When M is a free monoid in $\langle \textsc{Set}, \times, 1\rangle$, i.e. a list, this optimisation is known as *difference lists* [Hughes 1986]. When $\mathscr{E} = \langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$, this optimisation is known as *codensity transformation* [Hinze 2012].

2.3.2∗5. Cartesian products and coproducts in monoidal categories can be internalised similarly: we add type formers $A \times B$ and $A + B$ with term formers:

$$\frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \langle t_1, t_2\rangle : A_1 \times A_2}$$

$$\frac{\Gamma \vdash t_1 : A_1 \times A_2 \qquad \Gamma_l, x : A_i, \Gamma_r \vdash t_2 : B}{\Gamma_l, \Gamma, \Gamma_r \vdash t_2[\pi_i\ t_1/x] : B}\ i \in \{1,2\}$$

$$\frac{\Gamma \vdash t : A_i}{\Gamma \vdash \iota_i\ t : A_1 + A_2}\ i \in \{1,2\}$$

$$\frac{\Gamma \vdash t : A_1 + A_2 \qquad x_i : A_i \vdash t_i : C, i \in \{1,2\}}{\Gamma \vdash \mathit{case}\ t\ \mathit{of}\ \{\iota_1\ x_i \mapsto t_1;\ \iota_2\ x_2 \mapsto t_2\} : C}$$

as well as their β and η rules. Sometimes we also write $[t_1, t_2]\ t$ instead of $\mathit{case}\ t\ \mathit{of}\ \{\iota_1\ x_i \mapsto t_1;\ \iota_2\ x_2 \mapsto t_2\}$ for brevity.

These rules straightforwardly generalise to the non-binary cases. Note that the first rule introduces non-linearity to the syntax as the variables in Γ may appear in both subterms t_1 and t_2.

2.3.2∗6 Remark. In the presence of cartesian products, we can borrow the idea

of *bunched logic* [O'Hearn and Pym 1999] to introduce a new context former $\Gamma;\Gamma'$ with semantics $[\![\Gamma;\Gamma']\!] = [\![\Gamma]\!] \times [\![\Gamma']\!]$. This would simplify talking about linear functions $-/A$ (right adjoint to $- \square A$) and non-linear functions $-^A$ (right adjoint to $- \times A$) at the same time, but we will not need this extension in this thesis.

2.3.3 Syntactic Presentations of Equational Systems

2.3.3∗1. In this subsection, we carve out a class of monoidal algebraic theories that can be defined as equational systems, therefore allowing us to apply the theorems of equational systems to those monoidal algebraic theories, or from the opposite perspective, allowing us to define equational systems syntactically use those monoidal algebraic theories.

2.3.3∗2 Definition. Let $\mathscr{L} = \langle \mathscr{B}, \mathscr{P}, \mathscr{A} \rangle$ be a monoidal algebraic theory. When all primitive operations $f : A \to B$ and equations $\Gamma \vdash t_l = t_r : B$ of a monoidal algebraic theory $\mathscr{L}$ satisfy that $B \in \mathscr{B}$, we say that $\mathscr{L}$ has *basic outputs*.

2.3.3∗3. For example, the theory of monoids in Example 2.3.1∗2 has basic outputs, but the internal language $\mathscr{L}(\mathscr{C})$ of a monoidal category $\mathscr{C}$ does not have basic outputs because not all axioms of $\mathscr{L}(\mathscr{C})$ have a base type B.

2.3.3∗4 Theorem. *For every monoidal algebraic theory $\mathscr{L} = \langle \mathscr{B}, \mathscr{P}, \mathscr{A} \rangle$ with basic outputs and every monoidal category $\mathscr{C}$ with small-coproducts, there is an equational system $\Sigma_{\mathscr{L}}$ over the product category $\mathscr{C}^{\mathscr{B}}$ such that $\mathscr{L}$-Mod$(\mathscr{C}) \cong \Sigma_{\mathscr{L}}$-Alg.*

Proof. Recall that types A of $\mathscr{L}$ are generated by I, $\square$, and $\alpha \in \mathscr{B}$. Therefore the interpretation of every type A in $\mathscr{C}$ determines a functor $[\![A]\!] : \mathscr{C}^{\mathscr{B}} \to \mathscr{C}$:

$$[\![I]\!]X = I_{\mathscr{C}} \qquad [\![A_1 \square A_2]\!]X = [\![A_1]\!]X \square [\![A_2]\!]X \qquad [\![\alpha]\!]X = X_\alpha$$

Similarly, every context Γ of $\mathscr{L}$ determines a functor $[\![\Gamma]\!] : \mathscr{C}^{\mathscr{B}} \to \mathscr{C}$.

We define an endofunctor $[\![\mathscr{P}]\!] : \mathscr{C}^{\mathscr{B}} \to \mathscr{C}^{\mathscr{B}}$ by

$$[\![\mathscr{P}]\!]X = \langle \coprod_A \coprod_{f:A\to\alpha\in\mathscr{P}} [\![A]\!]X \rangle_{\alpha\in\mathscr{B}}.$$

It can be seen that an algebra of the endofunctor $[\![\mathscr{P}]\!]$ is precisely an interpretation of base types $\mathscr{B}$ and primitive operations $\mathscr{P}$ in $\mathscr{C}$.

Let $\uparrow_\alpha : \mathscr{C} \to \mathscr{C}^{\mathscr{B}}$ be the functor mapping every $X \in \mathscr{C}$ and $\beta \in \mathscr{B}$ to X if $\alpha = \beta$, or to $0_{\mathscr{C}}$ if $\alpha \neq \beta$. We write $[\![\Gamma \vdash \alpha]\!] : \mathscr{C}^{\mathscr{B}} \to \mathscr{C}^{\mathscr{B}}$ for $\uparrow_\alpha \circ [\![\Gamma]\!]$. An algebra of the endofunctor $[\![\Gamma \vdash \alpha]\!]$ is then an object $X \in \mathscr{C}^{\mathscr{B}}$ with a morphism in $\mathscr{C}^{\mathscr{B}}$

$$\langle \uparrow_\alpha ([\![\Gamma]\!]X)\ \beta \rangle_{\beta\in\mathscr{B}} \longrightarrow \langle X_\beta \rangle_{\beta\in\mathscr{B}},$$

which amounts to just an $\mathscr{C}$-morphism $[\![\Gamma]\!]X \to X_\alpha$, i.e. $[\![\Gamma]\!]X \to [\![\alpha]\!]X$, since all other components are the unique $0_{\mathscr{C}} \to X_\beta$. Therefore the interpretation

of every term $\Gamma \vdash t : \alpha$ of $\mathscr{L}$ with $\alpha \in \mathscr{B}$ then determines a mapping from a $[\![\mathscr{P}]\!]$-algebra to a $[\![\Gamma \vdash \alpha]\!]$-algebra. By induction on the term t in the style of Reynolds's [1983] *abstraction theorem*, it can be shown that this mapping extends to a functor $[\![t]\!] : [\![\mathscr{P}]\!]\text{-}\textsc{Alg} \to [\![\Gamma \vdash \alpha]\!]\text{-}\textsc{Alg}$ satisfying $\mathrm{U}_{[\![\Gamma\vdash\alpha]\!]} \circ [\![t]\!] = \mathrm{U}_{[\![\mathscr{P}]\!]}$.

Let $\Sigma_{\mathscr{L}}$ be the equational system over $\mathscr{E}^{\mathscr{B}}$ with signature $[\![\mathscr{P}]\!]$ and a set of functorial equations $[\![\Gamma]\!] \vdash [\![t_l]\!] = [\![t_r]\!]$ for every axiom $(\Gamma \vdash t_l = t_r : \alpha) \in \mathscr{A}$ (c.f. Notation 2.2.1∗7). By construction, the category $\mathscr{L}\text{-}\textsc{Mod}(\mathscr{E})$ of $\mathscr{L}$-models in $\mathscr{E}$ and the category $\Sigma_{\mathscr{L}}\text{-}\textsc{Alg}$ of $\Sigma_{\mathscr{L}}$-algebras (in $\mathscr{E}^{\mathscr{B}}$) are isomorphic. □

2.3.3∗5 Corollary. Let $\mathscr{E}$ be a cocomplete monoidal category with $\Box$ preserving colimits of α-chains for a limit ordinal α. By the theorem above and Theorem 2.2.1∗12, the category of models $\mathscr{L}\text{-}\textsc{Mod}(\mathscr{E})$ of a monoidal algebraic theory $\mathscr{L}$ with basic outputs is cocomplete and monadic over the category $\mathscr{E}^{\mathscr{B}}$.

2.3.3∗6 Example. The monoidal algebraic theory of monoids in Example 2.3.1∗2 has basic outputs, and the corresponding equational system obtained using the theorem above is precisely the equational system in Example 2.2.1∗6.

2.3.3∗7 Remark. The restriction of basic outputs is reminiscent of the relationship between *operads* and *PROPs* [Markl 2006], which are two frameworks for doing algebraic theories in symmetric monoidal categories. It is also the case that the former allows multiple inputs but one output, whereas the latter allows multiple inputs and multiple outputs. However, an advantage of the restricted frameworks is that the category of algebras of operads/monoidal algebraic theories with basic outputs is monadic (under the conditions of Corollary 2.3.3∗5).

2.3.3∗8. When considering monoidal algebraic theories with extra type formers that are not covariant, such as the linear function type B/A in 2.3.2∗1, Theorem 2.3.3∗4 must additionally require a theory $\mathscr{L} = \langle \mathscr{B}, \mathscr{P}, \mathscr{A} \rangle$ with basic outputs to have *positive inputs*: every primitive operation $f : A \to \alpha$ and equation $\Gamma \vdash t_l = t_r : \alpha$ of $\mathscr{L}$ must satisfy that every base type $\beta \in \mathscr{B}$ occurs *positively* in A and Γ, so that A and Γ can be interpreted as covariant functors $\mathscr{E}^{\mathscr{B}} \to \mathscr{E}$.

The rule for types in which $\beta \in \mathscr{B}$ occurs positively P and negatively N is defined inductively by the following grammar as usual:

$$\begin{aligned} P &:= \beta \mid \alpha \mid P \,\Box\, P \mid P/N \\ N &:= \quad \alpha \mid N \,\Box\, N \mid N/P \end{aligned}$$

where α ranges over $\mathscr{B} \setminus \{\beta\}$. A base type β occurs positively in a context Γ if it occurs positively in every item of the context Γ. Other covariant type formers such as $\times$ and $+$ in 2.3.2∗5 should be treated similarly to $\Box$.

2.3.3∗9. As explained above, type expressions P in which base types occur positively denote functors $[\![P]\!]$. It will be convenient if we also have access in the internal language to the corresponding mappings on morphisms of those functors $[\![P]\!]$. Let $\tau \in \mathscr{B}$ and $x : A \vdash f : B$. For all types P and N in which $\tau \in \mathscr{B}$ occurs positively and negatively respectively, we define terms

$$x : P[A/\tau] \vdash P_\tau f : P[B/\tau] \qquad\qquad x : N[B/\tau] \vdash N_\tau f : N[A/\tau]$$

by induction on the structure of P and N:

$$\tau_\tau f := f \qquad \alpha_\tau f := x \qquad (P/N)_\tau f := \lambda y.\ P_\tau f\ [x\ (N_\tau f\ [y])]$$
$$(P \,\square\, P')_\tau f := \mathit{let}\ (x_l, x_r) = x\ \mathit{in}\ (P_\tau f\ [x_l], P'_\tau f\ [x_r])$$

and symmetrically for N. When the type expression P contains (at most) one base type τ, we will just write Pf for $P_\tau f$.

As a slight abuse of notation, we will write $P_\tau f\ t$ for $P_\tau f[t/x]$, so we have the following admissible rule:

$$\frac{x : A \vdash f : B \qquad \Gamma \vdash t : P[A/\tau]}{\Gamma \vdash P_\tau f\ t : P[B/\tau]}$$

As an example, if $\mathscr{B} = \{\tau\}$, then τ occurs positively in the type expression $T = \tau \,\square\, \tau$, which denotes the functor $\square : \mathscr{C} \times \mathscr{C} \to \mathscr{C}$. Moreover we have $\Gamma \vdash Tf\ t : B \,\square\, B$ for all terms $A \vdash f : B$ and $\Gamma \vdash t : A \,\square\, A$.

2.3.3∗10. A notable difference between a monoidal algebraic theory $\mathscr{L}$ and an equational system $\dot{\Sigma}$ over a monoidal category $\mathscr{C}$ is that the definition of the former (2.3.1∗1) makes no reference to any specific monoidal category, while the definition of the latter (2.2.1∗4) is parameterised by the underlying category $\mathscr{C}$. Therefore the operations of $\mathscr{L}$ must make sense for all monoidal categories. Such a generality can also be a limitation: sometimes we may be interested in operations that only make sense for a specific monoidal category $\mathscr{C}$. In this case, we need a slight generalisation of Theorem 2.3.3∗4 to use monoidal algebraic theories to denote equational systems over $\mathscr{C}$ syntactically.

2.3.3∗11. Let $\mathscr{L} = \langle \mathscr{B}, \mathscr{P}, \mathscr{A} \rangle$ be a monoidal algebraic theory and M be a model of it in some monoidal category $\mathscr{C}$. We say that another theory $\mathscr{L}' = \langle \mathscr{B}', \mathscr{P}', \mathscr{A}' \rangle$ is an *extension* of $\mathscr{L}$ if $\mathscr{B} \subseteq \mathscr{B}'$, $\mathscr{P} \subseteq \mathscr{P}'$, and $\mathscr{A} \subseteq \mathscr{A}'$, where we implicitly treat types/terms generated by $\mathscr{B}$ and $\mathscr{P}$ as types/terms generated generated by the superset $\mathscr{B}'$ and $\mathscr{P}'$. Moreover, a model M' of $\mathscr{L}'$ in $\mathscr{C}$ is said to be *over* the model M if $[\![A]\!]_{M'} = [\![A]\!]_M$ and $[\![t]\!]_{M'} = [\![t]\!]_M$ for all types A and terms t of $\mathscr{L}$.

For example, let $\mathscr{C}$ be a monoidal category and $A \in \mathscr{C}$ be an object of $\mathscr{C}$. Let $\mathscr{L}'$ be the extension of the internal language $\mathscr{L}(\mathscr{C})$ with a new base type τ and a

new operation $f : A \to \tau$. A model of $\mathscr{L}'$ over the canonical model of $\mathscr{L}(\mathscr{E})$ in $\mathscr{E}$ is precisely an object $X \in \mathscr{E}$ with a morphism $f : A \to X$.

2.3.3∗12. We say that an extension $\mathscr{L}'$ of $\mathscr{L}$ has basic outputs if every new operation $(f : A \to B) \in (\mathscr{P}' \setminus \mathscr{P})$ and every equation $(\Gamma \vdash t_l = t_r : B) \in (\mathscr{A}' \setminus \mathscr{A})$ satisfies that $B \in \mathscr{B}' \setminus \mathscr{B}$. Similarly, we say that the extension $\mathscr{L}'$ has positive inputs if every $\alpha \in \mathscr{B}' \setminus \mathscr{B}$ occurs positively in A and Γ.

2.3.3∗13 Theorem. *Let $\mathscr{L}$ be a monoidal algebraic theory (with possibly extra type formers such as $/$, $+$, $\times$-types in Section 2.3.2), and let M be a model of $\mathscr{L}$ in a monoidal category $\mathscr{E}$ with small-coproducts. Every extension $\mathscr{L}'$ of $\mathscr{L}$ that has basic outputs (and positive inputs) determines an equational system $\Sigma_{\mathscr{L}',M}$ over $\mathscr{E}^{\mathscr{B}\setminus\mathscr{B}'}$ such that algebras of $\Sigma_{\mathscr{L}',M}$ are in bijection with models of $\mathscr{L}'$ in $\mathscr{E}$ over M.*

Proof. The proof goes almost the same as Theorem 2.3.3∗4, except that the base types and primitive operations of $\mathscr{L}$ are fixed by the model M. □

2.3.3∗14. Our main use case of Theorem 2.3.3∗13 is when $\mathscr{L}$ is the internal language $\mathscr{L}(\mathscr{E})$ of a monoidal category with coproducts $\mathscr{E}$ (2.3.1∗9), and M is the canonical model, and there is exactly one new base type: $\mathscr{B}' \setminus \mathscr{B} = \{\tau\}$ of $\mathscr{L}(\mathscr{E})$ in $\mathscr{E}$. This allows us to present an equational system over $\mathscr{E}$ syntactically by a set of operations $f : A \to \tau$ and axioms $\Gamma \vdash t_l = t_r : \tau$ where A, Γ, t_l and t_r can refer to the existing objects and morphisms in $\mathscr{E}$.

2.4 Equational Systems for Monoids with Operations

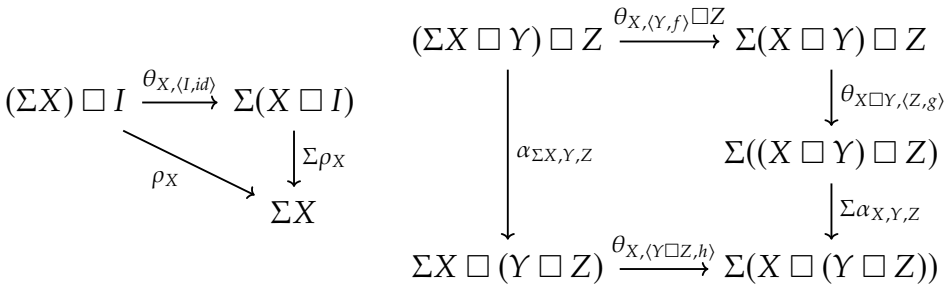
2.4∗1. After the detour to monoidal algebraic theories, now we come back to monoids with operations, by which we mean the equational system Mon extended with a new operation $op : \Sigma M \to M$ for some endofunctor $\Sigma : \mathscr{E} \to \mathscr{E}$ and possibly some equations. In this section, we will discuss a general notion known as Σ-*monoids* in the literature [Fiore et al. 1999], which demands that the operation op is compatible with the monoid multiplication in a sense. After this, we will see some concrete examples.

2.4∗2. Let $\mathscr{E}$ be a monoidal category and $\Sigma : \mathscr{E} \to \mathscr{E}$ be an endofunctor. A *pointed strength* θ for Σ is a natural transformation

$$\theta_{X,\langle Y,f\rangle} : (\Sigma X) \,\square\, Y \to \Sigma(X \,\square\, Y)$$

for all X in $\mathscr{E}$ and $\langle Y \in \mathscr{E}, f : I \to Y\rangle$ in the coslice category $I/\mathscr{E}$, satisfying

coherence conditions analogous to those of strengths (2.4, 2.5):



for all $X, Y, Z \in \mathscr{C}$, $f : I \to Y$, $g : I \to Z$, and $h := (f \square g) \cdot \rho_I^{-1} : I \to Y \square Z$.

2.4∗3. To denote Σ and θ syntactically, we extend the internal language $\mathscr{L}(\mathscr{C})$ (2.3.1∗9) with a new type constructor Σ and the following typing rules

$$\frac{x : A \vdash f : B \qquad \Gamma \vdash t : \Sigma A}{\Gamma \vdash \Sigma f\ t : \Sigma B} \qquad \frac{\cdot \vdash f : Y \qquad \Gamma \vdash t : (\Sigma X) \square Y}{\Gamma \vdash \theta_{X,\langle Y,f\rangle}\ t : \Sigma(X \square Y)}$$

which of course are interpreted in $\mathscr{C}$ by the functor Σ and the strength θ.

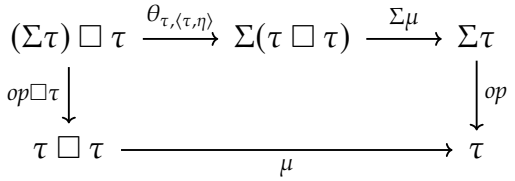
2.4∗4. The equational system Σ-Mon of Σ-*monoids* [Fiore and Hur 2009; Fiore et al. 1999] extends the theory Mon of monoids (2.2.1∗6 and 2.3.1∗2) with a new operation $op : \Sigma\tau \to \tau$ and a new equation $L_{\Sigma\text{-Mon}} = R_{\Sigma\text{-Mon}}$:

$$\Sigma\text{-Mon} = \text{Mon} \curlywedge \Sigma \curlywedge \left(\Sigma - \square - \vdash L_{\Sigma\text{-Mon}} = R_{\Sigma\text{-Mon}}\right) \tag{2.22}$$

where the new equation $L_{\Sigma\text{-Mon}} = R_{\Sigma\text{-Mon}}$ is given in terms of an extension of the internal language of $\mathscr{C}$ by the following pair of terms:

$$x : \Sigma\tau, y : \tau \vdash \mu\ (op\ x, y) = op\ (\Sigma\mu\ (\theta_{\tau,\langle\tau,\eta\rangle}\ (x, y))) : \tau \tag{2.23}$$

which encodes the following commutative diagram:



Note that (2.23) refers to θ, so technically the equational system should be denoted by $\langle\Sigma, \theta\rangle$-Mon, but writing Σ-Mon is unlikely to cause confusion.

2.4∗5. Equation (2.23) expresses that the operation *op* commutes with monoid multiplication. When using the monoidal category $\langle \text{End}_f(\text{Set}), \bullet, V\rangle$ for modelling higher-order abstract syntax, which was the original context where Fiore et al. [1999] introduced Σ-monoids, this equation expresses the sensible condition that syntactic operations must commute with substitution.

However, this equation might not be desirable in other contexts. For example, in the study of *modal type theories*, there are plenty of operations that do not commute with substitution [Gratzer 2023]. Also, when using $\langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$ to model computational effects, this equation expresses that effectful operations must commute with sequential composition, which is not true in general. Those effectful operations that do satisfy this condition and have signature $\Sigma = A\,\square\,-$ for some $A \in \mathscr{C}$ are called *algebraic operations* [Jaskelioff and Moggi 2010; Plotkin and Power 2001a]. We will say more about equation (2.23) shortly in Lemma 2.4∗13 and see that imposing it on Σ-monoids actually does not lose generality.

2.4∗6. When the monoidal category $\mathscr{C}$ is cocomplete and functors Σ, $\square$ both preserve colimits of α-chains for some limit ordinal α, Theorem 2.2.1∗12 ensures the existence of free Σ-monoids. When $\mathscr{C}$ is additionally closed, such as $\langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$ in Section 2.1, there is a simple description of the free Σ-monoid [Fiore and Hur 2007]: it is carried by the initial algebra $\mu X.\ I + A\,\square\,X + \Sigma X$. This formula has many applications in modelling abstract syntax: variable binding [Fiore and Szamozvancev 2022], explicit substitution [Ghani et al. 2006], and scoped operations [Piróg et al. 2018].

2.4∗7 Remark. Given a strong monad T over $\mathscr{C}$, Piróg [2016], Fiore and Saville [2017] studied a notion akin to Σ-monoids in 2.4∗4 (with $\Sigma = T$) but additionally asking the operation $op : T\tau \to \tau$ to be an Eilenberg-Moore algebra of T. This concept is called *Eilenberg-Moore monoids* by Piróg [2016] and *T-monoids* by Fiore and Saville [2017]. When $\mathscr{C}$ is closed, there is again a simple formula $\mu X.\ T(I + A\,\square\,X)$ for the free T-monoid/Eilenberg-Moore monoid over $A \in \mathscr{C}$. Thinking of a monad T as an algebraic theory, using T-monoids instead of Σ-monoids allows us to impose equational laws on the operation op. In this thesis we have opted in to use equational systems to present equations, so in the following we will continue with using Σ-monoids (with possibly additional equations). This has an additional advantage that we can have equations that involve both the monoid operations η, μ and the operation op at the same time, rather than just equations involving op.

2.4∗8. Now let us look at some concrete examples. In all the following examples, the monoidal category $\langle \mathscr{C}, \square, I\rangle$ is assumed to have small-coproducts $\coprod_{i\in S} A_i$ and finite products $\prod_{i\in F} A_i$. Additionally, we assume the monoidal product distributes over coproducts from the right:

$$(\textstyle\coprod_{i\in S} A_i)\,\square\,B \;\cong\; \coprod_{i\in S}(A_i\,\square\,B).$$

2.4∗9 Example. Let S be a set. The theory $\textsc{St}_S$ of *monads with global S-state* [Plotkin and Power 2002] can be generally defined for monoids as follows. The theory

St_S is Σ_{St_S}-Mon with signature Σ_{St_S} denoted by the type expression:

$$((\prod_S I) \square \tau) + ((\coprod_S I) \square \tau),$$

whose first component represents an operation $g : (\prod_S I) \square \tau \to \tau$ reading the state, and the second component represents an operation $p : (\coprod_S I) \square \tau \to \tau$ writing an S-value into the state.

Plotkin and Power's [2002] equations of these two operations can also be specified at this level of generality. For example, the law saying that writing $s \in S$ to the state and reading it immediately gives back s is

$$k : \prod_S I \vdash p_s(g(k, \eta^\tau)) = p_s(\mathit{let} * = \pi_s k \ \mathit{in}\ \eta^\tau) : \tau$$

where $p_s(x)$ abbreviates $p(\iota_s *, x)$.

2.4∗10 Example (Exception throwing). Letting E be a set, the theory Et_E of *exception throwing* is the theory Σ_{Et_E}-Mon where $\Sigma_{\mathrm{Et}_E} = (\coprod_E 1) \square - : \mathscr{C} \to \mathscr{C}$ equipped the associator $\alpha : ((\coprod_E 1) \square X) \square Y \to \coprod_E 1 \square (X \square Y)$ as the strength, where $\coprod_E 1$ is the E-fold coproduct of the terminal object in $\mathscr{C}$ (which may be different from the monoidal unit I).

For the case $\mathscr{C} = \langle \mathrm{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$, the equational system Et_E describes (κ-accessible) monads $M : \mathscr{C} \to \mathscr{C}$ equipped with a natural transformation

$$\mathit{throw} : (\coprod_E 1) \circ M = \coprod_E (1 \circ M) = \coprod_E 1 \quad \longrightarrow \quad M \tag{2.24}$$

whose component $1 \to M$ for each $e \in E$ represents a computation throwing an exception e. Working in the generality of monoids allows us to generalise exceptions to more settings: taking $\mathscr{C} = \langle \mathrm{Endo}_\kappa(\mathrm{Set}), *, \mathrm{Id} \rangle$, the theory describes applicative functors F with exception throwing:

$$(\coprod_E 1) * F \cong \coprod_E (1 * F) = \coprod_E (\textstyle\int^{a,b} 1a \times Fb \times -^{a \times b}) \cong \coprod_E (\int^b Fb) \quad \to \quad F \tag{2.25}$$

Note that exception throwing for monads (2.24) and for applicatives (2.25) differ by the domain: 1 vs $\int^b Fb$. This reflects the nature of applicative functors that computations are independent, so the computation after exception throwing is not necessarily discarded.

2.4∗11. Although exception *throwing* is an algebraic operation, exception *catching* is not: if we were to model it as an operation $\mathit{catch} : M \times M \to M$ on a monad M such that $\mathit{catch}\ \langle p, h \rangle$ means catching exceptions possibly thrown by p and handling exceptions using h, then equation (2.23) for $\Sigma M = M \times M$ with the isomorphism $(M \times M) \circ Y \cong (M \circ Y) \times (M \circ Y)$ as the strength implies that

$$ph : M \times M, k : M \vdash \mu(\mathit{catch}\ ph, k) = \mathit{catch}\ \langle \mu(\pi_1\ ph,\ k), \mu(\pi_2\ ph,\ k) \rangle : M \tag{2.26}$$

But this is undesirable because the scopes of catching on the two sides of the

equation are different: the left-hand side does not catch exceptions in k while the right-hand side catches exceptions in k.

2.4∗12. Plotkin and Pretnar's [2009; 2013] take on this problem is that catching is inherently different from throwing: throwing is the only *operation* of the theory of exceptions, but catching is a *model* of the theory. This view leads to the fruitful line of research on *handlers of algebraic effects*.

Wu et al. [2014] proposed an alternative perspective: *catch* is still an operation in the theory of monads supporting the effect of exceptions, but its signature should be $(M \times M) \circ M \to M$ rather than $M \times M \to M$. Their perspective can be justified by the following observation.

2.4∗13 Lemma. Let $\langle \mathscr{E}, \Box, I \rangle$ be a monoidal category. For all functors $\Phi : \mathscr{E} \to \mathscr{E}$ and monoids $\langle M, \mu, \eta \rangle$ in $\mathscr{E}$, define $\Sigma = (\Phi -) \Box -$ and a pointed strength:

$$(\Sigma X) \Box Y \xrightarrow{\cong} \Phi(X \Box I) \Box (X \Box Y) \xrightarrow{\Phi(X \Box \eta^Y) \Box id} \Phi(X \Box Y) \Box (X \Box Y) = \Sigma(X \Box Y).$$

Then morphisms $f : \Phi M \to M$ *without any condition* are in bijection with morphisms $g : \Sigma M \to M$ that satisfy the compatibility equation (2.23) of Σ-monoids instantiated with the Σ.

Proof. The bijection between f and g is given by

$$f \mapsto (\Phi M \Box M \xrightarrow{f \Box M} M \Box M \xrightarrow{\mu} M) \qquad g \mapsto (\Phi M \xrightarrow{\Phi M \Box \eta} \Phi M \Box M \xrightarrow{g} M).$$

The reader is encouraged to use the internal language of $\mathscr{E}$ to check the round-trip property of this bijection, and below is a proof using the traditional notation. If we start with $f : \Phi M \to M$, after a round-trip we get the morphism

$$\Phi M \xrightarrow{\Phi M \Box \eta} \Phi M \Box M \xrightarrow{f \Box M} M \Box M \xrightarrow{\mu} M,$$

which, by the naturality of η, is equal to $(\mu \cdot \eta) \cdot f = f$. If we start with a morphism $g : \Sigma M \to M$ satisfying (2.23), after a round-trip we get the morphism

$$\Phi M \Box M \xrightarrow{\Phi M \Box \eta \Box M} \Phi M \Box M \Box M \xrightarrow{g \Box M} M \Box M \xrightarrow{\mu} M. \tag{2.27}$$

By the definition of the pointed strength of Σ above, equation (2.23) instantiates to the equation of the following two morphisms $\Phi M \Box M \Box M \to M$:

$$\mu \cdot (g \Box M) = g \cdot (\Phi \mu \Box \mu) \cdot (\Phi(M \Box \eta) \Box M \Box M)$$

The right-hand side is equal to $g \cdot (\Phi M \Box \mu)$, so (2.27) is equal to $g \cdot (\Phi M \Box \mu) \cdot (\Phi M \Box \eta \Box M) = g \cdot (\Phi M \Box (\mu \cdot (\eta \Box M))) = g \cdot (\Phi M \Box id) = g$. □

2.4∗14. Therefore, using Lemma 2.4∗13, exception catching on a monad can still be formulated as a Σ-monoid with $\Sigma M := (M \times M) \circ M = (\mathrm{Id} \times \mathrm{Id}) \circ M \circ M$. More

generally, operations $s : A \circ M \circ M \to M$ satisfying (2.23) for an endofunctor A, or equivalently $A \circ M \to M$ without (2.23), are called *scoped operations* by Wu et al. [2014] and Piróg et al. [2018].

As we explained in 1.4∗2, the name 'scoped operations' refers to the intuition that the operation s genuinely take computations as its argument, so there is a meaningful boundary between its argument and the future continuation, so its arguments are 'in the scope of s'. This is in contrast with algebraic operations $a : B \circ M \to M$ satisfying (2.23), which are equivalently $B \to M$ by 2.4∗13, so they do not genuinely take computations as input although they are usually presented in the form of $B \circ M \to M$.

2.4∗15 Example (Exception catching). Now let us come back the example of exception catching. Let $\mathscr{C}$ be a category with finite products and coproducts. Exception throwing and catching can be modelled as the theory of Σ_{Ec}-monoids in $\langle \textsc{Endo}(\mathscr{C}), \circ, \mathrm{Id} \rangle$, where the signature functor $\Sigma_{\mathrm{Ec}} : \mathscr{C} \to \mathscr{C}$ is

$$\Sigma_{\mathrm{Ec}} = ((\mathrm{Id} \times \mathrm{Id}) \circ - \circ -) + (1 \circ -)$$

equipped with the following pointed strength for all $X \in \mathscr{C}$ and $\langle Y, f \rangle : I/\mathscr{C}$:

$$\begin{aligned}(\Sigma_{\mathrm{Ec}} X) \circ Y &= \big(((\mathrm{Id} \times \mathrm{Id}) \circ X \circ X) + (1 \circ X)\big) \circ Y \\ &\cong ((\mathrm{Id} \times \mathrm{Id}) \circ X \circ X \circ Y) + (1 \circ X \circ Y) \\ &\to ((\mathrm{Id} \times \mathrm{Id}) \circ X \circ \boxed{Y} \circ X \circ Y) + (1 \circ X \circ Y) \cong \Sigma_{\mathrm{Ec}}(X \circ Y)\end{aligned}$$

where the boxed Y is inserted using $f : I \to Y$.

The intuition for the signature Σ_{Ec} is that the first operation $1 \circ M \to M$ is *throw*ing an exception as in Example 2.4∗10, and the second operation

$$\mathit{catch} : (\mathrm{Id} \times \mathrm{Id}) \circ M \circ M \cong (M \times M) \circ M \quad \longrightarrow \quad M \tag{2.28}$$

is catching. As we explained above, the trick here to avoid the undesirable equation (2.26) is that *catch* has after $M \times M$ an additional $- \circ M$ that represents an *explicit continuation* after the scoped operation *catch* [Piróg et al. 2018]: $\mathit{catch}\ (\langle p, h \rangle, k)$ is understood as handling the exception in p with h and then continuing as k. Then equation (2.23) of Σ-monoids instantiates to

$$ph : M \times M, k : M, k' : M \vdash \mu(\mathit{catch}\ (ph, k), k') = \mathit{catch}\ (ph, \mu(k, k')) : M. \tag{2.29}$$

Unlike (2.26), this equation is semantically correct: catching ph and then doing k and then k' should be the same as catching ph and then continuing as $\mu(k, k')$. The scope of *catch* is not confused.

2.4∗16. Moreover, we can add equations to the theory Σ_{Ec}-Mon to characterise the interaction of *throw* and *catch*. The theory Ec is Σ_{Ec}-Mon extended with the

following equations:

$$k : \tau \vdash \mathit{catch}(\langle \mathit{throw}, \eta \rangle, k) = k : \tau \qquad k : \tau \vdash \mathit{catch}(\langle \mathit{throw}, \mathit{throw} \rangle, k) = \mathit{throw} : \tau$$
$$k : \tau \vdash \mathit{catch}(\langle \eta, \mathit{throw} \rangle, k) = k : \tau \qquad k : \tau \vdash \mathit{catch}(\langle \eta, \eta \rangle, k) = k : \tau$$

where $\eta : I \to \tau$, $\mathit{throw} : 1 \to \tau$, and $\mathit{catch} : (\tau \times \tau) \,\square\, \tau \to \tau$. These equations can be alternatively presented with an empty context by replacing all the k's with η as in $\cdot \vdash \mathit{catch}(\langle \mathit{throw}, \eta \rangle, \eta) = \eta : \tau$, which is equivalent to the first equation above, since by (2.29), $\mathit{catch}(\langle x, y \rangle, \eta); k = \mathit{catch}(\langle x, y \rangle, k)$.

2.4∗17 Remark. By modelling scoped operations like *catch* as genuine operations rather than handlers, they enjoy the usual benefits of algebraic effects: we can impose equational axioms on them, consider the free models, combine their theories with effects, etc. But the reader may still wonder – from a purely practical programming perspective, what do we gain by modelling them as scoped operations rather than as effect handlers? Towards this question, Wu et al. [2014] argued that there is a problem of *compositionality* if we model operations like *catch* as effect handlers. Wu et al. used a concrete example of non-deterministic parsing with effect handlers, and Yang et al. [2022] clarified this problem using exception catching. Here, we explain this problem again using a small example about parallel composition as either a handler or an operation in its own right; c.f. Castellano et al. [1987].

Implementing parallel composition as a handler is similar to *interleaving concurrency* in concurrency theory: the parallel composition $P \,||_i\, Q$ of two processes P and Q is handled/reduced to the nondeterministic choice of all the ways of interleaving the actions of P and Q. For example, if $P := a.0$ is the process that performs an action a and stops, and $Q := b.0$ is the process that performs an action b and stops, then we have

$$P \,||_i\, Q = (a.b.0) + (b.a.0).$$

A problem arises if we want to lower the level of abstraction by refining the action a into two actions a_1 and a_2 (e.g. we may want to refine the action of writing a memory location into several actions when we move down the level of abstractions from an abstract memory model to a more realistic memory model). We may define this refinement as a meta-operation r on programs that replaces every action a in the program with two actions a_1 and a_2. The expected outcome of 'parallel composition of P and Q' with actions refined is

$$r(P) \,||_i\, r(Q) = (a_1.a_2.0) \,||_i\, (b.0) = (a_1.a_2.b.0) + (a_1.b.a_2.0) + (b.a_1.a_2.0),$$

but refining $P \,||_i\, Q$ above gives us

$$r(P \,||_i\, Q) = r((a.b.0) + (b.a.0)) = (a_1.a_2.b.0) + (b.a_1.a_2.0),$$

which is not the expected outcome in this scenario.

The problem here is precisely that if we treat parallel composition as a handler, then the scope of parallel composition is lost after handling. On the other hand, if we treat parallel composition as an operation in its own right, parallel composition of P and Q is *just* $P \mid\mid Q$ (with perhaps some axioms), so

$$r(P \mid\mid Q) = (a_1.a_2.0) \mid\mid b.0 = r(P) \mid\mid r(Q).$$

Therefore, if we model parallel composition as a handler $||_i$, we must carefully ensure that the refinement operation r is applied to the processes P and Q before applying $||_i$ to them, whereas if we model parallel composition a scoped operation $||$, we can apply the refinement operation r to processes built with $||$, which is a more compositional approach in terms of practical programming.

2.4∗18. There are many more examples of Σ-monoids that we cannot expand on here. Some interesting ones are lambda abstraction [Fiore et al. 1999], the algebraic operations of π-calculus [Stark 2008], and the non-algebraic operation of parallel composition [Piróg et al. 2018].

2.5 Families of Operations

2.5∗1. Given a monoidal category $\mathscr{E}$, the coslice category $\textsc{Mon}/\textsc{Eqs}(\mathscr{E})$ contains all equational systems that extend the theory of monoids with new operations/equations. However, this category is sometimes too general – we will see later that there are many constructions that only work for a certain kind of operations, such as algebraic operations or scoped operations. Therefore, we will need to consider subcategories of $\textsc{Mon}/\textsc{Eqs}(\mathscr{E})$ that contain equational systems that extend $\textsc{Mon}$ with a certain family of operations. In this section, let us have a look at some important operation families, and it turns out that there are quite some interesting things to be said about them.

2.5∗2 Definition. An *operation family* on monoids in a monoidal category $\mathscr{E}$ is a subcategory $\mathcal{F} \subseteq \textsc{Mon}/\textsc{Eqs}(\mathscr{E})$ of the coslice category of equational systems under the theory $\textsc{Mon}$ of monoids.

2.5∗3. An object $\ddot{\Sigma}$ in an operation family $\mathcal{F}$ is a pair $\langle \dot{\Sigma}, T_\Sigma : \textsc{Mon} \to \dot{\Sigma} \rangle$, but we will colloquially say *something* of $\ddot{\Sigma}$ to mean that thing of $\dot{\Sigma}$. For example, when we say $\ddot{\Sigma}$ has the free-forgetful adjunction, we mean that $\dot{\Sigma}$ has it.

2.5∗4 (Algebraic operations). The simplest example is the family $\textsc{Alg}(\mathscr{E})$ of *algebraic operations* on a monoidal category $\langle \mathscr{E}, \square, I \rangle$ with binary coproducts. The

full subcategory $\textsc{Alg}(\mathscr{E}) \subseteq \textsc{Mon}/\textsc{Eqs}(\mathscr{E})$ contains objects of the following form

$$\langle\ (A \,\square\, -)\text{-}\textsc{Mon} \upharpoonleft (\mathrm{K}_B \vdash L = R),\quad T\ \rangle \tag{2.30}$$

where $A, B \in \mathscr{E}$; the functor $A \,\square\, -$ is equipped with the pointed strength

$$\alpha_{A,X,Y} : (A \,\square\, X) \,\square\, Y \cong A \,\square\, (X \,\square\, Y);$$

T is the inclusion translation from $\textsc{Mon}$; $\mathrm{K}_B : \mathscr{E} \to \mathscr{E}$ is the constant functor mapping to B. In other words, $\textsc{Alg}(\mathscr{E})$ contains all equational systems Σ-$\textsc{Mon}$ in 2.4∗4 extended with an equation for some $\Sigma = A \,\square\, -$.

2.5∗5. In particular, the theory of exceptions (Example 2.4∗10) and state (Example 2.4∗9) are in $\textsc{Alg}(\mathscr{E})$. When $\mathscr{E} = \langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for an lκp category $\mathscr{C}$, $\textsc{Alg}(\mathscr{E})$ consists of theories of algebraic operations $A \circ M \to M$ for $A \in \textsc{Endo}_\kappa(\mathscr{C})$ on κ-accessible monads M. When $\mathscr{E}$ is $\langle \textsc{Endo}_\kappa(\textsc{Set}), *, \mathrm{Id}\rangle$, it then contains theories of applicatives F with 'applicative-algebraic' operations $A * F \to F$.

2.5∗6. When the monoidal category $\mathscr{E}$ is *right distributive* for binary coproducts, which means that the canonical morphism

$$[\iota_1 \,\square\, C, \iota_2 \,\square\, C] : (A \,\square\, C + B \,\square\, C) \to (A + B) \,\square\, C$$

is an isomorphism, the category $\textsc{Alg}(\mathscr{E})$ has binary coproducts as well: binary coproducts in $\textsc{Alg}(\mathscr{E})$ are equivalently pushouts in $\textsc{Eqs}(\mathscr{E})$, and by Lemma 2.2.3∗5, such a pushout still has a constant context $\mathrm{K}_B + \mathrm{K}_{B'} = \mathrm{K}_{B+B'}$ and a signature $(A \,\square\, -) + (A' \,\square\, -) \cong (A + A') \,\square\, -$, so it is still in $\textsc{Alg}(\mathscr{E})$.

2.5∗7. Many of the monoidal categories in Section 2.1 are right distributive:

- $\langle \textsc{Endo}(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for an small-complete small $\mathscr{C}$;
- $\langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for an lκp $\mathscr{C}$;
- $\langle \mathscr{C}, \times, 1\rangle$ for a cocomplete cartesian closed $\mathscr{C}$;
- $\langle \textsc{Endo}_\kappa(\textsc{Set}), *, \mathrm{Id}\rangle$ underlying applicative functors;
- $\langle \textsc{Endo}_{s\kappa}(\mathscr{C}), \circ_s, \mathrm{Id}_s\rangle$ for an lκp as a cartesian closed category $\mathscr{C}$;
- $\langle \textsc{Endo}_f(\textsc{Set})^{\mathscr{G}}, *, I\rangle$ for a small strict monoidal category $\mathscr{G}$.

Moreover, for these choices of $\mathscr{E}$, every object of $\textsc{Alg}(\mathscr{E})$ has the free-forgetful adjunction using the freeness conditions in Section 2.2.1.

2.5∗8. The restriction in (2.30) that the context of the equation must be a constant functor K_B deserves some explanation. Let $\Sigma : \mathscr{C} \to \mathscr{C}$ be an endofunctor on a category $\mathscr{C}$. We call a functorial equation $\mathrm{K}_B \vdash L = R$ over the signature Σ with a constant functor as its context a *constant equation*.

Assume that $\mathscr{C}$ is locally small and small-complete, and that the functor Σ has the free-forgetful adjunction $\mathrm{F}_\Sigma \dashv \mathrm{U}_\Sigma$. By Corollary 2.2.2∗7, the equation $\mathrm{K}_B \vdash L = R$ is equivalently a pair of natural transformations $\mathrm{K}_B \rightrightarrows \mathrm{F}_\Sigma\mathrm{U}_\Sigma$. Moreover, when $\mathscr{C}$ has an initial object 0, such a natural transformation is uniquely determined by its component at 0:

$$\begin{array}{ccc} \mathrm{K}_B 0 = B & \xrightarrow{id_B} & \mathrm{K}_B X = B \\ {\scriptstyle L_0}\downarrow & & \downarrow{\scriptstyle L_X} \\ \mathrm{U}_\Sigma\mathrm{F}_\Sigma 0 & \xrightarrow[\mathrm{U}_\Sigma\mathrm{F}_\Sigma f]{} & \mathrm{U}_\Sigma\mathrm{F}_\Sigma X \end{array}$$

Therefore constant equations over the signature $\Sigma : \mathscr{C} \to \mathscr{C}$ are in bijection with pairs of $\mathscr{C}$-morphisms $L, R : B \to \mathrm{U}_\Sigma\mathrm{F}_\Sigma 0$, whose codomain is precisely (the carrier of) the initial algebra $\mu\Sigma$ of Σ.

2.5∗9. More specially, assume that a monoidal category $\mathscr{E}$ satisfies the conditions on $\mathscr{C}$ in the last paragraph, and that $\Box : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserves colimits of α-chains for some limit ordinal α. In this case, let Σ be the signature functor of the equational system $(A \Box -)$-Mon, i.e. $\Sigma = I + - \Box - + A \Box -$. Then both Σ and the equational system $(A \Box -)$-Mon have initial algebras, whose carriers we denote by X and Y respectively. For every algebra $\langle Z, \alpha \rangle$ of $(A \Box -)$-Mon, it satisfies a constant equation $\mathrm{K}_B \vdash L = R$ if and only if the following diagram commutes:

$$B \underset{r}{\overset{l}{\rightrightarrows}} X \xrightarrow{!_\Sigma} Z$$

where the morphism $!_\Sigma : X \to Z$ is the unique Σ-homomorphism from X to Z. Since the algebras of $(A \Box -)$-Mon is a full subcategory of the algebras of Σ, the morphism $!_\Sigma$ factors via $!_{(A\Box-)\text{-Mon}} : Y \to Z$, the unique homomorphism from the initial algebra of $(A \Box -)$-Mon:

$$B \underset{r}{\overset{l}{\rightrightarrows}} X \xrightarrow{!_\Sigma} Y \xrightarrow{!_{(A\Box-)\text{-Mon}}} Z$$

Therefore, equations $\mathrm{K}_B \vdash L = R$ can be given as a pair of morphisms $B \rightrightarrows Y$ without loss of expressivity. In particular, if $\mathscr{E}$ is $\langle \mathrm{End}_f(\mathrm{Set}), \circ, \mathrm{Id} \rangle$, Y is precisely the free monad A^* over A, a pair of morphisms $B \rightrightarrows A^*$ for $A, B \in \mathrm{End}_f(\mathrm{Set})$ is indeed the traditional way of presenting a finitary algebraic theory. In fact, we can show that the category $\mathrm{Alg}(\mathscr{E})$ is equivalent to the category (as defined by e.g. Fiore and Mahmoud [2014]) of presentations finitary algebraic theories and translations. Therefore, although constant equations seem very restrictive, they are still useful enough when the monoidal category $\mathscr{E}$ itself is informative.

2.5∗10 Theorem. *Let $\mathscr{E}$ be a monoidal category with binary coproducts such that every object in* $\mathrm{Alg}(\mathscr{E})$ *has the free-forgetful adjunction. There is an equivalence* $\mathrm{Alg}(\mathscr{E}) \cong$

Mon($\mathscr{E}$) *between* Alg($\mathscr{E}$) *and the category* Mon($\mathscr{E}$) *of monoids in* $\mathscr{E}$.

Proof. In sketch, every $\ddot{\Sigma} \in \text{Alg}(\mathscr{E})$ is mapped to its initial algebra treated as a monoid, forgetting the operation. On the other hand, every monoid M is mapped to the theory of *M-actions on monoids*.

In more detail, the direction $\text{Alg}(\mathscr{E}) \to \text{Mon}(\mathscr{E})$ of the equivalence sends every object $\ddot{\Sigma} = \langle \dot{\Sigma}, T_\Sigma \rangle$ in $\text{Alg}(\mathscr{E})$ to the initial algebra $\langle \mu\dot{\Sigma}, \alpha^\Sigma \rangle$ regarded as a monoid $T_\Sigma\langle \mu\dot{\Sigma}, \alpha^\Sigma \rangle$. On morphisms, every translation $T : \ddot{\Sigma} \to \ddot{\Psi} \in \text{Alg}(\mathscr{E})$ induces a unique $\ddot{\Sigma}$-homomorphism out of the initial algebra:

$$h : \langle \mu\dot{\Sigma}, \alpha^\Sigma \rangle \to T\langle \mu\dot{\Psi}, \alpha^\Psi \rangle.$$

Then the arrow mapping is $T \mapsto T_\Sigma h$ where

$$T_\Sigma h : T_\Sigma\langle \mu\dot{\Sigma}, \alpha^\Sigma \rangle \to T_\Sigma(T\langle \mu\dot{\Psi}, \alpha^\Psi \rangle) = T_\Psi\langle \mu\dot{\Psi}, \alpha^\Psi \rangle.$$

The equality $T_\Sigma \circ T = T_\Psi$ is by the definition of morphisms in $\text{Alg}(\mathscr{E})$.

For the other direction, every monoid $\dot{M} = \langle M, \mu^M, \eta^M \rangle$ in $\mathscr{E}$ is sent to the theory $\dot{M}$-Act of *$\dot{M}$-actions on monoids*, which is the theory of $(M \,\square\, -)$-Mon extended with the following two equations (expressed as an extension of the internal language of $\mathscr{E}$ as in 2.3.3*14):

$$\vdash op(\eta^M, \eta^\tau) = \eta^\tau : \tau$$
$$x : M, y : M \vdash op(\mu^M(x, y), \eta^\tau) = op(x, op(y, \eta^\tau)) : \tau$$

saying that $op : M \,\square\, \tau \to \tau$ is a monoid action on τ. Every monoid morphism $f : \dot{M} \to \dot{N}$ is mapped to the translation $\dot{M}$-Act $\to \dot{N}$-Act sending $\dot{N}$-actions

$$\langle A \in \mathscr{E}, \alpha : (N \,\square\, A) + \Sigma_{\text{Mon}} A \to A \rangle$$

to $\dot{M}$-actions $\langle A, [\alpha \cdot \iota_1 \cdot (f \,\square\, A), \alpha \cdot \iota_2] : (M \,\square\, A) + \Sigma_{\text{Mon}} A \to A \rangle$.

It remains to show that the mappings above are a pair of equivalence. Starting from a monoid $\dot{M}$, it can be shown that the category ($\dot{M}$-Act)-Alg is equivalent to the coslice category $\dot{M}/\text{Mon}(\mathscr{E})$ (see e.g. Fiore and Saville [2017, Proposition 5.5]). Thus the initial algebra of $\dot{M}$-Act is $\dot{M}$ as required.

Starting from a theory $\langle \dot{\Sigma}, T_\Sigma \rangle \in \text{Alg}(\mathscr{E})$ where

$$\dot{\Sigma} = (S \,\square\, -)\text{-Mon} \pitchfork (\text{K}_B \vdash L = R),$$

it is mapped to the monoid $T_\Sigma\langle \mu\dot{\Sigma}, \alpha^\Sigma \rangle$, which is then mapped back to the theory $\mu\dot{\Sigma}$-Act with the inclusion translation. We need to construct an isomorphism translation $T : \dot{\Sigma} \to \mu\dot{\Sigma}$-Act that preserves monoid operations. Given a monoid A with $\alpha : \mu\dot{\Sigma} \,\square\, A \to A$ satisfying the laws of $\mu\dot{\Sigma}$-Act, T maps it to the $\dot{\Sigma}$-algebra

on A with the following operation:

$$S \,\square\, A \xrightarrow{S\square\eta^{\mu\dot\Sigma}\square A} S \,\square\, \mu\dot\Sigma \,\square\, A \xrightarrow{\alpha^{\mu\dot\Sigma}\square A} \mu\dot\Sigma \,\square\, A \xrightarrow{\alpha} A$$

where $\alpha^{\mu\dot\Sigma} : S \,\square\, \mu\dot\Sigma \to \mu\dot\Sigma$ is the structure map of the initial algebra.

For the inverse of T, every tuple $\langle A, \beta : S \,\square\, A \to A\rangle \in \dot\Sigma\text{-}\textsc{Alg}$ is mapped to the following $\mu\dot\Sigma$-Act-algebra on A:

$$\mu\dot\Sigma \,\square\, A \xrightarrow{(\!|\beta|\!)\square A} A \,\square\, A \xrightarrow{\mu^A} A$$

where $(\!|\beta|\!)$ is the unique homomorphism from the initial $\dot\Sigma$-algebra $\mu\dot\Sigma$ to the $\dot\Sigma$-algebra $\langle A, \beta\rangle$. This completes the proof. □

2.5∗11. Instantiating $\mathscr{E}$ with $\langle \textsc{End}o_f(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for an lfp $\mathscr{C}$, we obtain an equivalence between finitary monads over $\mathscr{C}$ and theories of algebraic operations on finitary monads. This is reminiscent of the classical *theory-monad correspondence* between finitary monads and (presentations of) first-order algebraic theories. What is new is that Theorem 2.5∗14 is applicable to other monoidal categories, such as those in Section 2.1, giving us equivalences of cartesian monoids/applicative functors/graded monads and the corresponding categories of theories of algebraic operations.

2.5∗12. Another interesting property of $\textsc{Alg}(\mathscr{E})$ is the following saying that (almost) all equational systems of operations on monoids can be turned into one in $\textsc{Alg}(\mathscr{E})$ by a coreflection, and the coreflection preserves initial algebras, i.e. the abstract syntax of terms of operations. Hence in principle, theories of algebraic operations alone are sufficient for the purpose of modelling syntax.

2.5∗13 Lemma. Let $\mathscr{E}$ be a monoidal category with binary coproducts. For every $\ddot\Psi \in \textsc{Alg}(\mathscr{E})$ and $\ddot\Sigma \in \textsc{Mon/Eqs}(\mathscr{E})$ such that both of them have initial algebras, $\textsc{Mon/Eqs}(\ddot\Psi, \ddot\Sigma)$ is in natural bijection to monoid morphisms $\mu\ddot\Psi \to \mu\ddot\Sigma$ between the initial algebras of $\ddot\Psi$ and $\ddot\Sigma$ viewed as monoids.

Proof sketch. For one direction of the bijection ϕ, every translation $T : \ddot\Psi \to \ddot\Sigma$ sends the initial algebra $\mu\ddot\Sigma$ of $\ddot\Sigma$ to a $\ddot\Psi$-algebra carried by $\mu\ddot\Sigma$. Then by the initiality of $\mu\ddot\Psi$, there is a $\ddot\Psi$-homomorphism, which is also a monoid morphism, $u : \mu\ddot\Psi \to \mu\ddot\Sigma$. We set $\phi(T) = u$. For the backward direction of the bijection ϕ, given a monoid morphism $h : \mu\ddot\Psi \to \mu\ddot\Sigma$, we define a translation $T : \ddot\Psi \to \ddot\Sigma$, i.e. a functor $T : \ddot\Sigma\text{-}\textsc{Alg} \to \ddot\Psi\text{-}\textsc{Alg}$ as follows. Recall that $\ddot\Psi \in \textsc{Alg}(\mathscr{E})$ must be of the form $\dot\Psi = (G \,\square\, -)\text{-}\textsc{Mon} \upharpoonleft (\mathrm{K}_B \vdash L = R)$, for some $G \in \mathscr{E}$. The functor T maps every $\ddot\Sigma$-algebra $\langle A, \alpha : \Sigma A \to A\rangle$ to the $\dot\Psi$-algebra carried by A with

$$G \,\square\, A \xrightarrow{G\square\eta^{\mu\ddot\Psi}} G \,\square\, \mu\ddot\Psi \,\square\, A \xrightarrow{\alpha^{\mu\ddot\Psi}} \mu\ddot\Psi \,\square\, A \xrightarrow{h} \mu\ddot\Sigma \,\square\, A \xrightarrow{(\!|\alpha|\!)} A \,\square\, A \xrightarrow{\mu^A} A$$

where $\alpha^{\mu\ddot{\Psi}} : G \,\square\, \mu\ddot{\Psi} \to \mu\ddot{\Psi}$ is the structure map of the initial $\ddot{\Psi}$-algebra, $(\!|\alpha|\!) : \mu\ddot{\Sigma} \to A$ is the unique $\ddot{\Sigma}$-homomorphism from the initial algebra $\mu\ddot{\Sigma}$ to $\langle A, \alpha\rangle$. It can be checked that ϕ is a natural bijection. □

2.5∗14 Theorem. *Let $\mathscr{E}$ be a monoidal category with binary coproducts such that every object of* Alg($\mathscr{E}$) *has the free-forgetful adjunction. The category* Alg($\mathscr{E}$) *is a coreflective subcategory of* Mon/Eqs$_f$($\mathscr{E}$)*, where* Eqs$_f$($\mathscr{C}$) $\subseteq$ Eqs($\mathscr{C}$) *is the full subcategory containing equational systems with the free-forgetful adjunction*:

$$\textsc{Alg}(\mathscr{E}) \;\underset{\lfloor - \rfloor}{\overset{\top}{\leftrightarrows}}\; \textsc{Mon}/\textsc{Eqs}_f(\mathscr{E}).$$

Moreover, the coreflector $\lfloor - \rfloor$ preserves initial algebras: *for every $\langle \dot{\Sigma}, T\rangle$ in the category* Mon/Eqs$_f$($\mathscr{E}$)*, the initial $\dot{\Sigma}$-algebra* (*viewed as a monoid via T*) *is isomorphic to the initial algebra of $\lfloor \langle \dot{\Sigma}, T\rangle \rfloor$ also viewed as a monoid.*

Proof sketch. Every theory $\langle \dot{\Sigma}, T_\Sigma\rangle \in \textsc{Mon}/\textsc{Eqs}_f(\mathscr{E})$ has an initial algebra $\mu\dot{\Sigma} \in \mathscr{E}$ by assumption, and $\mu\dot{\Sigma}$ carries a monoid structure by $T_\Sigma : \textsc{Mon} \to \dot{\Sigma}$. We define the coreflector $\lfloor - \rfloor$ to map every $\langle \dot{\Sigma}, T_\Sigma\rangle$ to $\mu\dot{\Sigma}$-Act $\in$ Alg($\mathscr{E}$) as in the proof of Theorem 2.5∗10. For every theory $\langle \dot{\Psi}, T_\Psi\rangle \in$ Alg($\mathscr{E}$), by Lemma 2.5∗13, each translation in the hom-set Mon/Eqs$_f(\langle \dot{\Psi}, T_\Psi\rangle, \langle \dot{\Sigma}, T_\Sigma\rangle)$ is equivalently a monoid morphism $\mu\dot{\Psi} \to \mu\dot{\Sigma}$, which is also equivalently a translation in Alg$(\langle \dot{\Psi}, T_\Psi\rangle, \lfloor \langle \dot{\Sigma}, T_\Sigma\rangle \rfloor)$ by Theorem 2.5∗10.

The coreflector maps each $\langle \dot{\Sigma}, T\rangle \in$ Mon/Eqs$_f$($\mathscr{E}$) to the theory $\mu\dot{\Sigma}$-Act. It can be shown that the category of algebras of $\mu\dot{\Sigma}$-Act is equivalent to the coslice category $\mu\dot{\Sigma}/$Mon($\mathscr{E}$) of monoids under $T\langle \mu\dot{\Sigma}, \alpha^{\Sigma}\rangle$, so the initial algebra of $\mu\dot{\Sigma}$-Act is still the same monoid $\mu\dot{\Sigma}$. □

2.5∗15. Although Alg($\mathscr{E}$) is sufficient for modelling syntax, it is *not* enough when we also consider models. The counit of the coreflection gives us a translation $\lfloor \langle \dot{\Sigma}, T\rangle \rfloor \to \langle \dot{\Sigma}, T\rangle$, i.e. a functor $\dot{\Sigma}$-Alg $\to \lfloor \langle \dot{\Sigma}, T\rangle \rfloor$-Alg, but these two categories of models are in general not equivalent.

2.5∗16 (Scoped operations). Our next example of operation families is the family Scp($\mathscr{E}$) of *scoped (and algebraic) operations*, such as exception catching (Example 2.4∗15). Let $\mathscr{E}$ be a monoidal category with right distributive binary coproducts (2.5∗6). The family Scp($\mathscr{E}$) is the full subcategory of Mon/Eqs($\mathscr{E}$) containing objects

$$\langle\; ((A \,\square\, - \,\square\, -) + (B \,\square\, -))\text{-Mon}^{\,\lceil}(\mathrm{K}_C \vdash L = R), \quad T \;\rangle \tag{2.31}$$

where $A, B, C \in \mathscr{E}$ and T is the inclusion translation. Letting $\Sigma := (A \,\square\, - \,\square\, -) +$

$(B \,\square\, -)$, the pointed strength $\theta_{X,\langle Y,f\rangle}$ for Σ is the following composite:

$$\begin{aligned}(\Sigma X) \,\square\, Y &= (A \,\square\, X \,\square\, X + B \,\square\, X) \,\square\, Y \\ &\cong (A \,\square\, X \,\square\, X \,\square\, Y) + (B \,\square\, X \,\square\, Y) \\ &\to (A \,\square\, X \,\square\, \boxed{Y} \,\square\, X \,\square\, Y) + (B \,\square\, X \,\square\, Y) \\ &\cong \Sigma(X \,\square\, Y)\end{aligned}$$

where the boxed Y is inserted using $f : I \to Y$.

In the example of exception throwing and catching in Example 2.4∗15, the monoidal category $\mathscr{E}$ is $\langle \textsc{Endo}(\mathscr{C}), \circ, \mathrm{Id}\rangle$, and A is $\mathrm{Id} \times \mathrm{Id}$, encoding the scoped operation *catch* that takes in two operands, and B is 1, encoding the algebraic operation *throw* that takes in no operands.

2.5∗17. When every object of $\textsc{Scp}(\mathscr{E})$ has the free-forgetful adjunction, for example when $\mathscr{E}$ is cocomplete and $\square : \mathscr{E} \times \mathscr{E} \to \mathscr{E}$ preserves colimits of α-chains for some limit ordinal α, a corollary of Theorem 2.5∗14 is that the initial-algebra preserving coreflection there restricts to

$$\textsc{Alg}(\mathscr{E}) \quad \underset{\top}{\overset{\longleftarrow}{\hookrightarrow}} \quad \textsc{Scp}(\mathscr{E}).$$

Thus the abstract syntax of programs with scoped operations can be alternatively expressed with only algebraic ones, but as argued by Piróg et al. [2018] and Yang et al. [2022], the models of scoped operations are different from those of the coreflected algebraic operations.

2.5∗18. The operation family $\textsc{Scp}(\mathscr{E})$ should be more accurately called the family of *scoped and algebraic operations with constant equations*. We can certainly relax the restriction of constant equations to obtain bigger operation families. For example, we may have a family $\textsc{Scp}_1(\mathscr{E})$ that is similar to (2.31) but permits *first-order equations*, meaning that the functorial context can be either a constant functor K_C or a functor $C \,\square\, -$ for some $C \in \mathscr{E}$.

For computational effects in practice, it seems constant equations in $\textsc{Endo}_\kappa(\mathscr{C})$ are enough for algebraic operations, evidenced by the examples in [Plotkin and Power 2002], whereas scoped operations sometimes need first-order equations. For example, a reasonable equation for exception catching $\mathit{catch} : M \times M \to M$ is that it is associative with respect to $\times$:

$$\begin{array}{ccc} M \times M \times M & \xrightarrow{\mathit{catch} \times M} & M \times M \\ {\scriptstyle M \times \mathit{catch}}\downarrow & & \downarrow{\scriptstyle \mathit{catch}} \\ M & \xrightarrow[\mathit{catch}]{} & M \end{array}$$

As an equation in the monoidal category $\langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$, this equation is first-order, since the context is $(\mathrm{Id}_{\mathscr{C}} \times \mathrm{Id}_{\mathscr{C}} \times \mathrm{Id}_{\mathscr{C}}) \circ - : \textsc{Endo}(\mathscr{C}) \to \textsc{Endo}(\mathscr{C})$.

2.5∗19. We did *not* require operation families $\mathcal{F} \subseteq \textsc{Mon}/\textsc{Eqs}(\mathscr{E})$ to be full, so we may also restrict the translations. For example, we can consider only translations that map operations to *operations* (rather than terms in general). Such translations are sometimes called *transliterations* [Arkor 2022]. In particular, we define $\textsc{Scp}_l(\mathscr{E}) \subseteq \textsc{Scp}(\mathscr{E})$ to be the subcategory containing translations

$$T : ((A \,\square - \square -) + (B \,\square -))\text{-}\textsc{Mon} \to ((A' \,\square - \square -) + (B' \,\square -))\text{-}\textsc{Mon}$$

that sends $\langle M, s : A' \square M \square M, a : B' \square M \to M \rangle$ to $\langle M, s \cdot (f \square M \square M), a \cdot (g \square M) \rangle$ for some morphisms $f : A \to A'$ and $g : B \to B'$ in $\mathscr{E}$.

For example, let $A = \mathrm{Id} \times \mathrm{Id} \times \mathrm{Id}$ and $A' = \mathrm{Id} \times \mathrm{Id}$, corresponding to a ternary operation t and a binary operation b respectively. Then the transliteration given by $f\langle x, y, z\rangle = \langle x, y\rangle$ translates the operation $t(x, y, z)$ to $b(x, y)$, but a transliteration cannot translate $t(x, y, z)$ to $b(b(x, y), z)$ that uses the target operation more than once, nor can it use the monoid structure μ and η in the translation.

2.5∗20 (Variable-binding operations). Algebraic theories of operations with *variable-bindings* are called *second-order algebraic theories* [Fiore and Hur 2010; Fiore and Mahmoud 2010, 2014; Fiore and Szamozvancev 2022; Fiore et al. 1999], and they can be formulated as an operation family as follows. For simplicity, we work specially in the monoidal category $\langle \textsc{Set}^{\textsc{Fin}}, \bullet, V \rangle$ in 2.1.2∗2, but it is possible to replace $\textsc{Set}^{\textsc{Fin}}$ with $\textsc{Endo}_\kappa(\textsc{Set})$ for infinitary syntax or with $\textsc{Set}^{\textsc{Ctx}}$ for simply typed syntax given a category $\textsc{Ctx}$ of contexts and renamings.

2.5∗21. A *binding signature* $\langle O, a\rangle$ consists of a set O of operations and an arity assignment $a : O \to \mathbb{N}^*$ of a sequence of natural numbers to each operation. Each $o \in O$ with $a(o) = \langle n_i \rangle_{1 \leqslant i \leqslant k}$ stands for an operation taking k arguments, each binding n_i variables in their arguments:

$$o((x_{1,1}x_{1,2}\cdots x_{1,n_1}).\, e_1,\ \cdots,\ (x_{k,1}x_{k,2}\cdots, x_{k,n_k}).\, e_k)$$

For example, the binding signature for λ-calculus has two operations $\{app, abs\}$: function application $a(app) = \langle 0, 0\rangle$ has two arguments binding no variables; λ-abstraction $a(abs) = \langle 1\rangle$ has one argument that binds one variable.

A binding signature $\langle O, a\rangle$ determines an endofunctor $[\![O, a]\!]$ on $\textsc{Set}^{\textsc{Fin}}$:

$$[\![O, a]\!] = \textstyle\coprod_{o \in O,\ a(o) = \langle n_i \rangle_{1 \leqslant i \leqslant k}} \prod_{1 \leqslant i \leqslant k} (-)^{V^{n_i}}$$

where $(-)^{V^{n_i}}$ is the exponential by n_i-fold product of the monoidal unit V. This functor has a pointed strength $\theta_{X, \langle Y, \eta^Y \rangle}$:

$$(\textstyle\coprod_o \prod_i X^{V^{n_i}}) \bullet Y \cong \coprod_o \prod_i (X^{V^{n_i}} \bullet Y) \xrightarrow{\coprod_o \prod_i t_{o,i}} \coprod_o \prod_i (X \bullet Y)^{V^{n_i}}$$

where $t_{o,i}$ is the adjoint transpose of

$$(X^{V^{n_i}} \bullet Y) \times V^{n_i} \xrightarrow{id \times \eta^Y} (X^{V^{n_i}} \bullet Y) \times (V^{n_i} \bullet Y) \cong (X^{V^{n_i}} \times V^{n_i}) \bullet Y \to X \bullet Y.$$

2.5∗22. The operation family $\textsc{Var}(\textsc{Set}^{\textsc{Fin}}) \subseteq \textsc{Mon}/\textsc{Eqs}(\textsc{Set}^{\textsc{Fin}})$ then contains all objects of the following form:

$$\langle [\![O, a]\!]\text{-}\textsc{Mon} \upharpoonleft ([\![P, b]\!] \vdash L = R),\ T \rangle$$

where $\langle O, a \rangle$ and $\langle P, b \rangle$ are two binding signatures and T is still the inclusion translation. The category $\textsc{Var}(\textsc{Set}^{\textsc{Fin}})$ is closed under coproducts by Lemma 2.2.3∗5 and every object of it has the free-forgetful adjunction because $[\![O, a]\!]$ and $[\![P, b]\!]$ are finitary, which is a consequence of $(-)^V$ being a left adjoint to the right Kan extension $\textsc{Ran}_{V+1}$, so $(-)^V$ preserves all colimits.

2.5∗23. Again, the coreflector in Theorem 2.5∗14 allows us to turn every theory in $\textsc{Var}$ into one with only algebraic operations but isomorphic initial algebras. For example, under the coreflection, the theory Λ of untyped λ-calculus is turned into a theory $\lfloor \Lambda \rfloor$ which has an ordinary n-ary operation t for *every* λ-term t with n free variables, together with suitable equations. The equational systems Λ and $\lfloor \Lambda \rfloor$ have isomorphic initial algebras (as monoids).

* * *

2.5∗24. It is a good time to reflect what we have achieved in this chapter.

- We have seen how to present theories of monoids with operations using equational systems and monoidal algebraic theories (Section 2.3);
- We can construct (relative) free algebras on cocomplete categories: Theorem 2.2.1∗12, Theorem 2.2.1∗19, Theorem 2.2.2∗14;
- We can combine such theories modularly using colimits (Section 2.2.3);
- Theories of monoids with operations can be classified into operation families (Section 2.5), with the family of algebraic operations playing a special role: Theorem 2.5∗10 and 2.5∗14.

These results achieve syntactic modularity for computational effects. In the next chapter, we will develop a framework of *modular models*, which will allow us to combine models of existing theories into a model of a combined theory, thus achieving modularity for both syntax and semantics.

Chapter 3

Modular Constructions of Algebraic Structures

3∗1. In many frameworks of algebraic theories, we can combine smaller theories into bigger ones by taking colimits. This gives us a modular way to design programming languages: language features are defined individually as algebraic theories, which are then combined to form algebraic theories of full-fledged languages. Programming language theory is not only about syntax/theories of languages though. What is usually more interesting is the implementations/models, and it turns out that modularly combining models is significantly harder than modularly combining theories. In this chapter, we propose a formal theory of modularity in algebraic structures and show some concrete examples. The material in this chapter is a generalisation of that appeared in the author's earlier papers [Yang and Wu 2021, 2023].

3∗2. Let us begin with some high-level description of the concepts that we will define in this chapter. Let $\mathscr{T}$ be a category of some notion of algebraic theories, together with a functor $(-)$-Alg : $\mathscr{T}^{\mathrm{op}} \to$ CAT that associates a category of models to every theory in $\mathscr{T}$, such that $\mathscr{T}$ has finite coproducts and each category of models Γ-Alg has an initial object $\mu\Gamma$. For example, $\mathscr{T}$ may be the category Alg($\mathscr{E}$) or Scp($\mathscr{E}$) of theories of monoids with algebraic or scoped operations that we saw in Section 2.5 or the category of (ordinary) algebraic theories.

The motivation for a *modular model* of $\Gamma \in \mathscr{T}$ is that we treat $\Gamma \in \mathscr{T}$ as a *language feature* that can be added into existing programming languages, rather than a complete language on its own. A modular model for Γ is then going to be a family of functors $M_\Sigma : \Sigma$-Alg $\to (\Sigma + \Gamma)$-Alg, (oplax-) natural in $\Sigma \in \mathscr{T}$. In this way, for an arbitrary language $\Sigma \in \mathscr{T}$ and a model $A \in \Sigma$-Alg of it, we can add the language feature Γ to the language Σ, giving us the combined language $\Sigma + \Gamma$, and a new model $M_\Sigma A \in (\Sigma + \Gamma)$-Alg.

3∗3. In the situation above, we may also want to relate the old model $A \in \Sigma$-Alg and the new model $M_\Sigma A \in (\Sigma + \Gamma)$-Alg. For this, we will define an *updater* u for the modular model M to be a family of Σ-homomorphisms $u_{\Sigma,A} : A \to M_\Sigma A$, (oplax-) natural in Σ and A. The name 'updater' comes from the informal intuition

that a model $A \in \Sigma\text{-Alg}$ is like a compiler for the language Σ; an element a fo A is like a compiled binary file; the morphism $u_{\Sigma,A}$ transforms every compiled binary a for the old compiler A to a compiled binary for the new compiler $M_\Sigma A$.

3∗4. Moreover, we notice that coproducts $\Sigma + \Gamma$ are just one particular way to combine algebraic theories, and we could also consider other ways of combining theories, such as the *commutative combination* $\Sigma \otimes \Gamma$, also known as the *tensor* [Hyland et al. 2006], which is the theory $\Sigma + \Gamma$ with additionally equations stating that operations from Σ and operations from Γ are commutative. Motivated by this, we will generalise from $(- + \Gamma) : \mathscr{T} \to \mathscr{T}$ in the definition of modular models to an arbitrary functor $\mathscr{T} \to \mathscr{T}$, better still, a functor $T : \mathscr{T} \to \mathscr{T}'$ where the domain $\mathscr{T}$ and codomain $\mathscr{T}'$ can be different ($\mathscr{T}'$ is also equipped with a functor $(-)\text{-Alg} : \mathscr{T}'^{\mathrm{op}} \to \mathrm{CAT}$). We will then call an (oplax-) natural family of functors $M_\Sigma : \Sigma\text{-Alg} \to (T\Sigma)\text{-Alg}$ a *model transformer* for T.

Model transformers will be shown to be equivalent to *liftings of functors* along fibrations, i.e. functors M making the diagram commute strictly,

$$\begin{array}{ccc} \mathscr{A} & \xrightarrow{M} & \mathscr{A}' \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle p'} \\ \mathscr{T} & \xrightarrow[T]{} & \mathscr{T}' \end{array}$$

where p and p' are the fibrations corresponding to $(-)\text{-Alg} : \mathscr{T}^{\mathrm{op}} \to \mathrm{CAT}$ and $(-)\text{-Alg} : \mathscr{T}'^{\mathrm{op}} \to \mathrm{CAT}$ respectively via the Grothendieck construction. The main technical contribution of this chapter will then be a toolbox of constructions (3.3∗1) of such model transformers.

3∗5. The structure of this chapter is as follows:

- In Section 3.1, we define modular models of monoids with operations, and establish the correspondence between two formulations based on *indexed categories* and *fibrations* respectively.
- In Section 3.2, motivated by more scenarios of algebraic structures, we generalise modular models to *model transformers*, which are just a 'name with an attitude' [nLab 2024] for liftings of functors along fibrations.
- In Section 3.3, we show a handful of general constructions and concrete examples of modular models and model transformers.

3.1 Modular Models of Monoids

3.1∗1 Notation. In this section, when we say 'a category $\mathscr{C}$', by default we mean a *large* category unless otherwise specified, so we would have $\mathscr{C} \in \mathrm{CAT}$, where

CAT is the category of large categories. We fix a monoidal category $\mathscr{E}$ and an operation family $\mathcal{F} \subseteq \textsc{Mon}/\textsc{Eqs}(\mathscr{E})$ such that $\mathcal{F}$ is closed under finite coproducts in $\textsc{Mon}/\textsc{Eqs}(\mathscr{E})$. Every object $\ddot{\Sigma} \in \mathcal{F}$ is a pair $\langle \dot{\Sigma} \in \textsc{Eqs}(\mathscr{E}), T : \textsc{Mon} \to \dot{\Sigma} \rangle$. We will write $\ddot{\Sigma}$-$\textsc{Alg}$ to mean the category $\dot{\Sigma}$-$\textsc{Alg}$ of algebras for $\dot{\Sigma}$.

3.1∗2. In this section, we will make precise the idea of *modular models*, as motivated in 3∗2, in two equivalent formulations, one based on *indexed categories* (Definition 3.1∗5), and another based on *fibrations* (Theorem 3.1∗19). The former is more explicit while the latter is more convenient.

3.1∗3. Recall that a morphism $T : \dot{\Sigma} \to \dot{\Psi}$ in $\textsc{Eqs}(\mathscr{E})$ is a functor $\dot{\Psi}$-$\textsc{Alg} \to \dot{\Sigma}$-$\textsc{Alg}$ such that $\mathrm{U}_{\dot{\Sigma}} \circ T = \mathrm{U}_{\dot{\Psi}} : \dot{\Psi}\text{-}\textsc{Alg} \to \mathscr{E}$, so we can treat $(-)$-$\textsc{Alg}$ as a functor $\textsc{Eqs}(\mathscr{E})^{\mathrm{op}} \to \mathrm{CAT}$. Similarly, we have a functor $(-)$-$\textsc{Alg} : \mathcal{F}^{\mathrm{op}} \to \mathrm{CAT}$.

3.1∗4 Definition (Johnson and Yau [2020]). For a category $\mathscr{C}$ and two functors $F, G : \mathscr{C} \to \mathrm{CAT}$, a *lax transformation* $\alpha : F \to G$ consists of a family of functors $\alpha_X : FX \to GX$ for all $X \in \mathscr{C}$ and a family of natural transformations $\alpha_f : Gf \circ \alpha_X \to \alpha_Y \circ Ff$ for all morphisms $f : X \to Y$ in $\mathscr{C}$:

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ {\scriptstyle \alpha_X}\downarrow & \overset{\alpha_f}{\nearrow} & \downarrow{\scriptstyle \alpha_Y} \\ GX & \xrightarrow[Gf]{} & GY \end{array}$$

Additionally, α must satisfy that $\alpha_{id_X} = id : \alpha_X \to \alpha_X$ for all $X \in \mathscr{C}$, and for all $f : X \to Y$ and $g : Y \to Z$ in $\mathscr{C}$, $\alpha_{g \cdot f}$ is exactly the pasting of α_f and α_g:

$$\begin{array}{ccccc} FX & \xrightarrow{Ff} & FY & \xrightarrow{Gg} & FZ \\ {\scriptstyle \alpha_X}\downarrow & \overset{\alpha_f}{\nearrow} & {\scriptstyle \alpha_Y}\downarrow & \overset{\alpha_g}{\nearrow} & \downarrow{\scriptstyle a_Z} \\ GX & \xrightarrow[Gf]{} & GY & \xrightarrow[Gg]{} & GZ \end{array} \quad = \quad \begin{array}{ccccc} FX & \xrightarrow{Ff} & FY & \xrightarrow{Gg} & FZ \\ {\scriptstyle \alpha_X}\downarrow & & \overset{\alpha_{g \cdot f}}{\nearrow} & & \downarrow{\scriptstyle a_Z} \\ GX & \xrightarrow[Gf]{} & GY & \xrightarrow[Gg]{} & GZ \end{array}$$

An *oplax* transformation from F to G is similar to a lax transformation except that the direction of the 2-cells α_f becomes $\alpha_Y \circ Ff \to Gf \circ \alpha_X$. An (op)lax transformation α is called *strong* if α_f is a natural isomorphism for every $f : X \to Y$ in $\mathscr{C}$, and it is called *strict* if α_f is exactly the identity for every f.

3.1∗5 Definition. Given $\ddot{\Psi} \in \mathcal{F}$, a (strong/strict) *modular model* M of $\ddot{\Psi}$ is a (strong/strict) oplax transformation from $(-)$-$\textsc{Alg}$ to $(- + \ddot{\Psi})$-$\textsc{Alg} : \mathcal{F}^{\mathrm{op}} \to \mathrm{CAT}$.

3.1∗6. Unpacking the definition, a modular model M of $\ddot{\Psi} \in \mathcal{F}$ consists of a family of functors $M_{\ddot{\Sigma}} : \ddot{\Sigma}\text{-}\textsc{Alg} \to (\ddot{\Sigma} + \ddot{\Psi})\text{-}\textsc{Alg}$ for all $\ddot{\Sigma} \in \mathcal{F}$ and a family of

natural transformations $M_T : M_{\ddot{\Sigma}} \circ T \to (T + \ddot{\Psi}) \circ M_{\ddot{\Sigma}'}$ for all $T : \ddot{\Sigma} \to \ddot{\Sigma}'$ in $\mathcal{F}$:

$$\begin{array}{ccc} \ddot{\Sigma}\text{-Alg} & \xleftarrow{\quad T \quad} & \ddot{\Sigma}'\text{-Alg} \\ {\scriptstyle M_{\ddot{\Sigma}}}\downarrow & \swarrow\!\!\!\swarrow {\scriptstyle M_T} & \downarrow{\scriptstyle M_{\ddot{\Sigma}'}} \\ (\ddot{\Sigma} + \ddot{\Psi})\text{-Alg} & \xleftarrow[T+\ddot{\Psi}]{} & (\ddot{\Sigma}' + \ddot{\Psi})\text{-Alg} \end{array}$$

such that M_{id} is the identity transformation, and for a pair of morphisms $T : \ddot{\Sigma} \to \ddot{\Sigma}'$ and $T' : \ddot{\Sigma}' \to \ddot{\Sigma}''$, $M_{T' \cdot T}$ is exactly the pasting of M_T and $M_{T'}$:

$$\begin{array}{ccccc} \ddot{\Sigma}\text{-Alg} & \xleftarrow{\quad T \quad} & \ddot{\Sigma}'\text{-Alg} & \xleftarrow{\quad T' \quad} & \ddot{\Sigma}''\text{-Alg} \\ {\scriptstyle M_{\ddot{\Sigma}}}\downarrow & \swarrow\!\!\!\swarrow {\scriptstyle M_T} & \downarrow{\scriptstyle M_{\ddot{\Sigma}'}} & \swarrow\!\!\!\swarrow {\scriptstyle M_{T'}} & \downarrow{\scriptstyle M_{\ddot{\Sigma}''}} \\ (\ddot{\Sigma} + \ddot{\Psi})\text{-Alg} & \xleftarrow[T+\ddot{\Psi}]{} & (\ddot{\Sigma}' + \ddot{\Psi})\text{-Alg} & \xleftarrow[T'+\ddot{\Psi}]{} & (\ddot{\Sigma}'' + \ddot{\Psi})\text{-Alg} \end{array}$$

Specially, the data of a *strict* modular model M of $\ddot{\Psi} \in \mathcal{F}$ is only a family of functors $M_{\ddot{\Sigma}}$ such that the following diagram in CAT commutes:

$$\begin{array}{ccc} \ddot{\Sigma}\text{-Alg} & \xleftarrow{\quad T \quad} & \ddot{\Sigma}'\text{-Alg} \\ {\scriptstyle M_{\ddot{\Sigma}}}\downarrow & & \downarrow{\scriptstyle M_{\ddot{\Sigma}'}} \\ (\ddot{\Sigma} + \ddot{\Psi})\text{-Alg} & \xleftarrow[T+\ddot{\Psi}]{} & (\ddot{\Sigma}' + \ddot{\Psi})\text{-Alg} \end{array} \tag{3.1}$$

Therefore a strict modular model for $\ddot{\Psi}$ is exactly an ordinary natural transformation between the functors $(-)$-Alg and $(- + \ddot{\Psi})$-Alg : $\mathcal{F}^{\mathrm{op}} \to \mathrm{CAT}$.

3.1∗7 Example. For a trivial example, let $\mathcal{F}$ be the operation family containing only the initial object $\langle \textsc{Mon}, \mathrm{Id} : \textsc{Mon} \to \textsc{Mon} \rangle$ of $\textsc{Mon}/\textsc{Eqs}(\mathscr{E})$. In this case, $\langle \textsc{Mon}, \mathrm{Id} \rangle + \langle \textsc{Mon}, \mathrm{Id} \rangle$ is still $\langle \textsc{Mon}, \mathrm{Id} \rangle$, so a modular model of $\langle \textsc{Mon}, \mathrm{Id} \rangle$ is exactly an endofunctor $\textsc{Mon}(\mathscr{E}) \to \textsc{Mon}(\mathscr{E})$ over the category of monoids in $\mathscr{E}$.

3.1∗8 Example. Let $\mathscr{E}$ be $\langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ for $\mathrm{l}\kappa\mathrm{p}\ \mathscr{C}$. A strict modular model M for the theory $\textsc{Et}_E$ of *exception throwing* (Example 2.4∗10) in the family $\textsc{Alg}(\mathscr{E})$ of algebraic operations is given by a family of functors

$$M_{\ddot{\Sigma}} : \ddot{\Sigma}\text{-Alg} \to (\ddot{\Sigma} + \textsc{Et}_E)\text{-Alg}$$

natural in $\ddot{\Sigma} \in \textsc{Alg}(\mathscr{E})$. Recall that each $\langle \dot{\Sigma}, T_\Sigma \rangle \in \textsc{Alg}(\mathscr{E})$ is of the form

$$(S \,\square\, -)\text{-Mon} \upharpoonleft (\mathrm{K}_G \vdash L = R)$$

for some S and $G \in \textsc{Endo}_\kappa(\mathscr{C})$, so objects of $\ddot{\Sigma}$-Alg are tuples

$$\langle A \in \textsc{Endo}_\kappa(\mathscr{C}),\ \ \alpha : S \circ A \to A,\ \ \eta^A : \mathrm{Id} \to A,\ \ \mu^A : A \circ A \to A \rangle$$

that are mapped by L and R to the same algebra. We define $M_{\ddot{\Sigma}}$ to send each of them to a $(\ddot{\Sigma} + \textsc{Et}_E)$-algebra carried by $C_A := A \circ (\bar{E} + \mathrm{Id})$, i.e. the *exception*

monad transformer applied to A, where $\bar{E}$ is the E-fold coproduct of the terminal object 1 in $\textsc{Endo}_\kappa(\mathscr{C})$. Using the internal language of $\textsc{Endo}_\kappa(\mathscr{C})$, the operations $[\alpha^\sharp, \beta] : (S \circ C_A) + \bar{E} \to C_A$ are

$$\begin{aligned}\alpha^\sharp &= [\![s : S, a : A, e : \bar{E} + \mathrm{Id} \vdash (\alpha(s,a),\ e) : C_A]\!] \\ \beta &= [\![e : \bar{E} \vdash (\eta^A,\ \iota_1\, e) : C_A]\!]\end{aligned}$$

and C_A has the following monad structure:

$$\begin{aligned}\eta^C &= [\![\vdash (\eta^A, \iota_2(*)) : C_A]\!] \\ \mu^C &= [\![a : A,\ e : \bar{E} + \mathrm{Id},\ a' : A,\ e' : \bar{E} + \mathrm{Id} \vdash \\ &\qquad \mathit{let}\ (a'', e'') = d(e, a')\ \mathit{in}\ (\mu^A(a, a''), \mu^{\bar{E}+\mathrm{Id}}(e'', e'))]\!]\end{aligned}$$

where $d : (\bar{E} + \mathrm{Id}) \circ A \to A \circ (\bar{E} + \mathrm{Id})$ is the following distributive law:

$$e : \bar{E} + \mathrm{Id},\ a : A \ \vdash \mathit{case}\ e\ \mathit{of}\ \{\ \iota_1\, e' \mapsto (\eta^A, \iota_1 e');\ \ \iota_2 * \mapsto (a, \iota_2 *) : C_A \},$$

and $\mu^{\bar{E}+\mathrm{Id}}$ is the multiplication of the exception monad $\bar{E} + \mathrm{Id}$:

$$x : \bar{E} + \mathrm{Id},\ y : \bar{E} + \mathrm{Id} \ \vdash \mathit{case}\ x\ \mathit{of}\ \{\ \iota_1\, e \mapsto \iota_1\, e;\ \ \iota_2 * \mapsto y \} : \bar{E} + \mathrm{Id}.$$

On morphisms, $M_{\ddot{\Sigma}}$ sends a $\dot{\Sigma}$-homomorphism $h : A \to B$ to $h \circ (\bar{E} + \mathrm{Id})$.

We need to check that the family of functors $M_{\ddot{\Sigma}}$ satisfies the naturality square (3.1): for all morphisms $T : \ddot{\Sigma} \to \ddot{\Sigma}'$, every object $\langle A, \alpha, \eta^A, \mu^A \rangle$ of $\ddot{\Sigma}'$-$\textsc{Alg}$ is mapped by functors $M_{\ddot{\Sigma}} \circ T$ and $(T + \mathrm{ET}_E) \circ M_{\ddot{\Sigma}'}$ to two objects in $(\ddot{\Sigma} + \mathrm{ET}_E)$-$\textsc{Alg}$ with the same carrier C_A, the same ET_E-algebra structure, and the same monad structure by construction. We need to show that the respective $\ddot{\Sigma}$-algebras on C_A are also the same: the $\ddot{\Sigma}$-algebra on C_A from $M_{\ddot{\Sigma}} \circ T$ is

$$T\alpha \circ (\bar{E} + \mathrm{Id}) : S \circ A \circ (\bar{E} + \mathrm{Id}) \to A \circ (\bar{E} + \mathrm{Id}),$$

and the one from $(T + \mathrm{ET}_E) \circ M_{\ddot{\Sigma}'}$ is $T(\alpha \circ (\bar{E} + \mathrm{Id}))$. By Lemma 2.4*13, it is sufficient to show that the following diagram commutes

$$S \xrightarrow{S \circ \eta^C} S \circ C_A \underset{T\alpha \circ (\bar{E}+\mathrm{Id})}{\overset{T(\alpha \circ (\bar{E}+\mathrm{Id}))}{\rightrightarrows}} C_A \tag{3.2}$$

To see this, we first observe that we have the following $\ddot{\Sigma}'$-homomorphism square:

$$\begin{array}{ccc} S' \circ A & \xrightarrow{\quad\alpha\quad} & A \\ {\scriptstyle S' \circ A \circ \iota_2}\downarrow & & \downarrow{\scriptstyle A \circ \iota_2} \\ S' \circ A \circ (\bar{E} + \mathrm{Id}) & \xrightarrow[\alpha \circ (\bar{E}+\mathrm{Id})]{} & A \circ (\bar{E} + \mathrm{Id}) \end{array}$$

and this square is mapped by the translation T to a commuting square

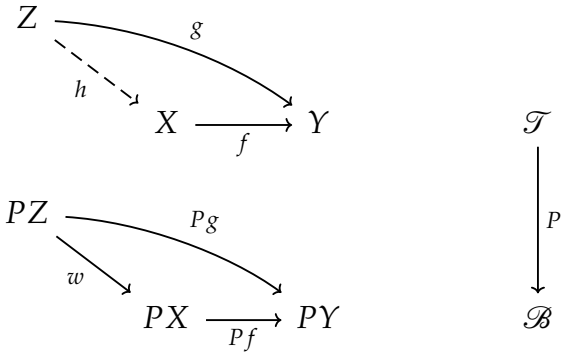
$$\begin{array}{ccc} S \circ A & \xrightarrow{T\alpha} & A \\ {\scriptstyle S\circ A\circ \iota_2}\downarrow & & \downarrow{\scriptstyle A\circ \iota_2} \\ S \circ A \circ (\bar{E} + \mathrm{Id}) & \xrightarrow[T(\alpha\circ(\bar{E}+\mathrm{Id}))]{} & A \circ (\bar{E} + \mathrm{Id}) \end{array}$$

This implies (3.2) because

$$\begin{aligned} & T(\alpha \circ (\bar{E} + \mathrm{Id})) \cdot (S \circ \eta^C) \\ = \quad & \{\text{definition of } \eta^C\} \\ & T(\alpha \circ (\bar{E} + \mathrm{Id})) \cdot (S \circ A \circ \iota_2) \cdot (S \circ \eta^A) \\ = \quad & \{\text{by the last commutativity square above}\} \\ & (A \circ \iota_2) \cdot T\alpha \cdot (S \circ \eta^A) \\ = \quad & \{\text{by naturality}\} \\ & (T\alpha \circ (\bar{E} + \mathrm{Id})) \cdot (A \circ \iota_2) \cdot (S \circ \eta^A) \\ = \quad & (T\alpha \circ (\bar{E} + \mathrm{Id})) \cdot (S \circ \eta^C) \end{aligned}$$

3.1∗9. CAT-valued functors also known as (strict) *indexed categories*, which are equivalent to *split fibrations* via the *Grothendieck construction*, so we can alternatively formulate modular models based on fibrations. The fibrational formulation is usually easier to work with, especially when we talk about morphisms between modular models later, which would be a 3-categorical concept in the indexed-category formulation. Also, the fibrational formulation slightly simplifies some 'dependently typed' constructions such as mapping each $\ddot{\Sigma}$ in $\mathcal{F}$ to the initial algebra in $\ddot{\Sigma}$-Alg. We will only need the very basics about fibrations (see e.g. Borceux [1994b]; Jacobs [1999]; Streicher [2023]), which we review below.

3.1∗10 (Fibrations). Let $P : \mathscr{T} \to \mathscr{B}$ be a functor. A morphism $f : X \to Y$ in $\mathscr{T}$ is called *cartesian* if for every $g : Z \to Y$ and $w : PZ \to PX$ such that $Pg = Pf \cdot w$, there is a unique $h : Z \to X$ in $\mathscr{T}$ satisfying $Ph = w$ and $f \cdot h = g$:



The functor P called a *(Grothendieck) fibration* if for every morphism $u : I \to J$

in $\mathscr{B}$ and object $Y \in \mathscr{T}$ such that $PY = J$, there exists a cartesian morphism $f : X \to Y$ in $\mathscr{T}$ with $Pf = u$. It is customary to call the category $\mathscr{T}$ the *total category* and $\mathscr{B}$ the *base category*. If an object X or a morphism f in $\mathscr{T}$ is sent by the functor P to an object I or a morphism u in $\mathscr{B}$, we colloquially say that X or f is *over* I or u. Given an object $I \in \mathscr{B}$, the *fiber category* $\mathscr{T}_I$ over I is the subcategory of $\mathscr{T}$ consisting of objects over I and morphisms over id_I.

A *cleavage* for a fibration $P : \mathscr{T} \to \mathscr{B}$ is an assignment κ of cartesian morphisms $\kappa(Y, u)$ over u to all pairs of $Y \in \mathscr{T}$ and $u : I \to PY$ for some $I \in \mathscr{B}$. A cleavage κ is said to be a *split cleavage* if it is functorial: $\kappa(Y, id_{PY}) = id_Y$ and $\kappa(Y, u \cdot v) = \kappa(Y, u) \cdot \kappa(\textsc{Dom}\, \kappa(Y, u), v)$. A *split fibration* is a fibration $P : \mathscr{T} \to \mathscr{B}$ equipped with a split cleavage.

Let $P : \mathscr{T} \to \mathscr{B}$ be a fibration with a cleavage κ. For every morphism $u : I \to J$ in the base category $\mathscr{B}$, the *reindexing functor* $u^* : \mathscr{T}_J \to \mathscr{T}_I$ sends every object $Y \in \mathscr{T}_J$ to the domain object $X \in \mathscr{T}_I$ of the morphism $\kappa(Y, u) : X \to Y$ and sends every morphism $f : Y \to Y'$ in $\mathscr{T}_I$ to the morphism $h : u^*Y \to u^*Y'$ as follows:

$$\begin{array}{ccc} u^*Y & \xrightarrow{\kappa(Y,u)} & Y \\ & \searrow^{h} & \downarrow{\scriptstyle f} \\ & u^*Y' \xrightarrow[\kappa(Y',u)]{} & Y' \end{array} \qquad \begin{array}{ccc} I & \xrightarrow{u} & J \\ & \searrow^{id} & \downarrow{\scriptstyle id} \\ & I \xrightarrow[u]{} & J \end{array}$$

where h is obtained from the cartesianess of $\kappa(Y', u) : u^*Y' \to Y'$ and the fact that $P(f \cdot \kappa(Y, u)) = id \cdot u = u = P(\kappa(Y', u))$. The functoriality of $u^* : \mathscr{T}_J \to \mathscr{T}_I$ is a consequence of the uniqueness part of cartesianess.

A morphism $P \to P'$ of fibrations is a pair of functors $\langle F, G\rangle$ such that the diagram below commutes and F maps cartesian morphisms to cartesian morphisms. A morphism $\langle P, \kappa\rangle \to \langle P', \kappa'\rangle$ of split fibrations is a morphism $\langle F, G\rangle : P \to P'$ that strictly preserves the split cleavage: $F\kappa(Y, u) = \kappa(FY, Gu)$.

$$\begin{array}{ccc} \mathscr{T} & \xrightarrow{F} & \mathscr{T}' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} \\ \mathscr{B} & \xrightarrow[G]{} & \mathscr{B}' \end{array} \tag{3.3}$$

With component-wise identity functors and functor composition, fibrations (resp. split fibrations) and morphisms form a category $\textsc{Fib}$ (resp. $\textsc{Fib}^s$). Also, given a category $\mathscr{B}$, there is a subcategory $\textsc{Fib}_{\mathscr{B}} \subseteq \textsc{Fib}$ (resp. $\textsc{Fib}^s_{\mathscr{B}} \subseteq \textsc{Fib}^s$) containing all (resp. split) fibrations over $\mathscr{B}$ and morphisms $\langle F, \mathrm{Id} : \mathscr{B} \to \mathscr{B}\rangle : P \to P'$.

Let $P : \mathscr{T} \to \mathscr{B}$ be a fibration and $F : \mathscr{C} \to \mathscr{B}$ be a functor. A basic result [Jacobs 1999, Lemma 1.5.1] in fibred category theory is that the pullback $F^*P : F^*\mathscr{T} \to \mathscr{C}$ of P along F in the (1-)category CAT is still a fibration:

$$\begin{array}{ccc} F^*\mathscr{T} & \longrightarrow & \mathscr{T} \\ {\scriptstyle F^*P}\downarrow & \lrcorner & \downarrow{\scriptstyle P} \\ \mathscr{C} & \xrightarrow[F]{} & \mathscr{B} \end{array}$$

The fibration F^*P is called the *change of base* of P along F. Moreover, if P has a split cleavage then so does F^*P. Explicitly, the objects of $F^*\mathscr{T}$ are pairs $\langle I \in \mathscr{C}, X \in \mathscr{T} \rangle$ such that $FI = PX$, the morphisms $\langle I, X \rangle \to \langle J, Y \rangle$ in $F^*\mathscr{T}$ are pairs $\langle f : I \to J, g : X \to Y \rangle$ such that $Ff = Pg$. A morphism $\langle f, g \rangle$ in $F^*\mathscr{T}$ is cartesian if and only if g is cartesian.

3.1∗11 Definition. Given a functor $F : \mathscr{B}^{\mathrm{op}} \to \mathrm{CAT}$, the *Grothendieck construction* is a split fibration $P : \int F \to \mathscr{B}$ where the category $\int F$ has tuples $\langle I \in \mathscr{B}, a \in FI \rangle$ as objects, and its morphisms $\langle I, a \rangle \to \langle J, a' \rangle$ are pairs $\langle f, g \rangle$ for $f : I \to J$ in $\mathscr{B}$ and $g : a \to Ffa'$ in the category FI. Identity arrows are $\langle id, id \rangle$ and composition $\langle f', g' \rangle \cdot \langle f, g \rangle$ is $\langle f' \cdot f, g' \cdot Ff'g \rangle$. The fibration $P : \int F \to \mathscr{B}$ is simply the projection functor $\langle I, a \rangle \mapsto I$ for the first component. The split cleavage $\kappa(\langle I, a \rangle, u)$ for some $u : J \to I$ is $\langle u, id \rangle : \langle J, (Fu)a \rangle \to \langle I, a \rangle$.

3.1∗12. Applying the Grothendieck construction to $(-)$-Alg $: \mathcal{F}^{\mathrm{op}} \to \mathrm{CAT}$, we obtain a (split) fibration $P : \mathcal{F}$-Alg $\to \mathcal{F}$, which we explicitly describe below. The intuition is that $\mathcal{F}$-Alg is the category of *all models of all equational systems* in $\mathcal{F}$. The objects of the category $\mathcal{F}$-Alg are tuples

$$\langle \dot{\Sigma} \in \mathrm{Eqs}(\mathscr{E}), \quad T_\Sigma : \mathrm{Mon} \to \dot{\Sigma}, \quad A \in \mathscr{E}, \quad \alpha : \Sigma A \to A \rangle$$

such that $\langle \dot{\Sigma}, T_\Sigma \rangle \in \mathcal{F}$ and $\langle A, \alpha \rangle \in \dot{\Sigma}$-Alg. Morphisms between two objects $\langle \dot{\Sigma}, T_\Sigma, A, \alpha \rangle$ and $\langle \dot{\Psi}, T_\Psi, B, \beta \rangle$ are pairs $\langle T, h \rangle$ where $T : \dot{\Sigma} \to \dot{\Psi}$ is a functorial translation in $\mathcal{F}$, and the other component $h : A \to B \in \mathscr{E}$ is a $\dot{\Sigma}$-algebra homomorphism from $\langle A, \alpha \rangle$ to $T\langle B, \beta \rangle$:

$$\begin{array}{ccc} \Sigma B & \xleftarrow{\Sigma h} & \Sigma A \\ {\scriptstyle T\beta}\downarrow & & \downarrow{\scriptstyle \alpha} \\ B & \xleftarrow{h} & A \end{array} \quad \text{in } \dot{\Sigma}\text{-Alg} \qquad \xleftarrow{T} \qquad \begin{array}{c} \Psi B \\ \downarrow{\scriptstyle \beta} \\ B \end{array} \quad \text{in } \dot{\Psi}\text{-Alg}$$

The identities in $\mathcal{F}$-Alg are pairs $\langle \mathrm{Id} : \dot{\Sigma} \to \dot{\Sigma}, id : A \to A \rangle$, and the composition of two morphisms $\langle T, h \rangle$ and $\langle T', h' \rangle$ is $\langle T \circ T', h \cdot h' \rangle$.

The fibration $P : \mathcal{F}$-Alg $\to \mathcal{F}$ is the projection: $P\langle \dot{\Sigma}, T_\Sigma, A, \alpha \rangle = \langle \dot{\Sigma}, T_\Sigma \rangle$ and $P\langle T, h \rangle = T$. It has a split cleavage assigning to every pair of a morphism $T : \langle \dot{\Sigma}, T_\Sigma \rangle \to \langle \dot{\Psi}, T_\Psi \rangle \in \mathcal{F}$ and an object $\langle \dot{\Psi}, T_\Psi, B, \beta \rangle \in \mathcal{F}$-Alg a cartesian

morphism $\langle T, id\rangle : \langle \dot\Sigma, T_\Sigma, B, T\beta\rangle \to \langle \dot\Psi, T_\Psi, B, \beta\rangle$ in $\mathcal{F}$-Alg.

3.1∗13 Example. Given an object $\ddot\Psi \in \mathcal{F}$, the Grothendieck construction of the functor $(- + \ddot\Psi)$-Alg : $\mathcal{F}^{\mathrm{op}} \to$ CAT is a split fibration $Q : (\mathcal{F} + \ddot\Psi)$-Alg $\to \mathcal{F}$. Explicitly, the objects of $(\mathcal{F} + \ddot\Psi)$-Alg are tuples:

$$\langle \dot\Sigma \in \mathrm{Eqs}(\mathscr{E}),\quad T_\Sigma : \mathrm{Mon} \to \dot\Sigma,\quad A \in \mathscr{E},\quad \alpha : \Sigma A \to A,\quad \beta : \Psi A \to A\rangle$$

such that $\langle \dot\Sigma, T_\Sigma\rangle \in \mathcal{F}$, $\langle A, \alpha\rangle \in \dot\Sigma$-Alg, $\langle A, \beta\rangle \in \dot\Psi$-Alg, and $T_\Psi \alpha = T_\Sigma \beta$. Morphisms in $(\mathcal{F} + \dot\Psi)$-Alg are similar to those $\langle T, h\rangle$ in $\mathcal{F}$-Alg, but require h also to be a $\dot\Psi$-homomorphism. Therefore, objects of $(\mathcal{F} + \ddot\Psi)$-Alg are models of some equational systems in $\mathcal{F}$ that are additionally equipped with a $\ddot\Psi$-algebra. The functor Q is the projection $\langle \dot\Sigma, T, A, \alpha, \beta\rangle \mapsto \langle \dot\Sigma, T\rangle$.

The split fibration $Q : (\mathcal{F} + \ddot\Psi)$-Alg $\to \mathcal{F}$ can be alternatively given as the change of base of the fibration $P : \mathcal{F}$-Mon $\to \mathcal{F}$ along the functor $(- + \ddot\Psi) : \mathcal{F} \to \mathcal{F}$, which is the following pullback in the category CAT of large categories:

$$\begin{array}{ccc} (\mathcal{F} + \ddot\Psi)\text{-Alg} & \xrightarrow{K} & \mathcal{F}\text{-Alg} \\ {\scriptstyle Q}\downarrow & \lrcorner & \downarrow{\scriptstyle P} \\ \mathcal{F} & \xrightarrow[-+\ddot\Psi]{} & \mathcal{F} \end{array} \tag{3.4}$$

The functor K maps objects $\langle \dot\Sigma, T_\Sigma, A, \alpha, \beta\rangle$ to $\langle \langle \dot\Sigma, T_\Sigma\rangle + \langle \dot\Psi, T_\Psi\rangle, A, [\alpha, \beta]\rangle$. The pair $\langle K, (- + \dot\Psi)\rangle$ is a morphism of split fibrations $Q \to P$.

3.1∗14 Example. Let $\mathscr{C}$ be a category with finite limits. For every $X \in \mathscr{C}$, the full subcategory of the slice category $\mathscr{C}/X$ containing monomorphisms is a preorder. Considering the elements of this preorder up to isomorphism, we obtain a partial order $\mathrm{Sub}(X)$, whose elements are called *subobjects* of X. This extends to a functor $\mathrm{Sub} : \mathscr{C}^{\mathrm{op}} \to$ CAT which acts on $f : X \to Y$ by pulling back along f:

$$\begin{array}{ccc} f^*A & \longrightarrow & A \\ \downarrow & & \downarrow \\ X & \xrightarrow[f]{} & Y \end{array}$$

The Grothendieck construction of Sub gives us a split fibration $\mathcal{S} \to \mathscr{C}$, where the total category $\mathcal{S}$ has as objects $\langle X \in \mathscr{C}, A \in \mathrm{Sub}(X)\rangle$, and a morphism $\langle X, A\rangle \to \langle Y, B\rangle$ is a $\mathscr{C}$-morphism $f : X \to Y$ such that there is a (necessarily unique) i satisfying $f \cdot a = b \cdot i$ for all representative elements a of A and b of B:

$$\begin{array}{ccc} \mathrm{Dom}\, a & \xrightarrow{i} & \mathrm{Dom}\, b \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ X & \xrightarrow[f]{} & Y \end{array}$$

The category $\mathcal{S}$ can be understood as the category of *predicates* on $\mathcal{C}$-objects.

3.1∗15 Definition. Given two fibrations $P : \mathcal{E} \to \mathcal{B}$ and $P' : \mathcal{E}' \to \mathcal{B}'$, a *lifting* of a functor $F : \mathcal{B} \to \mathcal{B}'$ along P and P' is a functor $G : \mathcal{E} \to \mathcal{E}'$ making the following diagram commute strictly:

$$\begin{array}{ccc} \mathcal{E} & \xrightarrow{G} & \mathcal{E}' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} \\ \mathcal{B} & \xrightarrow[F]{} & \mathcal{B}' \end{array}$$

The lifting is called *fibred* if $\langle F, G\rangle$ is a morphism of fibrations (i.e. G preserves all cartesian morphisms). When P and P' have split cleavages, the lifting G is called *split* if $\langle F, G\rangle$ is a morphism of split fibrations.

A morphism between two liftings $G \to H$ of F is a natural transformation $\sigma : M \to N$ that is vertical, i.e. $P'\sigma = id_{T \circ P}$:

$$\begin{array}{ccc} & \overset{G}{\curvearrowright} & \\ \mathcal{E} & \Downarrow \sigma & \mathcal{E}' \\ {\scriptstyle P}\downarrow & \underset{H}{\curvearrowright} & \downarrow{\scriptstyle P'} \\ \mathcal{B} & \xrightarrow[F]{} & \mathcal{B}' \end{array}$$

Liftings of F along P and P' and morphisms between them form a category $\textsc{Lift}_{P,P'}(F)$, whose identity morphisms are the identity natural transformations, and composition is vertical composition of natural transformations.

3.1∗16 (Modular models as liftings). Using the language of fibrations, we have now a very concise alternative formulation of modular models: a (strong/strict) modular model M of $\ddot{\Psi} \in \mathcal{F}$ is just a (fibred/split) lifting of the endofunctor $(- + \ddot{\Psi}) : \mathcal{F} \to \mathcal{F}$ along the fibration $P : \mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}$ from 3.1∗12:

$$\begin{array}{ccc} \mathcal{F}\text{-}\textsc{Alg} & \xrightarrow{M} & \mathcal{F}\text{-}\textsc{Alg} \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P} \\ \mathcal{F} & \xrightarrow[-+\ddot{\Psi}]{} & \mathcal{F} \end{array}$$

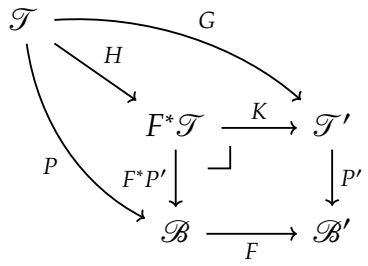
The commutativity of the square ensures that the functor M maps every object $\langle \ddot{\Sigma}, A, \alpha\rangle$ to an object $\langle \ddot{\Sigma} + \ddot{\Psi}, B, \beta\rangle$. Below, we show that this formulation of modular models is equivalent to Definition 3.1∗5 based on indexed categories.

3.1∗17 Lemma. Given two fibrations $P : \mathcal{T} \to \mathcal{B}$, $P' : \mathcal{T}' \to \mathcal{B}'$, and a functor $F : \mathcal{B} \to \mathcal{B}'$, let $F^*P' : F^*\mathcal{T}' \to \mathcal{B}$ be the change of base of P' along F.

1. Liftings of F along P and P' are in bijection with $\mathrm{CAT}/\mathcal{B}(P, F^*P')$.
2. Fibred liftings of F along P and P' are in bijection with $\textsc{Fib}_{\mathcal{B}}(P, F^*P')$.

3. If P and P' are equipped with split cleavages, split liftings of F along P and P' are in bijection with $\mathrm{Fib}^{s}_{\mathscr{B}}(P, F^*P')$.

Proof. By definition, F^*P' is the pullback in the following diagram:



By definition, liftings of F are functors $G : \mathscr{T} \to \mathscr{T}'$ such that $F \circ P = P' \circ G$, so by the universal property of the pullback, liftings G are in bijection with functors $H : \mathscr{T} \to F^*\mathscr{T}'$ such that $(F^*P') \circ H = P$, and the backward direction is given by $H \mapsto K \circ H$. This is the first item in the statement.

This bijection cuts down to fibred (resp. split) liftings. The functor K always preserves cartesian morphisms and the split cleavage. Hence from one direction, if H preserves cartesian morphisms (resp. split cleavage), then so does $K \circ H$. From the other direction, if G preserves cartesian morphisms (resp. split cleavage), then the functor H sends a cartesian morphism f in $\mathscr{T}$ to the pair $\langle Pf, Gf \rangle$, which is cartesian with respect to the fibration F^*P'. □

3.1∗18 Lemma. For a category $\mathscr{C}$ and two functors $F, G : \mathscr{C}^{\mathrm{op}} \to \mathrm{CAT}$, denote the Grothendieck construction of F and G by $p : \int F \to \mathscr{C}$ and $q : \int G \to \mathscr{C}$. Oplax transformations $F \to G$ are in bijection with $\mathrm{CAT}/\mathscr{C}(p, q)$.

Proof. For one direction, given an oplax transformation $\alpha : F \to G$, we define a functor $K : \int F \to \int G$ as follows. On objects, K sends every object $\langle I \in \mathscr{C}, a \in FI \rangle$ to $\langle I, \alpha_I a \in GI \rangle$. On morphisms, K sends every morphism

$$\langle f : I \to J, g : a \to (Ff)b \rangle : \langle I, a \rangle \to \langle J, b \rangle$$

in $\int F$ to $\langle f, g' \rangle$ where g' is the following morphism in the fiber category G_I:

$$\alpha_I a \xrightarrow{\alpha_I g} \alpha_I(Ffb) \xrightarrow{\alpha_{f,b}} (Gf)\alpha_J b.$$

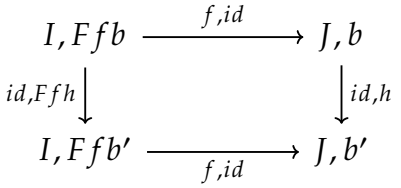
The functor preserves identities and composition following the unity and composition axioms of oplax transformations.

For the other direction, given a functor $K : \int F \to \int G$ with $q \circ F = p$, we define an oplax transformation $\alpha : F \to G$ as follows. For every object $I \in \mathscr{C}$, we define the functor $\alpha_I : FI \to GI$ to be K restricted to the fiber category of $\int F$ over I. For every $f : I \to J$ in $\mathscr{C}$, we need to define a natural transformation $\alpha_f : \alpha_I \circ Ff \to Gf \circ \alpha_J : FJ \to GI$. For each object $b \in FJ$, there is a morphism

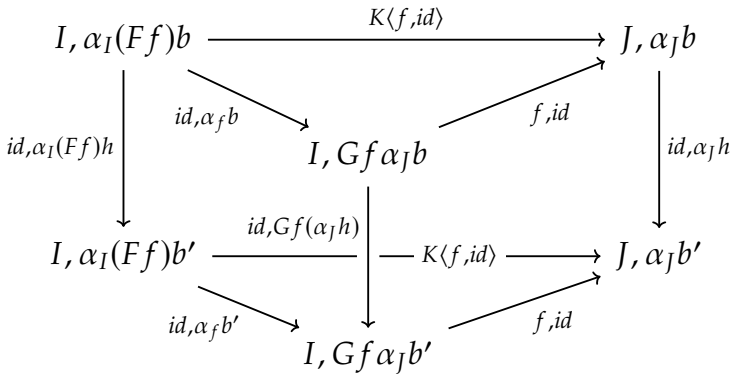
$\langle f, id\rangle : \langle I, (Ff)b\rangle \to \langle J, b\rangle$ in the total category $\int F$, and this morphism is mapped by K to some $\langle f, g\rangle : \langle I, \alpha_I(Ffb)\rangle \to \langle J, \alpha_J b\rangle$, we define the natural transformation α_f to be

$$\alpha_{f,b} := g : \alpha_I(Ffb) \to Gf(\alpha_J b)$$

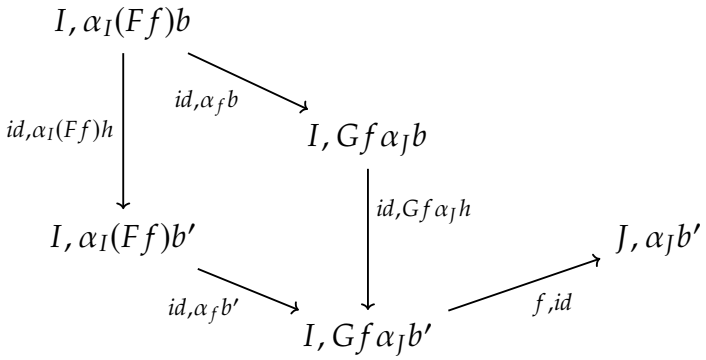
in the fiber category FI. For every $h : b \to b'$ in FJ, the naturality of α_f follows from the fact that the following diagram commutes in $\int F$:



and K maps this diagram to $\int G$, which is the back square in



We observe that in this diagram the two triangles commute by the definition of $\alpha_f b$ and $\alpha_f b'$; the right diagram commute by the definition of arrow composition in $\int G$. Hence the following diagram commutes, and the square on the left is exactly the naturality square for α_f that we need, which also commutes because the morphism $\langle f, id\rangle$ on the right is cartesian:



It can be checked that this natural transformations satisfies the axioms of oplax transformations and that the two directions define a bijection. □

3.1∗19 Theorem. *For every $\ddot{\Psi} \in \mathcal{F}$, the following are in bijection with each other*:

1. *modular models* (*resp. strong or strict modular models*) *as in Definition* 3.1∗5,
2. *functors in* $\mathrm{CAT}/\mathcal{F}(P, Q)$ (*resp. morphisms of fibrations or split fibrations from* P *to* Q) *where the split fibrations* P *and* Q *are as in* (3.4), *and*
3. *liftings* (*resp. fibred or split liftings*) *of the endofunctor* $(- + \ddot{\Psi}) : \mathcal{F} \to \mathcal{F}$ *along the fibration* $P : \mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}$.

Proof. The bijection between 2 and 3 is Lemma 3.1∗17. The bijection between modular models and morphisms in $\mathrm{CAT}/\mathcal{F}$ is Lemma 3.1∗18. The bijection between strong (resp. strict) modular models – which are exactly strong transformations (resp. natural transformations) between CAT-valued functors – and morphisms of fibrations (resp. split fibrations) is standard [Jacobs 1999, §1.10]. □

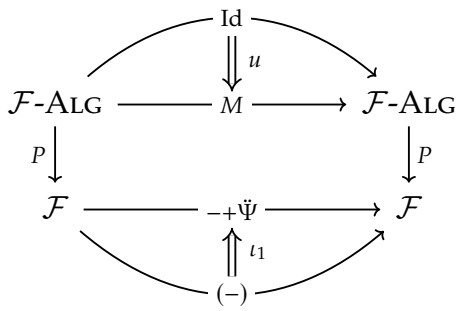
3.1∗20 Notation. In the future, we will leave implicit the conversion between modular models as oplax transformations and as liftings of functors, so we may say 'a modular model $M : \mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}\text{-}\textsc{Alg}$ of $\ddot{\Psi}$'.

3.1∗21. One advantage of the fibrational formulation is that we avoid the need of a category of categories CAT bigger than the base category. Also, it reduces the 2-categorical notion of oplax transformations to the 1-categorical notion of functors. Consequently, 3-categorical concepts can be avoided when talking about transformations of modular models, such as homomorphisms between modular models and the concept of *updaters* below.

3.1∗22 Definition. Let M be a modular model of $\ddot{\Psi} \in \mathcal{F}$ given in the lifting form. An *updater* u for M is a natural transformation $u : \mathrm{Id} \to M$ such that

$$(P \circ u) = (\iota_1 \circ P) : P \to (P \circ M) = (- + \ddot{\Psi}) \circ P$$

where $\iota_1 : (-) \to (-) + \ddot{\Psi}$ is the coprojection in $\mathcal{F}$:



3.1∗23. For an object $\langle \ddot{\Sigma}, A, \alpha \rangle \in \mathcal{F}\text{-}\textsc{Alg}$, the component of u at this object is a $\ddot{\Sigma}$-homomorphism from $\langle A, \alpha \rangle$ to the algebra $M\langle \ddot{\Sigma}, A, \alpha \rangle$ forgetting the $\ddot{\Psi}$-algebra. If we informally think of $\ddot{\Sigma}$ as a programming language, $\ddot{\Psi}$ as the new feature in a new release of the programming language, and $\langle A, \alpha \rangle$ as the existing compiled

programs of $\ddot{\Sigma}$, then the role of u is updating existing compiled programs to the new version, hence its name 'updater'.

3.1∗24. In the setting of Example 3.1∗7, updaters correspond to exactly the *lifting operation* $l : \mathrm{Id} \to T : \textsc{Mon}(\mathscr{E}) \to \textsc{Mon}(\mathscr{E})$ of monad/monoid transformers [Jaskelioff and Moggi 2010]. We have switched to the terminology *lifting* to avoid the confusion with liftings along fibrations (Definition 3.1∗15).

3.1∗25. Assuming objects of $\mathcal{F}$ have initial algebras, the 'dependently typed' mapping sending every $\langle \dot{\Sigma}, T_\Sigma \rangle \in \mathcal{F}$ to its initial algebra $\mu\dot{\Sigma}$ can be conveniently formulated as a functor $(-)^\star : \mathcal{F} \to \mathcal{F}\text{-}\textsc{Alg}$ using the fibrational language:

$$\begin{aligned} \ddot{\Sigma}^\star &= \langle \dot{\Sigma},\ T_\Sigma,\ \mu\dot{\Sigma},\ \alpha^\Sigma : \Sigma(\mu\dot{\Sigma}) \to \mu\dot{\Sigma} \rangle \\ (T : \ddot{\Sigma} \to \ddot{\Psi})^\star &= \langle T,\ !\, : \langle \mu\dot{\Sigma},\ \alpha^\Sigma \rangle \to T\langle \mu\dot{\Psi},\ \alpha^\Psi \rangle \rangle \end{aligned} \tag{3.5}$$

where ! is the unique $\dot{\Sigma}$-homomorphism out of the initial algebra $\mu\dot{\Sigma}$.

Let M be a modular model of some $\ddot{\Psi} \in \mathcal{F}$ in the lifting form. For every $\ddot{\Sigma} \in \mathcal{F}$, we have a unique $(\ddot{\Sigma} + \ddot{\Psi})$-homomorphism out of the initial algebra $(\ddot{\Sigma} + \ddot{\Psi})^\star$:

$$h^M_{\ddot{\Sigma}} : (\ddot{\Sigma} + \ddot{\Psi})^\star \to M\ddot{\Sigma}^\star, \tag{3.6}$$

which defines a natural transformation

$$h^M : (- + \ddot{\Psi})^\star \to M(-)^\star : \mathcal{F} \to \mathcal{F}\text{-}\textsc{Alg}. \tag{3.7}$$

This is how M modularly handles/interprets $\dot{\Psi}$-operations in programs $(\ddot{\Sigma} + \ddot{\Psi})^\star$ with both $\ddot{\Psi}$-operations and some other $\ddot{\Sigma}$-operations, leaving operations from the other theory $\ddot{\Sigma}$ uninterpreted. Specially, $\ddot{\Sigma}$ can be the initial object $\langle \textsc{Mon}, \mathrm{Id} \rangle$ of $\mathcal{F}$, whose initial algebra is the initial monoid I. In this case, the morphism $h^M_{\textsc{Mon},\mathrm{Id}} : \ddot{\Psi}^\star \to MI$ interprets the abstract syntax $\ddot{\Psi}^\star$ with no 'residual operations'.

3.1∗26 Remark. Since $\ddot{\Sigma} + \ddot{\Psi}$ extends the theory of monoids, the interpretation morphism (3.6) is always a monoid morphism, so it preserves monoid multiplication μ. This is called the *semantic substitution lemma* [Tennent 1991] for $\mathscr{E} = \langle \textsc{Set}^{\textsc{Fin}}, \bullet, V \rangle$ since μ stands for substitution in this case.

3.2 Model Transformers

3.2∗1. In the previous section, we have seen modular models of theories of monoids with operations as liftings of endofunctors $- + \ddot{\Psi}$ along a fibration P:

$$\begin{array}{ccc} \mathcal{F}\text{-}\textsc{Alg} & \xrightarrow{M} & \mathcal{F}\text{-}\textsc{Alg} \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P} \\ \mathcal{F} & \xrightarrow[-+\ddot{\Psi}]{} & \mathcal{F} \end{array} \tag{3.8}$$

But there is no reason that the idea of modular models is specific to coproducts $- + \ddot{\Psi}$ or the fibration $\mathcal{F}$-Alg $\to \mathcal{F}$ for algebras and theories of monoids with operations, since we may be interested in ways of combining algebraic theories other than coproducts, and we may be interested in other fibrations of theories and algebras. Thus we shall just study liftings of arbitrary functors along two possibly different fibrations. In this section, we justify the generalisation by showing a few more instances of liftings along fibrations related to modularity.

3.2∗2 (Commutative combination). Consider $\mathscr{E} = \langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id} \rangle$ for some $\mathscr{C}$ that is lκp as a cartesian closed category, as ωCpo and Set. For functors $A_1, A_2 \in \textsc{Endo}_\kappa(\mathscr{C})$, we had seen the *Day tensor product* $A_1 \otimes A_2$ in Section 2.1.6:

$$(A_1 \otimes A_2)n = \int^{m,k \in \mathscr{C}_\kappa} A_1 m \times A_2 k \times n^{m \times k},$$

which is intuitively a pair of A_1-operation and A_2-operation that do not depend on each other. This is clearly symmetric, so we have an isomorphism

$$s : A_1 * A_2 \cong A_2 * A_1.$$

Also, there is a canonical morphism $i : A_1 * A_2 \to A_1 \circ A_2$ as shown in 2.1.6∗3.

We define a functor $\otimes : \textsc{Alg}(\mathscr{E}) \times \textsc{Alg}(\mathscr{E}) \to \textsc{Alg}(\mathscr{E})$ as follows, which is closely related to the *commutative combination*, also known as the *tensor*, of enriched algebraic theories [Hyland et al. 2006]. Let $\ddot{\Sigma}_i \in \textsc{Alg}(\mathscr{E})$ be

$$\langle (A_i \,\square\, -)\text{-Mon} \upharpoonleft (\mathrm{K}_{B_i} \vdash L_i = R_i),\ T_i \rangle, \text{ for } i = 1, 2.$$

We define $\ddot{\Sigma}_1 \otimes \ddot{\Sigma}_2$ to be $\ddot{\Sigma}_1 + \ddot{\Sigma}_2$ extended with the following (constant) equation

$$o : A_1 * A_2 \vdash f\ (i\ o) = f\ (i\ (s\ o)) : \tau.$$

where the term $o' : A_1 \circ A_2 \vdash f : \tau$ is defined by

$$o' : A_1 \circ A_2 \vdash \mathit{let}\ (a_1, a_2) = o'\ \mathit{in}\ \mu\ (op_1\ (a_1, \eta\ *),\ op_2\ (a_2, \eta\ *)) : \tau,$$

and $op_i : A_i \circ \tau \to \tau$, $\eta : I \to \tau$, $\mu : \tau \circ \tau \to \tau$ are respectively the algebraic operations, the unit, and the multiplication of the monoid.

Informally, a model of $\ddot{\Sigma}_1 \otimes \ddot{\Sigma}_2$ is a monad equipped with A_1-operation and A_2-operation such that the order of an adjacent pair of an A_1-operation and an A_2-operation can be swapped:

$$\textsc{do}\ \{x \leftarrow op_1; y \leftarrow op_2; k\ x\ y\} = \textsc{do}\ \{y \leftarrow op_2; x \leftarrow op_1; k\ x\ y\}.$$

The Day tensor $A_1 * A_2$ shows up in the equation to ensure that each of the two operations do not depend on the other's output, so that it is possible to swap them. The functor $\otimes$ can be extended to monoidal product on $\textsc{Alg}(\mathscr{E})$ with the theory of monoids with no operations as the unit.

Let $\mathcal{F}$ be $\textsc{Alg}(\mathscr{E})$ and $\ddot{\Psi} \in \mathcal{F}$, a lifting of $- \otimes \ddot{\Psi}$ along $P : \mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}$ is then a modular model of $\ddot{\Psi}$ such that operations of $\ddot{\Psi}$ commute with all other operations, even if they may be unknown now. This may sound very strong, but the *state monad transformer* gives such a modular model [Yang and Wu 2021, §6].

3.2∗3 (Modular handlers). The theory of modular models is inspired by the concept of *modular handlers* studied by Schrijvers et al. [2019] and Yang and Wu [2021] in the setting of Haskell programming. If we re-express them in category theory, a reasonable definition of *a modular handler of an equational system* $\dot{\Sigma}$ *over a category* $\mathscr{C}$ is a mapping from monads M on $\mathscr{C}$ to tuples $\langle A, a, f \rangle$

$$\Sigma A \xrightarrow{\ a\ } A \xleftarrow{\ f\ } MA$$

where $A \in \mathscr{C}$, and $a : \Sigma A \to A$ is a $\dot{\Sigma}$-algebra, and $f' : MA \to A$ is an Eilenberg-Moore algebra of M. The idea is similar to our modular models: for whatever 'ambient effect' M, there is a model A of the effect $\dot{\Sigma}$, which also models any effect that M supports. In particular, if M has an algebraic operation $b : S \circ M \to M$ on M, then the object A can 'forward' this operation by

$$SA \xrightarrow{\ S\eta_A\ } S(MA) \xrightarrow{\ b\ } MA \xrightarrow{\ f\ } A.$$

Modular handlers in this sense are also an instance of liftings along fibrations. Let $|\textsc{Mnd}(\mathscr{C})|$ be the *discrete* category of monads over $\mathscr{C}$, then the identity functor $\mathrm{Id} : |\textsc{Mnd}(\mathscr{C})| \to |\textsc{Mnd}(\mathscr{C})|$ is a fibration. For every $\dot{\Sigma} \in \textsc{Eqs}(\mathscr{C})$, we define the functor $T_{\dot{\Sigma}} : |\textsc{Mnd}(\mathscr{C})| \to \textsc{Eqs}(\mathscr{C})$ to be $(-)\text{-}\textsc{Act} + \dot{\Sigma}$, where $(-)\text{-}\textsc{Act}$ is the equational system of monad algebras from 2.2.1∗8. Let $P : \textsc{Alg}(\mathscr{C}) \to \textsc{Eqs}(\mathscr{C})$ be the Grothendieck construction of $(-)\text{-}\textsc{Alg} : \textsc{Eqs}(\mathscr{C})^{\mathrm{op}} \to \mathrm{CAT}$. A modular handler H of $\dot{\Sigma}$ is then a lifting of $T_{\dot{\Sigma}}$ along Id and P:

$$\begin{array}{ccc} |\textsc{Mnd}(\mathscr{C})| & \xrightarrow{\ H\ } & \textsc{Alg}(\mathscr{C}) \\ {\scriptstyle \mathrm{Id}}\downarrow & & \downarrow{\scriptstyle P} \\ |\textsc{Mnd}(\mathscr{C})| & \xrightarrow[\ T_{\dot{\Sigma}}\ ]{} & \textsc{Eqs}(\mathscr{C}) \end{array}$$

In fact, most examples of modular handlers from Schrijvers et al. [2019] and Yang and Wu [2021], apart from those based on *continuation monad transformers* $M \mapsto (- \Rightarrow MR) \Rightarrow MR$, are covariant functors $\textsc{Mnd}(\mathscr{C}) \to \textsc{Alg}(\mathscr{C})$.

3.2∗4 (Output Effects). An intuition for effect handlers is that they *consume* effectful operations, but in almost all implementations of effect handlers, handlers can also *produce* effectful operations. For example, a handler may handle an exception by producing a nondeterministic failure, thus transforming the effect of exceptions to the effect of nondeterminism. Such handlers with both *input effect*

(the operations to be handled) and *output effect* (the operations to be generated) can be modelled as liftings along fibrations as well.

Let $\mathcal{F}$ be an operation family (Definition 2.5∗2) closed under binary coproducts. For every $\ddot{\Psi} \in \mathcal{F}$, we denote by $\mathrm{U}_{\ddot{\Psi}} : \ddot{\Psi}/\mathcal{F} \to \mathcal{F}$ the forgetful functor from the coslice category under $\ddot{\Psi} \in \mathcal{F}$ to $\mathcal{F}$. We also denote by $P_{\ddot{\Psi}} : (\ddot{\Psi}/\mathcal{F})\text{-}\textsc{Alg} \to \ddot{\Psi}/\mathcal{F}$ the change of base of the fibration $P : \mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}$ along the functor $\mathrm{U}_{\ddot{\Psi}}$:

$$\begin{array}{ccc} (\ddot{\Psi}/\mathcal{F})\text{-}\textsc{Alg} & \longrightarrow & \mathcal{F}\text{-}\textsc{Alg} \\ {\scriptstyle P_{\ddot{\Psi}}}\downarrow & \lrcorner & \downarrow{\scriptstyle P} \\ \ddot{\Psi}/\mathcal{F} & \xrightarrow[\mathrm{U}_{\ddot{\Psi}}]{} & \mathcal{F} \end{array}$$

For every pair of $\ddot{\Sigma}, \ddot{\Psi} \in \mathcal{F}$, we define a functor $T_{\ddot{\Sigma},\ddot{\Psi}} : \ddot{\Sigma}/\mathcal{F} \to \ddot{\Psi}/\mathcal{F}$

$$T_{\ddot{\Sigma},\ddot{\Psi}}\ \langle \ddot{\Phi} \in \mathcal{F}, S : \ddot{\Sigma} \to \ddot{\Phi} \rangle = \langle \ddot{\Phi} + \ddot{\Psi}, \iota_2 : \ddot{\Psi} \to \ddot{\Phi} + \ddot{\Psi} \rangle$$

A modular model M of input effect $\ddot{\Psi}$ and output effect $\ddot{\Sigma}$ is then a lifting of the functor $T_{\ddot{\Sigma},\ddot{\Psi}}$ along the fibrations $P_{\ddot{\Sigma}}$ and $P_{\ddot{\Psi}}$:

$$\begin{array}{ccc} (\ddot{\Sigma}/\mathcal{F})\text{-}\textsc{Alg} & \xrightarrow{M} & (\ddot{\Psi}/\mathcal{F})\text{-}\textsc{Alg} \\ {\scriptstyle P_{\ddot{\Sigma}}}\downarrow & & \downarrow{\scriptstyle P_{\ddot{\Psi}}} \\ \ddot{\Sigma}/\mathcal{F} & \xrightarrow[T_{\ddot{\Sigma},\ddot{\Psi}}]{} & \ddot{\Psi}/\mathcal{F} \end{array} \tag{3.9}$$

Note that it is $\ddot{\Psi}$ rather than $\ddot{\Sigma}$ that is the *input* effect, since $\ddot{\Psi}$ is the effect to be 'handled' by this modular model; this is similar to how translations of equational systems $\dot{\Psi} \to \dot{\Sigma}$ are functors $\dot{\Sigma}\text{-}\textsc{Alg} \to \dot{\Psi}\text{-}\textsc{Alg}$ from the opposite direction.

3.2∗5. In the diagram (3.9), $P_{\ddot{\Psi}}$ is a pullback of $P : \mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}$ along $\mathrm{U}_{\ddot{\Psi}}$, so functors M making (3.9) commute are in bijection with functors N making

$$\begin{array}{ccccc} (\ddot{\Sigma}/\mathcal{F})\text{-}\textsc{Alg} & & \xrightarrow{\quad N \quad} & & \mathcal{F}\text{-}\textsc{Alg} \\ {\scriptstyle P_{\ddot{\Sigma}}}\downarrow & & & & \downarrow{\scriptstyle P} \\ \ddot{\Sigma}/\mathcal{F} & \xrightarrow[T_{\ddot{\Sigma},\ddot{\Psi}}]{} & \ddot{\Psi}/\mathcal{F} & \xrightarrow[\mathrm{U}_{\ddot{\Psi}}]{} & \mathcal{F} \end{array}$$

commute. When $\ddot{\Sigma}$ is the theory of monoids with no operations, this recovers our earlier definition of modular models without output effects (3.8).

3.2∗6. Moreover, the fibration $\mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}$ in 3.2∗4 can be replaced by many other fibrations whose base category is a category of some notion of algebraic theories and total category is a category of pairs of a theory and its model. For example, let $\textsc{FPCat}$ be the category of small categories with finite products and finite-product-preserving functors; there is a functor $[-, \textsc{Set}] : \textsc{FPCat}^{\mathrm{op}} \to \mathrm{CAT}$

sending every $\mathscr{C} \in \text{FPC}\textsc{at}$ to the category of finite-product-preserving functors $\mathscr{C} \to \textsc{Set}$, which induces a fibration $P : \text{FPM}\textsc{od} \to \text{FPC}\textsc{at}$, and we can talk about modular models of finite-product theories by replacing $\mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}$ in 3.2∗4 with this fibration. The same thing can be said for *generalised algebraic theories* [Cartmell 1986], *second-order algebraic theories* [Fiore and Mahmoud 2010], and so on for any framework of algebraic theories that have a fibration of models over theories and useful ways of combining theories such as coproducts.

3.2∗7. Motivated by the above examples of liftings, we will use 'model transformers' as a suggestive synonym for liftings along fibrations. We could similarly call the functor $T : \mathscr{T} \to \mathscr{T}'$ a 'theory transformer', but we will just call it a functor as it is shorter. Incidentally, Hyland et al. [2006] have a concept of 'operation transformers', but they correspond to translations between algebraic theories rather than functors between the categories of theories.

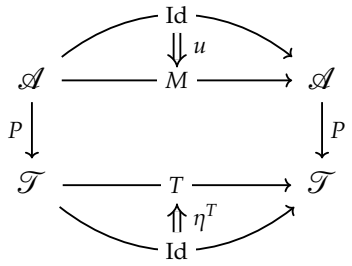
3.2∗8 Definition. Given two fibrations $P : \mathscr{A} \to \mathscr{T}$, $P' : \mathscr{A}' \to \mathscr{T}'$ and a functor $T : \mathscr{T} \to \mathscr{T}'$, a (strong/strict) *model transformer* M of T is defined to a (fibred/split) lifting of T along P and P' (Definition 3.1∗15).

$$\begin{array}{ccc} \mathscr{A} & \xrightarrow{M} & \mathscr{A}' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} \\ \mathscr{T} & \xrightarrow[T]{} & \mathscr{T}' \end{array}$$

The category $\textsc{Lift}_{P,P'}(T)$ of liftings will also be denoted by $\textsc{Motr}(T)$.

Specially, model transformers of a coproduct functor $- + \Psi : \mathscr{T} \to \mathscr{T}$ will be called *modular models of* $\Psi \in \mathscr{T}$, reusing the terminology from the last section.

3.2∗9. We can similarly generalise *updaters* (Definition 3.1∗22) to general fibrations. However, it seems more natural to require the two fibrations to be the same in this case. Therefore, given a fibration $P : \mathscr{A} \to \mathscr{T}$ and a pointed endofunctor $\langle T : \mathscr{T} \to \mathscr{T}, \eta^T : \text{Id} \to \mathscr{T} \rangle$, an *updater* for a model transformer of T along P is a natural transformation $u : \text{Id} \to M$ such that $P \circ u = \eta^T \circ P$:



We denote by $\textsc{Motr}_u(T)$ the category of model transformers of T equipped with an updater and whose morphisms $\langle M, u \rangle \to \langle N, v \rangle$ are morphisms $\sigma : M \to N$ that are compatible with the updaters: $v = \sigma \cdot u : \text{Id} \to N$.

3.3 Constructions of Model Transformers

3.3∗1. The definition of model transformers is just a mathematical formulation of semantic modularity, so in this section we will have a look at some concrete examples and general constructions of model transformers. Each subsection below discusses a construction and is basically independent of each other:

- ∗ initial model transformers (Section 3.3.1),
- ∗ initial model transformers with an updater (Section 3.3.2),
- ∗ free model transformers (Section 3.3.3),
- ∗ modular models from monoids transformers (Section 3.3.4),
- ∗ limits and colimits of model transformers (Section 3.3.5),
- ∗ modular models in symmetric monoidal categories (Section 3.3.7).

3.3.1 Initial Model Transformer

3.3.1∗1. We begin with the *initial model transformer*. Let $P : \mathscr{A} \to \mathscr{T}$ be a fibration and $P' : \mathscr{A}' \to \mathscr{T}'$ be a fibration with a cleavage κ such that for every object $\Sigma \in \mathscr{T}'$, the fiber category $\mathscr{A}'_\Sigma$ has a chosen initial object $\mu\Sigma$. Then for every functor $T : \mathscr{T} \to \mathscr{T}'$, we define a functor $0_T : \mathscr{A} \to \mathscr{A}'$ such that

- ∗ for every object $A \in \mathscr{A}$, $0_T A = \mu(TPA)$,
- ∗ for every morphism $f : A \to B \in \mathscr{A}$, $0_T f = \kappa(\mu(TPB), TPf) \cdot v$ as follows, where $v : \mu(TPA) \to X$ is the unique morphism out of the initial object $\mu(TPA)$ of the fiber category $\mathscr{A}'_{TPA}$:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \end{array} \qquad \begin{array}{ccc} \mu(TPA) & & \\ \downarrow{\scriptstyle v} & & \\ X & \xrightarrow{\kappa(\mu(TPB),TPf)} & \mu(TPB) \end{array}$$

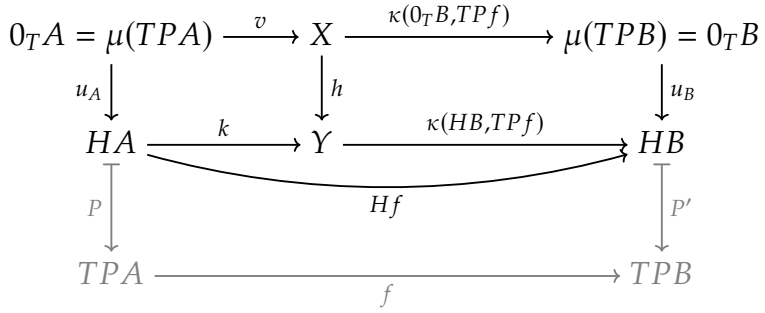
$$\begin{array}{ccc} PA & \xrightarrow[Pf]{} & PB \end{array} \qquad \begin{array}{ccc} TPA & \xrightarrow[TPf]{} & TPB \end{array}$$

3.3.1∗2 Theorem. *In the situation of* 3.3.1∗1*, the functor* $0_T : \mathscr{A} \to \mathscr{A}'$ *is a model transformer of* $T : \mathscr{T} \to \mathscr{T}'$ *and is initial in the category* $\textsc{Motr}(T)$.

$$\begin{array}{ccc} \mathscr{A} & \xrightarrow{0_T} & \mathscr{A}' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} \\ \mathscr{T} & \xrightarrow[T]{} & \mathscr{T}' \end{array}$$

Moreover, when the cleavage κ of P' preserves the chosen initial objects up to isomorphism (resp. strictly), i.e. for all $f : \Gamma \to \Sigma \in \mathscr{T}'$, the domain of $\kappa(\mu\Sigma, f)$ is initial in $\mathscr{A}'_\Gamma$ (resp. exactly $\mu\Gamma$), 0_T is a strong (resp. strict) model transformer.

Proof. The functor 0_T satisfies $P' \circ 0_T = T \circ P$ by construction, so it is a model transformer by definition (Definition 3.2∗8). Given any $H : \mathscr{A} \to \mathscr{A}'$ such that $P' \circ H = T \circ P$, for every $A \in \mathscr{A}$, HA and $0_T A$ are both in the fibre category $\mathscr{A}'_{TPA}$, so by the initiality of $0_T A$, there is a *unique* vertical morphism $u_A : 0_T A \to HA$. To show that u_A is natural, consider every $f : A \to B$ and the morphism $\kappa(HB, TPf) : Y \to HB$, we have the following situation



where $v : 0_T A \to X$ is the unique vertical morphism from $\mu(TPA)$ to X, and $h : X \to Y$ is the unique vertical morphism making the upper-right square commute, obtained from the cartesianess of $\kappa(HB, TPf) : Y \to HB$. Similarly, k is the unique vertical morphism satisfying $\kappa(HB, TPf) \cdot k = Hf$. Note that $0_T f$ is exactly the upper path $\kappa(0_T B, TPf) \cdot v$, and the upper-left square commutes by the initiality of $0_T A$. Hence we have the commutativity of the large rectangle, which is the naturality of $u : 0_T \to H$. This concludes the proof of the initiality of the model transformer 0_T in $\textsc{Motr}(T)$.

For the second part of the theorem, if the cleavage κ preserves initial algebras up to isomorphism, the object X in the diagram above is also initial among $\mathscr{A}'_{TPA}$, so the morphism $v : 0_T A \to X$ is a vertical isomorphism, and $0_T f = \kappa(0_T B, TPf) \cdot v$ is also cartesian. The case for splitting fibrations is similar. □

3.3.1∗3 Example. Let $\mathcal{F}$ be an operation family of monoids such that $\ddot{\Sigma}$-Alg has chosen initial algebras for all $\ddot{\Sigma} \in \mathcal{F}$. Applying the construction of initial model transformers to the situation of 3.1∗16, where $T = - + \ddot{\Psi}$ for some $\ddot{\Psi} \in \mathcal{F}$, the model transformer 0_T then maps every $\langle \ddot{\Sigma}, A, \alpha \rangle$ to the initial algebra of $\ddot{\Sigma} + \ddot{\Psi}$, ignoring the 'existing model' $\langle A, \alpha \rangle$ of the 'existing syntax' $\ddot{\Sigma}$ completely. Therefore the initial model transformer does not have an updater (Definition 3.1∗22) in general.

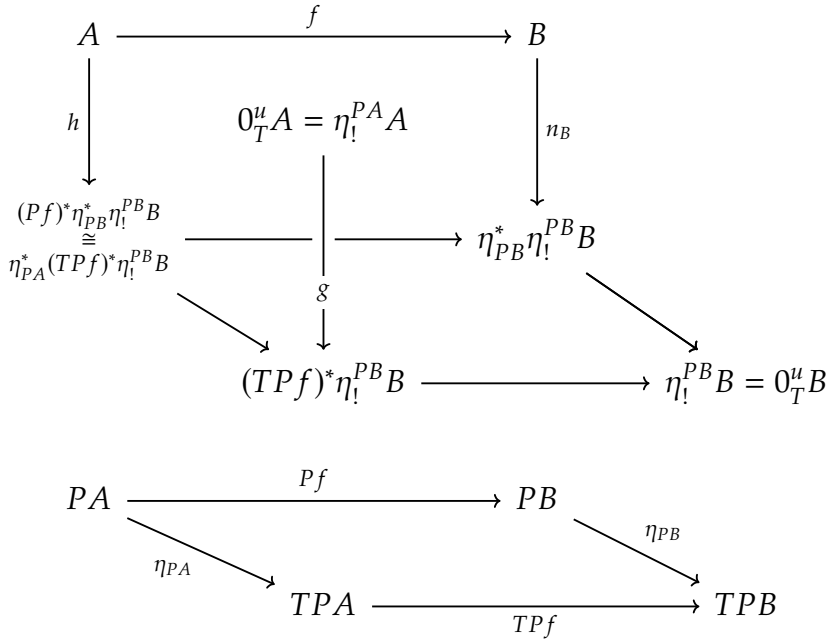
3.3.2 Initial Updatable Model Transformers

3.3.2∗1. Instead of completely ignoring the existing model, we can alternatively consider the *free model* of the new syntax over the existing model. As a special

case, let $\mathcal{F}$ be an operation family closed under coproducts, for every $\ddot{\Sigma}, \ddot{\Psi} \in \mathcal{F}$, Theorem 2.2.2∗14 provides a sufficient condition for the forgetful functor $U : (\ddot{\Sigma} + \ddot{\Psi})\text{-Alg} \to \ddot{\Sigma}\text{-Alg}$ to have a left adjoint $F : \ddot{\Sigma}\text{-Alg} \to (\ddot{\Sigma} + \ddot{\Psi})\text{-Alg}$, the idea is to define a model transformer of $\ddot{\Psi} \in \mathcal{F}$ by sending every $\langle \ddot{\Sigma}, A, \alpha \rangle \in \mathcal{F}\text{-Alg}$ to $F\langle A, \alpha \rangle$. Moreover, the unit of the adjunction $F \dashv U$ defines an updater. The universal property of the obtained model transformer is that it is the initial one in the category of model transformers with an updater (3.2∗9), so we will call it the *initial updatable model transformer* for short.

3.3.2∗2 Theorem. *Let $P : \mathscr{A} \to \mathscr{T}$ be a fibration with a cleavage such that for every morphism $t : \Gamma \to \Sigma$ in $\mathscr{T}$, there is an adjunction $t_! \dashv t^* : \mathscr{A}_\Sigma \to \mathscr{A}_\Gamma$. Let $\langle T, \eta \rangle$ be a pointed endofunctor on $\mathscr{T}$. The category of* $\text{Motr}_u(T)$ *of model transformers of T with an updater as defined in* 3.2∗9 *has an initial object.*

Proof. We define a model transformer $0^u_T : \mathscr{A} \to \mathscr{A}$ as follows. For every object $A \in \mathscr{A}$, we have a functor $\eta^{PA}_! : \mathscr{A}_{PA} \to \mathscr{A}_{TPA}$ left adjoint to $\eta^*_{PA} : \mathscr{A}_{TPA} \to \mathscr{A}_{PA}$, and we define $0^u_T A := \eta^{PA}_! A$. For every $f : A \to B \in \mathscr{A}$, let $n_B : B \to \eta^*_{PB}\eta^{PB}_! B$ be the unit of the adjunction $\eta^{PB}_! \dashv \eta^*_{PB}$. Since n_B is a morphism in $\mathscr{A}_{PB}$, it is vertical: $Pn_B = id_{PB}$, and $P(n_B \cdot f) = Pf$. Therefore there is a unique $h : A \to (Pf)^*\eta^*_{PB}\eta^{PB}_! B$ making the upper square commute:



(3.10)

The unlabelled morphisms are the evident cartesian morphisms from the cleavage κ. By the naturality of $\eta : \text{Id} \to T$, we have the commutativity of the bottom square in $\mathscr{T}$: $\eta_{PB} \cdot Pf = TPf \cdot \eta_{PA}$. Reindexing is pseudofunctorial, so we have

$$(Pf)^*\eta^*_{PB}\eta^{PB}_! B \quad \cong \quad \eta^*_{PA}(TPf)^*\eta^{PB}_! B.$$

Hence the morphism h determines a morphism $A \to \eta^*_{PA}(TPf)^*\eta^{PB}_! B$, which by

the adjunction $\eta^{PA}_! \dashv \eta^*_{PA}$, determines a vertical morphism

$$g : 0^u_T A = \eta^{PA}_! A \to (TPf)^* \eta^{PB}_! B.$$

By composing g with $\kappa(0^u_T B, TPf) : (TPf)^* \eta^{PB}_! B \to 0^u_T B$, we obtain a morphism $0^u_T A \to 0^u_T B$, which is our definition of the action of 0^u_T the morphism $f : A \to B$. We omit the checking of the functoriality of $0^u_T : \mathscr{A} \to \mathscr{A}$ here.

The functor 0^u_T by construction is a lifting of $T : \mathscr{T} \to \mathscr{T}$ along the fibration $P : \mathscr{A} \to \mathscr{T}$. It has an updater $u : \mathrm{Id} \to 0^u_T$ given by

$$u_B := \kappa(\eta_{PB}, \eta^{PB}_! B) \cdot n_B : B \to \eta^{PB}_! B = 0^u_T B$$

for all $B \in \mathscr{A}$ as in the diagram (3.10) above. To see the naturality of u, for all $f : A \to B$, the two morphisms $u_B \cdot f$ and $0^u_T f \cdot u_A$ are over the same morphism in $\mathscr{T}$, i.e. $\eta_{PB} \cdot Pf = TPf \cdot \eta_{PA}$, so it is sufficient to show that their induced vertical morphisms in $\mathscr{A}_{PA}$ are equal. It can be calculated that the one corresponds to $u_B \cdot f$ is h as in (3.10) and the one corresponds to $0^u_T f \cdot u_A$ is

$$(\eta^*_{PA} g) \cdot n_A = \eta^*_{PA}(e_A \cdot \eta^{PA}_! h) \cdot n_A \tag{3.11}$$

where $e_A : \eta^{PA}_! \eta^*_{PA} A \to A$ is the counit of the adjunction $\eta^{PA}_! \dashv \eta^*_{PA}$. Using naturality and triangle identity of the unit n and counit e of the adjunction $\eta^{PA}_! \dashv \eta^*_{PA}$, the morphism (3.11) can be shown to be exactly h.

As for the initiality of $\langle 0^u_T, u\rangle$ in $\textsc{Motr}_u(T)$, for every $\langle M, v\rangle \in \textsc{Motr}_u(T)$ and $B \in \mathscr{A}$, $v_B : B \to MB$ is over η_{PB} by the definition of updaters. Hence there is a vertical morphism $w : B \to \eta^{PB}_! MB$ such that $v_B = \kappa(\eta_{PB}, MB) \cdot w$. By the universal property of $n_B : B \to \eta^*_{PB} \eta^{PB}_! B$, there is a *unique* morphism $\sigma_B : \eta^{PB}_! B = 0^u_T B \to MB$ such that $\eta^*_{PB} \sigma_B \cdot n_B = w$:

$$\begin{array}{ccc} B & \xrightarrow{n_B} & \eta^*_{PB}\eta^{PB}_! B \\ & \searrow^{w} & \downarrow{\scriptstyle \eta^*_{PB}\sigma_B} \\ & & \eta^*_{PB} MB \end{array}$$

We omit the checking of the naturality of σ here (it is similarly to the proof of Theorem 3.3.1∗2). Since every $\tau : \langle 0^u_T, u\rangle \to \langle M, v\rangle$ is a vertical natural transformation $\tau : 0^u_T \to M$ satisfying $\tau_B \cdot u_B = v_B$, it must satisfy that $\eta^*_{PB} \tau_B \cdot n_B = w$. Therefore, σ is the unique morphism $\langle 0^u_T, u\rangle \to \langle M, v\rangle$ in $\textsc{Motr}_u(T)$. □

3.3.2∗3 Example. Let $\mathscr{C}$ be a category and $\mathscr{A}$ be a *freeness condition* for $\mathscr{C}$ (Definition 2.2.1∗22) that satisfies the assumption of Theorem 2.2.2∗14 (for example, preserving colimits of α-chains for some limit ordinal α and $\mathscr{C}$ being cocomplete). The fibration $P : \textsc{Alg}_{\mathscr{A}}(\mathscr{C}) \to \textsc{Eqs}_{\mathscr{A}}(\mathscr{C})$ then satisfies the condition of Theorem 3.3.2∗2, so the pointed functor $(- + \dot{\Psi}) : \textsc{Eqs}_{\mathscr{A}}(\mathscr{C}) \to \textsc{Eqs}_{\mathscr{A}}(\mathscr{C})$ for every

$\dot\Psi \in \mathrm{EQS}_{\mathscr{A}}(\mathscr{C})$ then has an initial updatable model transformer.

3.3.2∗4 Example. Let $\mathscr{C}$ be a cocomplete monoidal category such that the monoidal product $\Box : \mathscr{C} \times \mathscr{C} \to \mathscr{C}$ preserves colimits of α-chains for some limit ordinal α. Let $\mathcal{F} \subseteq \mathrm{MON}/\mathrm{EQS}(\mathscr{C})$ be the operation family containing all equational systems whose signature and context colimits of α-chains for some limit ordinal α. By Theorem 2.2.2∗14, for every morphism $T : \ddot\Sigma \to \ddot\Psi \in \mathcal{F}$, the corresponding functor $T : \ddot\Psi\text{-}\mathrm{ALG} \to \ddot\Sigma\text{-}\mathrm{ALG}$ has a left adjoint. Therefore the fibration $P : \mathcal{F}\text{-}\mathrm{ALG} \to \mathcal{F}$ satisfies the condition of Theorem 3.3.2∗2, and for every $\ddot\Psi \in \mathcal{F}$, the functor $- + \ddot\Psi : \mathcal{F} \to \mathcal{F}$ has an initial updater model transformer $0^{\mu}_{-+\ddot\Psi}$, which maps every $\ddot\Sigma$-algebra A to the relative free $(\ddot\Sigma + \ddot\Psi)$-algebra over A.

3.3.2∗5 Example. For a concrete example, let us instantiate $\mathscr{C}$ in the previous example to be $\langle \mathrm{SET}^{\mathrm{FIN}}, \bullet, V \rangle$ from 2.1.2∗2. As we mentioned in 2.5∗21, the syntax of untyped λ-calculus can be presented as an equational system $\Lambda = \Sigma_{\langle O,a\rangle}\text{-}\mathrm{MON}$ for the binding signature $O = \{app, abs\}$ with $a(abs) = \langle 1\rangle$ and $a(app) = \langle 0, 0\rangle$. Models of Λ can be obtained from *reflexive objects* $U \cong U^U$ in any cartesian closed category C: every U induces a functor $\bar U : \mathrm{FIN} \to \mathrm{SET}$ with $n \mapsto \mathscr{C}(U^n, U)$. The functor $\bar U$ has a monoid structure $[\eta_U, \mu_U]$ (similar to that of the continuation monad), and it is a model of Λ [Hyland 2017]:

$$abs_U : \bar U^V n \cong \mathscr{C}(U^{n+1}, U) \cong \mathscr{C}(U^n, U^U) \cong \mathscr{C}(U^n, U) \cong \bar U n$$
$$app_U : (\bar U \times \bar U)n \cong \mathscr{C}(U^n, U \times U) \cong \mathscr{C}(U^n, U^U \times U) \xrightarrow{eval_{U^U} \cdot -} \mathscr{C}(U^n, U) \cong \bar U n$$

Now consider the theory ST_S of mutable state (Example 2.4∗9) for some finite set S. Its initial updatable model transformer maps the Λ-model on $\bar U$ to a $(\Lambda +_{\mathrm{MON}} \mathrm{ST}_S)$-model whose carrier is the initial algebra

$$\mu X.\, \bar U + X \bullet X + V + X^V + X \times X + \textstyle\prod_S X + \coprod_S X : \mathrm{FIN} \to \mathrm{SET}$$

quotiented by equations of Λ and ST_S, as well as equations saying that the Λ operations of the initial algebra acting on the first component $\bar U$ is the same as the model $[\eta_U, \mu_U, abs_U, app_U]$ of $\bar U$.

3.3.2∗6. Left adjoints to reindexing functors of a fibration are used for modelling Σ-types and $\exists$-quantification in categorical logic, where the fibration models types or predicates over a type, and reindexing models substitution. In this context, reindexing functors typically preserve the left adjoints suitably, called satisfying the *Beck-Chevalley condition* [Jacobs 1999, Definition 1.9.4], reflecting the fact substitution commutes with Σ-types and $\exists$-quantifiers. Dually, in this context reindexing functors usually have right adjoints, which model Π or $\forall$.

However, for fibrations of algebras and theories, the reindexing functors typically do *not* have right adjoints (the translation functors between categories

of algebras almost never preserve colimits), and reindexing functors typically do *not* preserve the left adjoints (relative free algebras), so initial updatable model transformers typically are not strong or strict. For example, consider $\mathscr{E} = \langle \textsc{Set}, \times, 1 \rangle$. We have the following theories in the family $\textsc{Mon}/\textsc{Eqs}_{\mathscr{A}}(\mathscr{E})$: Mon with the identity translation; Grp with the inclusion translation $T : \textsc{Mon} \to \textsc{Grp}$ (Example 2.2.2∗4); and the theory BLat of bounded lattices with the translation that maps monoid multiplication to lattice join $\vee$, monoid identity to lattice bottom $\bot$. The following diagram in CAT does not commute:

$$\begin{array}{ccc} \textsc{Mon}\text{-}\textsc{Alg} & \xleftarrow{\quad T \quad} & \textsc{Grp}\text{-}\textsc{Alg} \\ {\scriptstyle \mathrm{F}_{\textsc{Mon}}}\downarrow & & \downarrow{\scriptstyle \mathrm{F}_{\textsc{Grp}}} \\ (\textsc{Mon} + \textsc{BLat})\text{-}\textsc{Alg} & \xleftarrow{\quad T+\textsc{BLat} \quad} & (\textsc{Grp} + \textsc{BLat})\text{-}\textsc{Alg} \end{array}$$

The theory Mon + BLat, bounded lattices whose join $\vee$ and $\bot$ form a monoid, is isomorphic to BLat since $\langle \vee, \bot \rangle$ of a lattice is already a monoid, so $\mathrm{F}_{\textsc{Mon}}(TG)$ for every group G is the free bounded lattice over G as a monoid. On the other hand, the theory Grp + BLat, bounded lattices whose $\langle \vee, \bot \rangle$ form a group, has only trivial models since for every element x, $\bot = x \vee x^{-1}$, and by the idempotent law of join, $\bot = (x \vee x) \vee x^{-1} = x \vee (x \vee x^{-1}) = x \vee \bot = x$. Therefore $(T + \textsc{BLat})(\mathrm{F}_{\textsc{Grp}} G)$ is always the trivial bounded lattice for every group G.

3.3.3 Free Model Transformers over Ordinary Models

3.3.3∗1. Consider model transformers of $- + \ddot{\Psi} : \mathcal{F} \to \mathcal{F}$ on an operation family $\mathcal{F}$: for every $\ddot{\Sigma} \in \mathcal{F}$ and every $A \in \ddot{\Sigma}\text{-}\textsc{Alg}$, the initial model transformer $0_{\ddot{\Psi}}$ just ignores the algebra A and freely generates a model of $\ddot{\Sigma} + \ddot{\Psi}$, whereas the initial updatable model transformer $0^{u}_{\ddot{\Psi}}$ takes into account of A and freely generates a model of $\ddot{\Sigma} + \ddot{\Psi}$ that has a $\ddot{\Sigma}$-homomorphism from A.

The natural next step is then freely generating a model of $\ddot{\Sigma} + \ddot{\Psi}$ that has both a $\ddot{\Sigma}$-homomorphism from A and a $\ddot{\Psi}$-homomorphism from some fixed $B \in \ddot{\Psi}\text{-}\textsc{Alg}$, using the construction in Example 2.2.2∗18. In this way, we can turn an ordinary model B of $\ddot{\Psi}$ to a model transformer of $\ddot{\Psi}$, and it is also going to be the *free* way.

3.3.3∗2. In this subsection, we fix a fibration $P : \mathscr{A} \to \mathscr{T}$ with a cleavage such that $\mathscr{T}$ has finite coproducts, and we fix a functor $\oplus : \mathscr{T} \times \mathscr{T} \to \mathscr{T}$ equipped with a natural transformation $\tau : + \to \oplus$. For all $\Sigma, \Gamma \in \mathscr{T}$, we define

$$\kappa_1 := (\Sigma \xrightarrow{\iota_1} \Sigma + \Gamma \xrightarrow{\tau} \Sigma \oplus \Gamma) \qquad \kappa_2 := (\Gamma \xrightarrow{\iota_2} \Sigma + \Gamma \xrightarrow{\tau} \Sigma \oplus \Gamma)$$

The natural transformation $\kappa_1 : - \to - \oplus \Gamma$ makes $- \oplus \Gamma$ a pointed functor, enabling us to talk about updaters (3.2∗9) of the functor $- \oplus \Gamma$.

As usual, the category $\mathscr{T}$ is expected to be a category of some notion of

algebraic theories and each fiber category $\mathscr{A}_\Sigma$ is the category of models of $\Sigma \in \mathscr{T}$. For example, P can be the fibration $\mathcal{F}$-Alg $\to \mathcal{F}$ for an operation family and $\oplus$ can be just the coproduct or the commutative combination (3.2∗2).

3.3.3∗3 Definition. When the fiber category $\mathscr{A}_0$ of the initial object $0 \in \mathscr{T}$ also has an initial object I, for every $\Gamma \in \mathscr{A}$ we define a functor $\mathrm{U}_{\oplus\Gamma} : \textsc{Motr}(- \oplus \Gamma) \to \mathscr{A}_\Gamma$:

$$\mathrm{U}_{\oplus\Gamma}\, M := \kappa_2^* MI$$

where κ_2^* is the reindexing functor $\mathscr{A}_{0\oplus\Gamma} \to \mathscr{A}_\Gamma$. The functor $\mathrm{U}_{\oplus\Gamma}$ is intuitively the forgetful functor from model transformers of ($\oplus$-combination with) Γ to ordinary models of Γ. Composing $\mathrm{U}_{\oplus\Gamma}$ with the functor that forgets updaters (3.2∗9), we also have a functor $\textsc{Motr}_u(- \oplus \Gamma) \to \mathscr{A}_\Gamma$ that we shall also denote by $\mathrm{U}_{\oplus\Gamma}$.

3.3.3∗4 Theorem. *Assume that all reindexing functors $!^* : \mathscr{A}_\Gamma \to \mathscr{A}_0$ to the fiber category over the initial object $0 \in \mathscr{T}$ is monadic and every fiber category $\mathscr{A}_\Sigma$ is finitely cocomplete. The functor $\mathrm{U}_{\oplus\Gamma} : \textsc{Motr}_u(- \oplus \Gamma) \to \mathscr{A}_\Gamma$ for every $\Gamma \in \mathscr{T}$ defined in Definition 3.3.3∗3 has a left adjoint $\mathrm{F}_{\oplus\Gamma} : \mathscr{A}_\Gamma \to \textsc{Motr}_u(- \oplus \Gamma)$.*

Proof. Every fibration is equivalent to a split one [Jacobs 1999, Corollary 5.2.5], and the statement is stable under equivalence of fibrations, so we can assume without loss of generality that P is a split fibration. By Mac Lane [1998, §IV.1 Theorem 2], it is sufficient to construct for every $\langle B, \beta\rangle \in \mathscr{A}_\Gamma$ a model transformer with an updater $M^B \in \textsc{Motr}_u(- \oplus \Gamma)$ and a universal arrow $e : \langle B, \beta\rangle \to \mathrm{U}_{\oplus\Gamma} M^B$.

First we observe that for every $\Sigma \in \mathscr{T}$, we have the following functors:

$$\begin{array}{ccc}
\mathscr{A}_\Sigma \times \mathscr{A}_\Gamma & \xleftarrow{\langle \kappa_1^*, \kappa_2^*\rangle} & \mathscr{A}_{\Sigma\oplus\Gamma} \\
\Big\downarrow{\scriptstyle !_\Sigma^* \times !_\Gamma^*} & & \Big\downarrow{\scriptstyle !_{\Sigma\oplus\Gamma}^*} \\
\mathscr{A}_0 \times \mathscr{A}_0 & \xleftarrow{\Delta} & \mathscr{A}_0
\end{array}$$

This diagram commutes strictly since $\kappa_1 \cdot !_\Sigma = !_{\Sigma\oplus\Gamma} = \kappa_2 \cdot !_\Gamma : 0 \to \Sigma \oplus \Gamma$, and we have assumed that P is a split fibration. By the assumption that each fiber category is finitely cocomplete, the category $\mathscr{A}_0$ has binary coproducts, so we have an adjunction $+ \dashv \Delta : \mathscr{A}_0 \to \mathscr{A}_0 \times \mathscr{A}_0$, and moreover the fiber category $\mathscr{A}_{\Sigma\oplus\Gamma}$ has coequalisers. By assumption $!_\Sigma^*$, $!_\Gamma^*$, and $!_{\Sigma\oplus\Gamma}^*$ are monadic functors. The product of monadic functors is also monadic, so $!_\Sigma^* \times !_\Gamma^*$ is monadic. By Borceux [1994b, Theorem 4.5.6] (c.f. our discussion in 2.2.2∗16), the functor $\langle \kappa_1^*, \kappa_2^*\rangle$ on the top of the diagram has a left adjoint $F_{\Sigma\oplus\Gamma} : \mathscr{A}_\Sigma \times \mathscr{A}_\Gamma \to \mathscr{A}_{\Sigma\oplus\Gamma}$.

Now we define M^B via an oplax transformation by Lemma 3.1∗18:

$$M^B : \mathscr{A}_- \to \mathscr{A}_{-\oplus\Gamma} : \mathscr{T}^{\mathrm{op}} \to \mathrm{CAT}.$$

We define the component M^B_Σ at every $\Sigma \in \mathscr{T}$ to be $F_{\Sigma\oplus\Gamma}\langle -, B\rangle : \mathscr{A}_\Sigma \to \mathscr{A}_{\Sigma\oplus\Gamma}$. For

every morphism $t : \Sigma \to \Phi$, we have the following functors:

$$\begin{array}{ccc} \mathscr{A}_\Phi \times \mathscr{A}_\Gamma & \underset{F_{\Phi\oplus\Gamma}}{\overset{\langle\kappa_1^*,\kappa_2^*\rangle}{\longleftarrow \top \longrightarrow}} & \mathscr{A}_{\Phi\oplus\Gamma} \\ {\scriptstyle t^*\times id}\downarrow & & \downarrow{\scriptstyle (t\oplus id)^*} \\ \mathscr{A}_\Sigma \times \mathscr{A}_\Gamma & \underset{F_{\Sigma\oplus\Gamma}}{\overset{\langle\kappa_1^*,\kappa_2^*\rangle}{\longleftarrow \top \longrightarrow}} & \mathscr{A}_{\Sigma\oplus\Gamma} \end{array}$$

By the naturality of κ, we have strict commutativity:

$$id : (t^* \times id) \circ \langle \kappa_1^*, \kappa_2^* \rangle = \langle t^* \kappa_1^*, \kappa_2^* \rangle = \langle \kappa_1^*, \kappa_2^* \rangle \circ (t \oplus id)^*,$$

which determines a canonical natural transformation

$$\tau : F_{\Sigma\oplus\Gamma} \circ (t^* \times id) \to (t \oplus id)^* \circ F_{\Phi\oplus\Gamma}$$

called the *mate* [nLab 2024b] or the *conjugate* of *id* [Mac Lane 1998, §IX.7]. Namely, τ is the transpose along the adjunction $F_{\Sigma\oplus\Gamma} \dashv \langle \kappa_1^*, \kappa_2^* \rangle$ of

$$(t^* \times id) \xrightarrow{\eta} \langle \kappa_1^*, \kappa_2^* \rangle \circ F_{\Sigma\oplus\Gamma} \circ (t^* \times id) \xrightarrow{id} \langle \kappa_1^*, \kappa_2^* \rangle \circ (t \oplus id)^* \circ F_{\Phi\oplus\Gamma}.$$

We define the 2-cell $M_t^B := (\tau \circ \langle \mathrm{Id}, \mathrm{K}_B \rangle) : M_\Sigma^B \circ t^* \to (t \oplus \Gamma)^* \circ M_\Phi^B$.

Then we define an updater $u : \mathrm{Id} \to M^B$ for M^B. For every $A \in \mathscr{A}$, letting $\Sigma := PA$, we define u_A to be

$$A \xrightarrow{\pi_1 \eta_{A,B}} \kappa_1^* F_{\Sigma\oplus\Gamma} \langle A, B \rangle \xrightarrow{\overline{\kappa_1}} M^B A$$

where $\eta_{A,B} : \langle A, B \rangle \to \langle \kappa_1^*, \kappa_2^* \rangle (F_{\Sigma\oplus\Gamma} \langle A, B \rangle)$ is the unit of the adjunction, $\overline{\kappa_1}$ is the cartesian morphism over κ_1 from the cleavage. The naturality of u is essentially a consequence of the naturality of η.

We have defined a model transformer M^B with an updater u for every $B \in \mathscr{A}_\Gamma$, and what remains is to define a universal arrow $e : B \to \mathrm{U}_{\oplus\Gamma} \langle M^B, u \rangle$. By Definition 3.3.3*3, $\mathrm{U}_{\oplus\Gamma} \langle M^B, u \rangle$ is $\kappa_2^* (F_{0\oplus\Gamma} \langle I, B \rangle) \in \mathscr{A}_\Gamma$. Therefore we define $e : B \to \mathrm{U}_{\oplus\Gamma} \langle M^B, u \rangle$ to be the second projection of the unit

$$\eta_{I,B} : \langle I, B \rangle \to \langle \kappa_1^*, \kappa_2^* \rangle (F_{0\oplus\Gamma} \langle I, B \rangle).$$

To show the universality of e, given any model transformer $\langle N, v \rangle \in \textsc{Mod}_u(- \oplus \Gamma)$ with a morphism $f : B \to \mathrm{U}_{\oplus\Gamma} \langle N, v \rangle$ in $\mathscr{A}_\Gamma$, we need to show that there is a unique $\sigma : \langle M^B, u \rangle \to \langle N, v \rangle$ in $\textsc{Mod}(- \oplus \Gamma)$ such that $(\mathrm{U}_{\oplus\Gamma} \sigma) \cdot e = f$.

First of all, the codomain of f is by definition $\kappa_2^*(NI)$, where I is the initial object of $\mathscr{A}_0$. It is not hard to see that the object I is also the initial object of the total category $\mathscr{A}$, so for every $A \in \mathscr{A}$, we have a morphism

$$\bar{f}_A := ((\kappa_2^* N!) \cdot f) : B \to \kappa_2^* NA.$$

Now recall that a morphism σ in $\textsc{Mod}_u(- \oplus \Gamma)$ is a vertical natural trans-

formation $M^B \to N$ that commutes with the updaters u and v. For every $A \in \mathscr{A}$, letting $\Sigma := PA$, the updater v at A is a morphism $v_A : A \to \kappa_1^*(NA)$. Paired with $\bar{f}_A$, we have a morphism $\langle v_A, \bar{f}_A \rangle : \langle A, B \rangle \to \langle \kappa_1^*, \kappa_2^* \rangle (NA)$, and we define $\sigma_A : M^B A = F_{\Sigma \oplus \Gamma} \langle A, B \rangle \to NA$ to be the transpose of $\langle v_A, \bar{f}_A \rangle$ along the adjunction $F_{\Sigma \oplus \Gamma} \dashv \langle \kappa_1^*, \kappa_2^* \rangle$. We omit the verification of naturality here.

What remains is to show that σ defined above is the unique morphism $\langle M^B, u \rangle \to \langle N, v \rangle$ satisfying $(\mathrm{U}_{\oplus\Gamma}\sigma) \cdot e = f$. First of all, σ satisfies this equation since by definition $\mathrm{U}_{\oplus\Gamma}\sigma = \kappa_2^* \sigma_I$ and σ_I makes the following triangle commute

$$\begin{array}{ccc} \langle I, B \rangle & \xrightarrow{\eta_{I,B}} & \langle \kappa_1^*, \kappa_2^* \rangle F_{I \oplus \Gamma} \langle I, B \rangle \\ & \searrow^{\langle V_I, \overline{f}_I \rangle} & \downarrow{\scriptstyle \langle \kappa_1^*, \kappa_2^* \rangle \sigma_I} \\ & & \langle \kappa_1^*, \kappa_2^* \rangle NI \end{array}$$

in the category $\mathscr{A}_0 \times \mathscr{A}_\Gamma$. The second projection of this commutativity diagram is exactly $(\mathrm{U}_{\oplus\Gamma}\sigma) \cdot e = f$. For the uniqueness of σ, given another τ satisfying $(\mathrm{U}_{\oplus\tau}\sigma) \cdot e = f$, for every $\Sigma \in \mathscr{T}$ and $A \in \mathscr{A}_\Sigma$, the naturality of τ for the unique morphism $! : I \to A$ implies the commutativity of the right trapezium below:

$$\begin{array}{ccccc} B & \xrightarrow{e} & \kappa_2^* F_{I \oplus \Gamma} \langle I, B \rangle & \xrightarrow{\kappa_2^* M^B !} & \kappa_2^* F_{\Sigma \oplus \Gamma} \langle A, B \rangle \\ & \searrow^{f} & \downarrow{\scriptstyle \kappa_2^* \tau_I} & & \downarrow{\scriptstyle \kappa_2^* \tau_A} \\ & & \kappa_2^* NI & \xrightarrow{\kappa_2^* N !} & \kappa_2^* NA \\ & \overline{f}_A \searrow & & & \nearrow \end{array} \tag{3.12}$$

Moreover, it can be shown that the following diagram in $\mathscr{A} \times \mathscr{A}$ commutes by expanding out the definition of the action of $M^B : \mathscr{A} \to \mathscr{A}$ on morphisms:

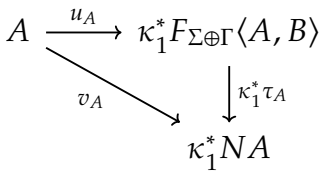
$$\begin{array}{ccc} \langle I, B \rangle & \xrightarrow{\eta_{I,B}} & \langle \kappa_1^*, \kappa_2^* \rangle F_{0 \oplus \Gamma} \langle I, B \rangle \\ {\scriptstyle \langle !, B \rangle}\downarrow & & \downarrow{\scriptstyle \langle \kappa_1^*, \kappa_2^* \rangle M^B !} \\ \langle A, B \rangle & \xrightarrow[\eta_{A,B}]{} & \langle \kappa_1^*, \kappa_2^* \rangle F_{\Sigma \oplus \Gamma} \langle A, B \rangle \end{array}$$

Applying the second projection to this commutative diagram,

$$\pi_2 \eta_{A,B} = \pi_2(\eta_{A,B} \cdot \langle !, B \rangle) = \pi_2(\langle \kappa_1^*, \kappa_2^* \rangle M^B ! \cdot \eta_{I,B}) = \kappa_2^* M^B ! \cdot \pi_2 \eta_{I,B}.$$

Recall that e is exactly $\pi_2 \eta_{I,B}$, so the top horizontal path of the diagram (3.12) is equal to $\pi_2 \eta_{A,B}$. Then the diagram (3.12) implies that $\kappa_2^* \tau_A \cdot \pi_2 \eta_{A,B} = \overline{f}_A$. This,

together with the fact τ is compatible with the updaters,



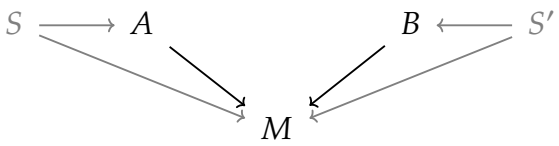
implies that τ_I is the transpose of $\langle v_A, \bar{f}_A\rangle$ along $F_{\Sigma\oplus\Gamma} \dashv \langle \kappa_1^*, \kappa_2^*\rangle$, so $\tau_I = \sigma_I$. □

3.3.3∗5 Example. Let $\mathscr{C}$ be a cocomplete category and $\mathscr{A}$ be the freeness condition (2.2.1∗22) containing all pairs of endofunctors that preserve colimits of α-chains for some limit ordinal α. By Theorem 2.2.1∗12 and Theorem 2.2.2∗14, the fibration $P : \textsc{Alg}_{\mathscr{A}}(\mathscr{C}) \to \textsc{Eqs}_{\mathscr{A}}(\mathscr{C})$ of algebras and equational systems in $\mathscr{A}$ then satisfies the assumptions of Theorem 3.3.3∗4. For every $\dot{\Psi} \in \textsc{Eqs}_{\mathscr{A}}(\mathscr{C})$, instantiating $\oplus$ to be $+ : \textsc{Eqs}_{\mathscr{A}}(\mathscr{C}) \times \textsc{Eqs}_{\mathscr{A}}(\mathscr{C}) \to \textsc{Eqs}_{\mathscr{A}}(\mathscr{C})$, Theorem 3.3.3∗4 constructs a modular model F_B of $\dot{\Psi}$ from an ordinary model B of $\dot{\Psi}$.

3.3.3∗6 Example. Similarly, let $\mathcal{F}$ and $\mathscr{E}$ be the operation family and monoidal category in Example 3.3.2∗4. Theorem 3.3.3∗4 lets us construct modular models of theories of monoids with operations from ordinary models.

In particular, if $\mathscr{E} = \langle \textsc{Endo}_\kappa(\mathscr{C}), \circ, \mathrm{Id}\rangle$ for some $\mathrm{l}\kappa\mathrm{p}\,\mathscr{C}$, and $\ddot{\Psi} \in \mathcal{F}$ be the theory of monads with some scoped operations, then Theorem 3.3.3∗4 lets us construct a modular model F_B of $\ddot{\Psi}\mathcal{F}$ from a monad B equipped with a $\ddot{\Psi}$-operation. For any theory $\ddot{\Sigma} \in \mathcal{F}$ of monads with operations, every monad A equipped with a $\ddot{\Sigma}$-operation is sent by the modular model F_B to a new monad C with monad morphisms $A \to C$ and $B \to C$ that are respectively a $\ddot{\Sigma}$-homomorphism and a $\ddot{\Psi}$-homomorphism.

3.3.3∗7 Example. Let $\mathscr{E}$ be a monoidal category, $\mathcal{F}$ be the operation family $\textsc{Alg}(\mathscr{E})$ of algebraic operations on $\mathscr{E}$-monoids, and $\ddot{\Psi} \in \mathcal{F}$. If the fibration $P : \mathcal{F}\text{-}\textsc{Alg} \to \mathcal{F}$ satisfies the assumption of Theorem 3.3.3∗4, the free modular model F_B of $\ddot{\Psi}$ over some $B \in \ddot{\Psi}\text{-}\textsc{Alg}$ has a simple characterisation – for every $\ddot{\Sigma} \in \textsc{Alg}(\mathscr{E})$ and $A \in \ddot{\Sigma}\text{-}\textsc{Alg}$, F_B simply maps A to the coproduct of B and A treated as monoids in $\mathscr{E}$. This is because an algebraic operation $S \circ M \to M$ on a monoid M is equivalently a morphism $S \to M$ (Lemma 2.4∗13), so the initial monoid with both $\ddot{\Sigma}$ and $\ddot{\Psi}$ operations together with a $\ddot{\Sigma}$-homomorphism from A and a $\ddot{\Psi}$-homomorphism from B is the same thing as the initial monoid with monoid morphisms from A and B, i.e. the coproduct of A and B as monoids:



3.3.4 Modular Models from Monoid Transformers

3.3.4∗1. The concept of modular models is directly inspired by Moggi's *monad transformers* and their generalisation, *monoid transformers* [Jaskelioff and Moggi 2010], to monoids in monoidal categories. A monoid transformer maps every monoid M in a monoidal category $\mathscr{E}$ to a monoid TM together with a monoid morphism $i : M \to TM$. The central question about monoid transformers is:

> If there is an operation on the monoid M, can this operation be transformed to an operation on TM?

The standard terminology here is to *lift* the operation to TM rather than to *transform* but we use the latter to avoid the confusion with liftings along fibrations.

This question was first formulated by Moggi [1989b, §4.1], accompanied by a basic result [Moggi 1989b, Proposition 4.1.3]: operations of the form $\alpha : A \to M$ for some fixed endofunctor A can always be transformed to $A \to TM$, namely $i \cdot \alpha : A \to TM$. About 20 years later, Jaskelioff and Moggi [2010] gave a new result: for *functorial monoid transformers* T on a *left-closed* monoidal category, every operation on M of the form $A \,\square\, M \to M$ can be lifted to $A \,\square\, TM \to TM$.

What we have done in this thesis is bringing *equations* on operations into the view and formulating transformations of operations as model transformers (liftings along fibrations). In this subsection, we put the old wine by Moggi [1989b] and Jaskelioff and Moggi [2010] in our new bottle.

3.3.4∗2. In this subsection, we fix a monoidal category $\mathscr{E}$ with finite coproducts that is right-distributive: $(\coprod_{i\in S} A_i) \,\square\, B \;\cong\; \coprod_{i\in S}(A_i \,\square\, B)$, which ensures that $\textsc{Alg}(\mathscr{E})$ and $\textsc{Scp}(\mathscr{E})$ from Section 2.5 are closed under coproducts.

3.3.4∗3 Theorem. *Let $\mathcal{F}$ be $\textsc{Alg}(\mathscr{E})$ and $\ddot{\Psi} = \langle \dot{\Psi}, T_\Psi \rangle \in \mathcal{F}$. Every functor $H : \textsc{Mon}(\mathscr{E}) \to \dot{\Psi}\text{-}\textsc{Alg}$ together with a natural transformation $\tau : \mathrm{Id} \to T_\Psi \circ H$*

$$\begin{array}{ccc} & \dot{\Psi}\text{-}\textsc{Alg} & \\ {\scriptstyle H}\nearrow & \Uparrow\tau & \searrow{\scriptstyle T_\Psi} \\ \textsc{Mon}(\mathscr{E}) & \xrightarrow[\mathrm{Id}]{} & \textsc{Mon}(\mathscr{E}) \end{array}$$

defines a strict modular model M of $\dot{\Psi} \in \mathcal{F}$ making the following commute:

$$\begin{array}{ccc} \mathcal{F}\text{-}\textsc{Alg} & \xrightarrow{\bar{M}} & (\mathcal{F} + \ddot{\Psi})\text{-}\textsc{Alg} \\ \downarrow & & \downarrow \\ \textsc{Mon}(\mathscr{E}) & \xrightarrow[H]{} & \dot{\Psi}\text{-}\textsc{Alg} \end{array} \tag{3.13}$$

where $\bar{M} : \mathcal{F}\text{-}\textsc{Alg} \to (\mathcal{F} + \dot{\Psi})\text{-}\textsc{Alg}$ is the functor corresponding to M by items 1 and 2

of Theorem 3.1∗19*, and the unlabelled vertical arrows are the evident projection functors. Moreover, M has an updater $u_{\langle\dot\Sigma,T_\Sigma,A,\alpha\rangle} = \tau_{\langle A,\, T_\Sigma\alpha\rangle}$.*

Proof. For every object $\langle\dot\Sigma, T_\Sigma, A, \alpha\rangle$ of $\mathcal{F}$-Alg with

$$\dot\Sigma = (S \circ -)\text{-Mon} \upharpoonleft (\mathrm{K}_G \vdash L = R),$$

we define $M : \mathcal{F}\text{-Alg} \to \mathcal{F}\text{-Alg}$ to send it to an $(\dot\Sigma + \dot\Psi)$-algebra with the same carrier of $H\langle A, T_\Sigma\alpha\rangle \in \dot\Psi\text{-Alg}$. Since $H\langle A, T_\Sigma\alpha\rangle$ already has a $\dot\Psi$-algebra, we only need to equip it with an $(S \circ -)$-operation. This can be done by the observation [Jaskelioff and Moggi 2010, Theorem 3.4] that *algebraic* operations can be transformed along monoid morphisms. Namely, the transformed operation is

$$\alpha^\sharp = [\![s : S, h : H_A \vdash \mu^H(\tau_A(\alpha_S(s, \eta^A)),\ h) : H_A]\!] \tag{3.14}$$

where H_A and τ_A stand for the carrier of $H\langle A,\ T_\Sigma\alpha\rangle$ and $\tau_{\langle A,T_\Sigma\alpha\rangle} : A \to H_A$ respectively, and $\alpha_S : S \circ A \to A$ is the component of α for the algebraic operation on A. We also need to show that the operation (3.14) satisfies the equation $\mathrm{K}_G \vdash L = R$. This follows from the functoriality of

$$L, R : ((S \circ -) + \Sigma_{\text{Mon}})\text{-Alg} \to G\text{-Alg},$$

which implies that the following diagrams commute

$$\begin{array}{ccc} G & \xrightarrow{L\langle A,\, \alpha\rangle} & A \\ & \searrow^{L\langle H_A,\, \alpha^\sharp\rangle} & \downarrow{\scriptstyle \tau_A} \\ & & H_A \end{array} \qquad \text{and} \qquad \begin{array}{ccc} G & \xrightarrow{R\langle A,\, \alpha\rangle} & A \\ & \searrow^{R\langle H_A,\, \alpha^\sharp\rangle} & \downarrow{\scriptstyle \tau_A} \\ & & H_A \end{array}$$

If α satisfies the equation $L = R$ (i.e. $L\langle A, \alpha\rangle = R\langle A, \alpha\rangle$), so does $\alpha^\sharp$. It can be shown that M is a *strict* modular model of $\dot\Psi$ by the same argument for the special case of exception monad transformers in Example 3.1∗8. □

3.3.4∗4. Example 3.1∗8 is exactly this theorem applied to the exception monad transformer. The *state monad transformer* $A \mapsto (A(S \times -))^S$ for a set S with $|S| < \kappa$ together with its model for the theory St_S of *mutable state* (Example 2.4∗9) yields a modular model of St_S in $\text{Alg}(\text{End}\text{o}_\kappa(\mathscr{C}))$. The *list monad transformer* $A \mapsto \mu X.A(1 + (- \times X))$ [Jaskelioff and Moggi 2010] with its model for the theory of *explicit nondeterminism* also gives rise to a modular model.

3.3.4∗5. Now we move on to scoped operations. First we recall that $\mathrm{Scp}_l(\mathscr{C})$ from 2.5∗19 is the operation family of scoped operations on monoids and *transliterations*. Let us again start with a concrete example.

3.3.4∗6 Example. The theory Ec of *exception throwing and catching* in Example 2.4∗15 is in the family $\mathrm{Scp}_l(\mathscr{C})$ for $\mathscr{C} = \langle \text{End}\text{o}_\kappa(\text{Set}), \circ, \mathrm{Id}\rangle$. A strict modular

model for it can be constructed by extending the modular model of throwing in Example 3.1∗8 with (1) a model of catching on $C_A = A \circ (1 + \mathrm{Id})$ and (2) a way to transform existing scoped operations on A to C_A.

For (1), we define $\mathit{catch} : (C_A \times C_A) \circ C_A \to C_A$ by $\mathit{catch} := \mu^C \cdot ((c \cdot s) \circ C_A)$ where $\mu^C : C_A \circ C_A \to C_A$ is the multiplication on C_A defined in Example 3.1∗8, and s is the following morphism in which the unlabelled arrow is the canonical strength for the functor $A \in \textsc{Endo}_\kappa(\textsc{Set})$:

$$s : C_A \times C_A = (A \circ (1 + \mathrm{Id})) \times C_A \to A \circ ((1 + \mathrm{Id}) \times C_A) \cong A \circ (C_A + \mathrm{Id} \times C_A),$$

and lastly the morphism $c : A \circ (C_A + \mathrm{Id} \times C_A) \to C_A$ is denoted by

$$\begin{aligned} a : A,\ b : (C_A + \mathrm{Id} \times C_A) \vdash \mu^C \Big(& (a, \iota_2\ *), \\ & \mathit{case}\ b\ \mathit{of}\ \{\iota_1\ h \mapsto h;\ \iota_2\ ih \mapsto \eta^C\ (\pi_1\ ih)\} \Big) : C_A \end{aligned}$$

The operational idea for this term is that the computation a is first executed and b is its result. The case $b = \iota_1\ h$ means that an exception is thrown and h is the exception handler, so h is executed in this case. On the other hand, the case $b = \iota_2\ ih$ means a normal termination, and the handler is ignored by $\pi_1\ ih$.

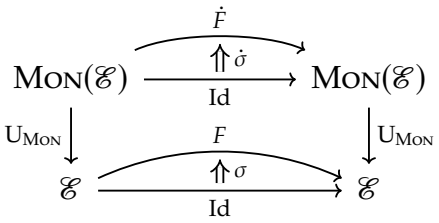
For (2), to transform a scoped operation $\alpha : S \circ A \circ A \to A$ on A to C_A, we define $\alpha^\sharp : S \circ C_A \circ C_A \to C_A$ by the term

$$s : S,\ a : A,\ m : 1 + \mathrm{Id},\ k : C_A \vdash \mu^C((\alpha(s, a, \eta^A), m),\ k) : C_A.$$

The transformed operation $\alpha^\sharp$ satisfies any constant equation $\mathrm{K}_C \vdash L = R$ whenever α does by the same argument as in the proof of Theorem 3.3.4∗3.

3.3.4∗7. Theorem 3.3.4∗3 constructs modular models for algebraic operations from monoid transformers. Jaskelioff and Moggi [2010] shows that this is also possible for scoped operations, provided that the monoid transformer is *functorial*.

3.3.4∗8 Definition (Jaskelioff and Moggi [2010]). A *functorial monoid transformer* on a monoidal category $\mathscr{E}$ consists of two functors $\dot{F} : \textsc{Mon}(\mathscr{E}) \to \textsc{Mon}(\mathscr{E})$ and $F : \mathscr{E} \to \mathscr{E}$ and two natural transformations $\dot{\sigma} : \mathrm{Id} \to \dot{F}$ and $\sigma : \mathrm{Id} \to F$:



such that $\mathrm{U}_{\textsc{Mon}} \circ \dot{\sigma} = \sigma \circ \mathrm{U}_{\textsc{Mon}}$

3.3.4∗9. As shown by Jaskelioff and Moggi [2010], many monad transformers in programming languages are functorial, including the exception monad trans-

former $M(E + -)$ for monads M, the state monad transformer $S \Rightarrow M(S \times -)$, the writer monad transformer $M(W \times -)$, and the (generalised) resumption monad transformer $\mu X.\, M(\Sigma X + -)$ [Cenciarelli and Moggi 1993]. However, the seemingly functorial list transformer $L_M := \mu X.\, M(1 + (- \times X))$ is in fact not functorial, because the associated natural transformation $\dot{\sigma} : M \to L_M$ defined by $a : M \vdash (a, \iota_2\, \langle *, (\eta^M, \iota_1\, \langle\rangle)\rangle) : M \circ (1 + \mathrm{Id} \times L_M)$ refers to the unit η^M of M, which is not a part of the underlying functor of M.

3.3.4∗10 Theorem. *Assume that the monoidal category $\mathscr{E}$ is left-closed. Let $\ddot{\Psi}$ be some $\langle \dot{\Psi}, T_\Psi \rangle \in \mathrm{Scp}_l(\mathscr{E})$. A functorial monoid transformer $\langle \dot{F}, F, \dot{\sigma}, \sigma \rangle$ and a functor $H : \mathrm{Mon}(\mathscr{E}) \to \dot{\Psi}\text{-Alg}$ such that $\dot{F} = T_\Psi \cdot H$ induce a strict modular model M of $\ddot{\Psi} \in \mathrm{Scp}_l(\mathscr{E})$ with an updater $u_{\langle \dot{\Sigma}, T_\Sigma, A, \alpha \rangle} = \sigma_{\langle A,\, T_\Sigma \alpha \rangle}$.*

Proof sketch. Compared to Theorem 3.3.4∗3, what is new is how scoped operations $\alpha : S \,\square\, A \,\square\, A \to A$ on a monoid $\langle A, \eta^A, \mu^A \rangle$ are transformed to FA. Such a transformation (not necessarily the unique one) is given by Jaskelioff and Moggi [2010], whose insight is that that scoped operations on A are the same as an algebraic operation on the monoid A/A, which embed A by Cayley's theorem (Example 2.3.2∗4), and we already know how to transform algebraic operations along monoid transformers. This is why we need (left) closedness in the assumption. We briefly record the transformation below and refer the reader to Jaskelioff and Moggi [2010, §5.1] for more details.

Firstly, recall that we have the Cayley embedding $e : A \to A/A$ and its retraction $r : A/A \to A$ defined as follows:

$$e = (a : A \vdash \lambda x.\, \mu^A(a, x) : A/A) \qquad r = (f : A/A \vdash f\ \eta^A : A).$$

We can similarly transpose the scoped operation $\alpha : S \,\square\, A \,\square\, A \to A$ on A to obtain a morphism $\tilde{\alpha} : S \to A/A$:

$$\tilde{\alpha} = (s : S \vdash \lambda x.\, \alpha(s, x, \eta^A) : A/A)$$

The transformed operation $\alpha^\sharp : S \,\square\, FA \,\square\, FA \to FA$ is then defined by

$$s : S, a : FA, b : FA \vdash \mu^{FA}\left(Fr\ \left(\mu^{F(A/A)}\ (\sigma_{A/A}\ (\tilde{\alpha}\ s), Fe\ a)\right), b\right) : FA$$

This transformed operation $\alpha^\sharp$ preserves any constant equation $\mathrm{K}_C \vdash L = R$ satisfied by α by the same argument for Theorem 3.3.4∗3. Moreover, the definition of $\alpha^\sharp$ is natural w.r.t. $\mathrm{Scp}_l(\mathscr{E})$: given any transliteration $f : S' \to S$ between scoped operations, we have

$$\alpha^\sharp \cdot (f \,\square\, FA \,\square\, FA) = (\alpha \cdot (f \,\square\, A \,\square\, A))^\sharp,$$

which can be directly checked or deduced from the general fact that all the term formers of monoidal algebraic theories are natural, similar to the abstraction

theorem of simply typed lambda calculus [Reynolds 1983]. □

3.3.4∗11 Remark. The theorem above needs the monoid transformer $\dot{F}$ to be over some $F : \mathscr{E} \to \mathscr{E}$ because the retract $r : A/A \to A$ of the Cayley embedding (Example 2.3.2∗4) used in the proof is not a monoid morphism, so we need $F : \mathscr{E} \to \mathscr{E}$ to have $Fr : F(A/A) \to A$. The requirement of having $\sigma : \mathrm{Id} \to F$ below $\dot{\sigma} : \mathrm{Id} \to \dot{F}$ is also essential. It is needed for showing that the updater is an algebra-homomorphism [Jaskelioff and Moggi 2010, Lemma 5.3] and the equations are preserved, which are omitted in the proof sketch above.

3.3.4∗12. A mistake in the earlier paper [Yang and Wu 2023] by the author is that the strict modular models from Theorem 3.3.4∗10 and Example 3.3.4∗6 were claimed to be w.r.t. the family $\textsc{Scp}(\mathscr{E})$ rather than $\textsc{Scp}_l(\mathscr{E})$. This is wrong because the operation transformation $\alpha^\sharp : S \,\square\, FA \,\square\, FA \to FA$ from a scoped operation $\alpha : S \,\square\, A \,\square\, A \to A$ in these modular models is not natural with respect to translations in $\textsc{Scp}(\mathscr{E})$, i.e. there exist translations T such that $(T\alpha)^\sharp \neq T(\alpha^\sharp)$.

For a counterexample, consider the modular model of exception catching in from Example 3.3.4∗6. Let $\ddot{\Sigma} \in \textsc{Scp}(\textsc{Endo}_\kappa(\textsc{Set}))$ be the theory of monads with a binary scoped operation $b : (\mathrm{Id} \times \mathrm{Id}) \circ A \circ A \to A$, and let $\ddot{\Sigma}' \in \textsc{Scp}(\textsc{Endo}_\kappa(\textsc{Set}))$ be an arbitrary theory. In the category Scp (but not in $\textsc{Scp}_l$), we have a translation $T : \ddot{\Sigma} \to \ddot{\Sigma}'$ that as a functor sends every $\langle A, \mu^A, \eta^A, \alpha\rangle \in \ddot{\Sigma}'\text{-}\textsc{Alg}$ to the $\ddot{\Sigma}$-algebra $\langle A, \mu^A, \eta^A, \mu^A \cdot (c \circ A)\rangle$ where c is

$$(\mathrm{Id} \times \mathrm{Id}) \circ A = A \times A \xrightarrow{s} A \circ (\mathrm{Id} \times A) \xrightarrow{A \circ \pi_2} A \circ A \xrightarrow{\mu^A} A$$

and the arrow s is the canonical strength $s_n : An \times An \to A(n \times An)$ for the set-endofunctor A. Note that this translation completely ignores the original operation α. It is perhaps more intuitive to use the syntax of an ordinary programming language here, say Haskell, which would be the following:

$$\begin{aligned} &T\alpha :: A\ x \to A\ x \to (x \to A\ y) \to A\ y \\ &T\alpha\ m\ n\ k = \textsc{do}\ \_ \leftarrow m; x \leftarrow n; k\ x \end{aligned}$$

In prose, T translates the binary scoped operation $b(x, y)$ to the computation that first runs x, ignores its result, and then runs y.

The operation lifting $\alpha^\sharp$ in Example 3.3.4∗6 for a binary scoped operation $\alpha :: A\ x \to A\ x \to (x \to A\ y) \to A\ y)$ would be the following in Haskell:

$$\begin{aligned} &\textsc{data}\ \mathit{Maybe}\ x \quad\ = \mathit{Nothing} \mid \mathit{Just}\ x \\ &\textsc{data}\ \mathit{MaybeT}\ A\ x = \mathit{MaybeT}\ (A\ (\mathit{Maybe}\ x)) \\ &\alpha^\sharp :: \mathit{MaybeT}\ A\ x \to \mathit{MaybeT}\ A\ x \to (x \to \mathit{MaybeT}\ A\ y) \to \mathit{MaybeT}\ A\ y \\ &\alpha^\sharp\ (\mathit{MaybeT}\ m')\ (\mathit{MaybeT}\ n')\ k = \textsc{do}\ x \leftarrow \mathit{MaybeT}\ (\alpha\ m'\ n'\ \mathit{return}); k\ x \end{aligned}$$

Now we can see that the two binary scoped operation $(T\alpha)^\sharp$ and $T(\alpha^\sharp)$ are not equal. The operation $T(\alpha^\sharp)$ written in Haskell would be

$$b_1 :: MaybeT\ A\ x \to MaybeT\ A\ x \to (x \to MaybeT\ A\ y) \to MaybeT\ A\ y$$
$$b_1\ m\ n\ k = \text{DO}\ \_ \leftarrow m; x \leftarrow n; k\ x$$

while the operation $(T\alpha)^\sharp$ would be

$$b_2\ (MaybeT\ m')\ (MaybeT\ n')\ k = \text{DO}\ x \leftarrow MaybeT\ (\text{DO}\ \_ \leftarrow m'; n'); k\ x$$

The difference is that when m throws an exception, i.e. when m' returns *Nothing*, b_1 will stop after m, whereas b_2 will continue as n'.

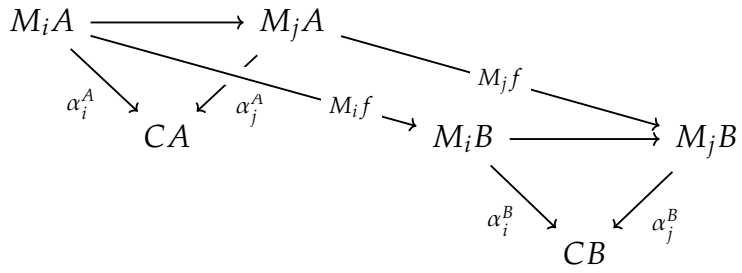
3.3.5 Colimits and limits of Model Transformers

3.3.5∗1. The category of model transformers (i.e. liftings along fibrations) seems to inherit many properties of categories of ordinary models (i.e. fiber categories). As a first step, in the following we show colimits and *reindexing-stable limits* of ordinary models can be lifted to model transformers.

3.3.5∗2 Theorem. *Let $P : \mathscr{A} \to \mathscr{T}$ be a fibration and $P' : \mathscr{A}' \to \mathscr{T}'$ be a fibration with a cleavage κ, and let $T : \mathscr{T} \to \mathscr{T}'$ be a functor and $\mathscr{D}$ be a category. If every fiber category $\mathscr{A}'_\Sigma$ of P' has (chosen) $\mathscr{D}$-indexed colimits, the category* $\textsc{Motr}(T)$ *of model transformers of T also has $\mathscr{D}$-indexed colimits, which are computed fiberwise.*

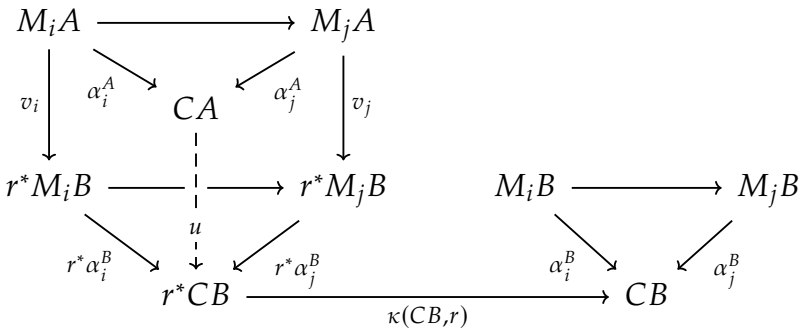
Moreover, if reindexing functors of P' preserve (or strictly preserve) $\mathscr{D}$-indexed colimits, the full subcategory of $\textsc{Motr}(T)$ *containing strong (or strict) model transformers is closed under $\mathscr{D}$-indexed colimits in* $\textsc{Motr}(T)$.

Proof. Let $M : \mathscr{D} \to \textsc{Motr}(T)$ be a $\mathscr{D}$-diagram of model transformers. We define a functor $C : \mathscr{A} \to \mathscr{A}'$ that sends every object $A \in \mathscr{A}$ to the colimit of $M_i A$ in the fiber category $\mathscr{A}_{TPA}$. For every morphism $f : A \to B$ in $\mathscr{A}$, let $\alpha_i^A : M_i A \to CA$ and $\alpha_i^B : M_i B \to CB$ be the colimiting cocones:



Writing $r := TPf$, for every $i \in \mathscr{D}$, the morphism $M_i f : M_i A \to M_i B$ factors as a vertical morphism $v_i : M_i A \to r^*(M_i B)$ in $\mathscr{A}'_{TPA}$ followed by a cartesian morphism. The reindexing functor r^* sends the cocone α_i^B in $\mathscr{A}'_{TPB}$ to a cocone

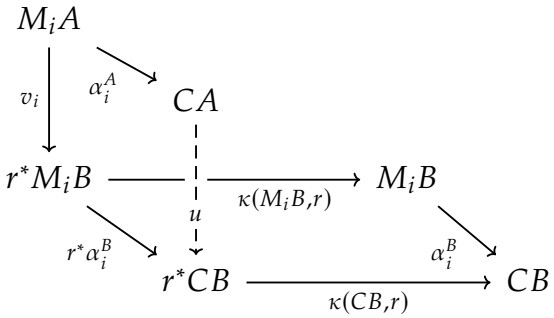
$r^*\alpha_i^B : r^*M_iB \to r^*CB$ in $\mathscr{A}'_{TPA}$:



(3.15)

The composite $(r^*\alpha_i^B) \cdot v_i : M_iA \to r^*CB$ can be checked to be a cocone too. By the universal property of CA as a colimit of M_iA, we have a unique vertical morphism $u : CA \to r^*CB$ such that $u \cdot \alpha_i^A = r^*\alpha^B \cdot v_i$. We define the action of C on the morphism $f : A \to B$ to be $\kappa(CB, r) \cdot u : CA \to CB$. The functoriality of C is a consequence of the functoriality of M_i and the universal property of CA as colimits. For example, if $f : A \to B$ above is $id_A : A \to A$, it can be checked by diagram chasing that for all $i \in \mathscr{D}$, $Cid_A \cdot \alpha_i^A = \alpha_i^A \cdot M_i id_A = \alpha_i^A$, so $Cid_A = id_A$. The case for $C(g \cdot f) = Cg \cdot Cf$ is more complex but similar.

The functor C is by construction a lifting of T. For each i, we show that the family of morphisms $\alpha_i^A : M_iA \to CA$ is natural in A. In the following diagram,



we have $Cf \cdot \alpha_i^A = \kappa(CB, r) \cdot u \cdot \alpha_i^A = \kappa(CB, r) \cdot r^*(\alpha_i^B) \cdot v_i$. The morphism $r^*\alpha_i^B$, which is the image of α_i^B under reindexing r^*, is by definition the unique morphism making the square at the bottom commute, so we have $\kappa(CB, r) \cdot r^*(\alpha_i^B) \cdot v_i = \alpha_i^B \cdot \kappa(M_iB, r) \cdot v_i = \alpha_i^B \cdot M_if$. Hence we have shown the required naturality: $Cf \cdot \alpha_i^A = \alpha_i^B \cdot M_if$, and thus we have a cocone $\langle \alpha_i \rangle_{i \in \mathscr{D}}$ in $\textsc{Motr}(T)$.

Given any cocone $\langle \beta_i : M_i \to N \rangle_{i \in \mathscr{D}}$, for every $A \in \mathscr{A}$, $\langle \beta_i^A \rangle$ is a cocone in $\mathscr{A}_{TPA}$ from M_iA to NA, so there is a unique mediating morphism $\sigma^A : CA \to NA$ such that $\sigma^A \cdot \alpha_i^A = \beta_i^A$. It can be checked by diagram chasing that the family of morphisms σ^A is natural in A, so C is the colimit of M_i in $\textsc{Motr}(T)$.

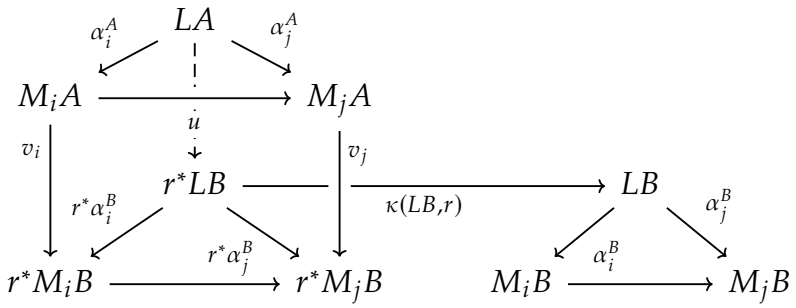
Finally, by the construction of the colimit C above, we can see that if reindexing functors of P' (strictly) preserve $\mathscr{D}$-indexed colimits in fiber categories, then when $f : A \to B$ is cartesian and all M_i are strong, the two cocones in the left of (3.15)

are isomorphic (the same), therefore C is strong (strict) too. □

3.3.5∗3. The situation for limits is slightly different: we need reindexing functors to preserve limits in fiber categories for $\textsc{Motr}(T)$ to inherit these limits. This requirement is not too demanding though, since in many fibrations of algebras and theories, reindexing functors are right adjoints so they preserve all limits.

3.3.5∗4 Theorem. *Let $P : \mathscr{A} \to \mathscr{T}$ be a fibration and $P' : \mathscr{A}' \to \mathscr{T}'$ be a fibration with a cleavage κ, and let $T : \mathscr{T} \to \mathscr{T}'$ be a functor and $\mathscr{D}$ be a category. If every fiber category $\mathscr{A}'_\Sigma$ of P' has (chosen) $\mathscr{D}$-indexed limits, and reindexing functors preserve these limits, then the category $\textsc{Motr}(T)$ has $\mathscr{D}$-indexed limits. Moreover, the subcategory containing strong/strict model transformers are closed under these limits.*

Proof sketch. Similar to the case of colimits above, the limit L of a diagram M_i of model transformers is defined fiberwise: for every object $A \in \mathscr{A}$, LA is defined to be the (chosen) limit of M_iA in the fiber category $\mathscr{A}'_{PTA}$. However, the action of L on a morphism $f : A \to B$ is different from the situation of colimits:



By reindexing the limiting cone $\alpha_i^B : LB \to M_iB$ along $r := TPf$, we have a cone $r^*\alpha_i^B : r^*LB \to r^*M_iB$. Let $v_i : M_iA \to r^*M_iB$ be the unique vertical morphism $\kappa(M_iB) \cdot v_i = M_i$. We have a cone $(v_i \cdot \alpha_i^A) : LA \to r^*M_iB$. Now we use the assumption that r^* preserves $\mathscr{D}$-limits, so $r^*\alpha_i^B : r^*LB \to r^*M_iB$ is still a limiting cone, and we have a vertical morphism $u : LA \to r^*LB$. The rest of this proof is similar to the proof of Theorem 3.3.5∗2. □

3.3.5∗5. Under the assumptions of Theorem 3.3.5∗4 and additionally that $P = P'$ are the same fibration, and $T : \mathscr{T} \to \mathscr{T}$ is equipped with $\eta : \mathrm{Id} \to T$, the category $\textsc{Motr}_u(T)$ of updatable model transformers also has $\mathscr{D}$-indexed limits. In fact, limits in $\textsc{Motr}_u(T)$ are *strictly created* by the forgetful functor $\mathrm{U} : \textsc{Motr}_u(T) \to \textsc{Motr}(T)$, which means that for every diagram $D : \mathscr{D} \to \textsc{Motr}_u(T)$, whenever $\mathrm{U} \circ D$ has a limiting cone $\alpha_i : L \to \mathrm{U}D_i$ in $\textsc{Motr}(T)$, there exists a unique updater u for L making $\alpha_i : \langle L, u\rangle \to D_i$ a limiting cone in $\textsc{Motr}_u(T)$.

To prove this, recall that an updater u for a model transformers M is a natural transformation $u : \mathrm{Id} \to M$ over $\eta : \mathrm{Id} \to T$. For a $\mathscr{D}$-indexed diagram $\langle M_i, u_i\rangle$ in $\textsc{Motr}_u(T)$, for every $A \in \mathscr{A}$, we have a vertical morphism $u_i^A : A \to \eta_A^* M_iA$.

Since morphisms in $\textsc{Motr}_u(T)$ are compatible with updaters, u_i^A is a cone over $\eta_A^* M_i A$. In the proof of Theorem 3.3.5∗4, the limit L of M_i is computed pointwise and fiberwise, so LA is the limit of $M_i A$ in the fiber $\mathscr{A}_{TPA}$. Moreover, the limit LA is preserved by reindexing η_A^*, so $\eta^* LA$ is a limit of $\eta_A^* M_i A$ in $\mathscr{A}_{PA}$, and the cone $u_i^A : A \to \eta_A^* M_i A$ then gives us a unique mediating morphism $u_A : A \to \eta^* LA$. It can be checked that this u is natural and is an updater for L.

Note however, the forgetful functor $\textsc{Motr}_u(T) \to \textsc{Motr}(T)$ does not create colimits: a cone $u_i^A : A \to \eta_A^* M_i A$ does not give us a morphism $A \to \eta_A^* CA$ into the colimit that commutes with $\eta_A^* \alpha_i^A : \eta_A^* M_i A \to \eta_A^* CA$ for all $i \in \mathscr{D}$.

3.3.5∗6. There are many more properties that we may wish to lift from ordinary models to model transformers. In particular, a question for the future is

> If every fiber category is locally κ presentable, under what conditions the category of model transformers is also locally κ-presentable?

3.3.6 Composition and Fusion of Model Transformer

3.3.6∗1. Model transformers are readily composable horizontally. Let M and N be two (strict/strong) model transformers of functors S and T respectively,

$$\begin{array}{ccccc} \mathscr{A} & \xrightarrow{M} & \mathscr{A}' & \xrightarrow{N} & \mathscr{A}'' \\ {\scriptstyle P}\downarrow & & \downarrow{\scriptstyle P'} & & \downarrow{\scriptstyle P''} \\ \mathscr{T} & \xrightarrow[S]{} & \mathscr{T}' & \xrightarrow[T]{} & \mathscr{T}'' \end{array}$$

it is immediate that the composite functor $N \circ M$ is a (strict/strong) model transformers of $T \circ S : \mathscr{T}' \to \mathscr{T}''$. Moreover, when P, P', and P'' are the same fibration, and the functors S and T are pointed, an updater u of M and an updater v of N can be composed horizontally to an updater $v \circ u : \mathrm{Id} \to N \circ M$ as well.

In particular, the composition of a modular model M of $\Sigma \in \mathscr{T}$ (i.e. a model transformer of $- + \Sigma : \mathscr{T} \to \mathscr{T}$) and a modular model N of $\Phi \in \mathscr{T}$ gives us a modular model of $\Sigma + \Phi$ via the isomorphism $- + (\Sigma + \Phi) \cong (- + \Sigma) + \Phi$.

3.3.6∗2 Example. Let M_E be the modular model of *exception* throwing and catching (Example 3.3.4∗6), and M_S be the modular model of *mutable state* arising from the state monad transformer by Theorem 3.3.4∗3. The composite $M_S \circ M_E$ is a modular model of $\textsc{Ec} + \textsc{St}_S$, the theories of exception and mutable state.

3.3.6∗3. Coproducts of theories are commutative, $\Sigma + \Phi \cong \Phi + \Sigma$, but the composition of modular models is of course not. For example, the opposite order $M_E \circ M_S$ of composing the modular models in Example 3.3.6∗2 gives rise to another modular model of the coproduct $\textsc{Ec} + \textsc{St}_S$. Both $M_S \circ M_E$ and $M_E \circ M_S$ satisfy the respective equations of exception and mutable state, but

they validate different interaction equations: $M_S \circ M_E$ additionally validates commutativity of stateful operations and exception throwing, so the following program equivalence is validated by $M_S \circ M_E$:

$$\mathit{catch}\ (\textsc{do}\ \mathit{put}\ s; \mathit{throw})\ h = \mathit{catch}\ (\textsc{do}\ \mathit{throw}; \mathit{put}\ s)\ h = \mathit{catch}\ \mathit{throw}\ h = h,$$

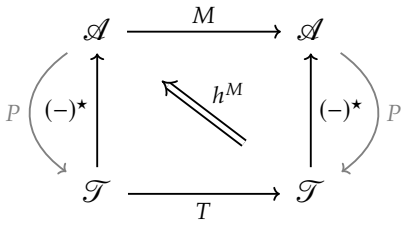
where the second step $\mathit{throw}; \mathit{put}\ s = \mathit{throw}$ is due to the algebraicity of *throw* as a nullary operation. On the other hand, $M_E \circ M_S$ validates

$$\mathit{catch}\ (\textsc{do}\ \mathit{put}\ s; p)\ h = \textsc{do}\ \mathit{put}\ s; \mathit{catch}\ p\ h,$$

so $\mathit{catch}\ (\textsc{do}\ \mathit{put}\ s; \mathit{throw})\ h = \textsc{do}\ \mathit{put}\ s; \mathit{catch}\ \mathit{throw}\ h = \textsc{do}\ \mathit{put}\ s; h$. An operational interpretation is that when an exception is caught, $M_S \circ M_E$ will roll back to the state before the *catch*, whereas $M_E \circ M_S$ will keep the state as it is. Both behaviours are desirable depending on the application. More discussion about interaction of effectful operations can be found in Yang and Wu [2021].

3.3.6∗4. A straightforward but useful result about composites of model transformers is the *fusion lemma* below: interpreting a term with two model transformers sequentially is equal to interpreting with the composite model transformer. Therefore two consecutive interpretations can be combined into one, eliminating the need to generate the intermediate result that is consumed immediately, a program optimisation known as *short-cut fusion* [Gill et al. 1993; Hinze et al. 2011].

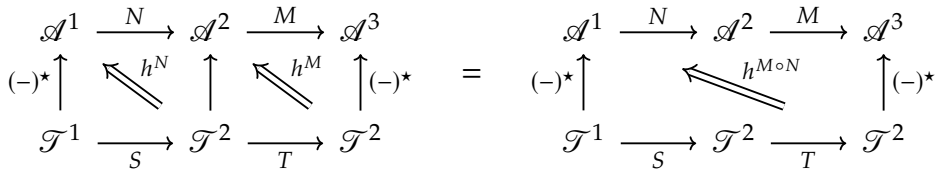
Generalising the natural transformation $h^M : (- + \ddot{\Psi})^\star \to M(-)^\star$ in 3.1∗25, let $P : \mathscr{A} \to \mathscr{T}$ be a fibration with a cleavage such that that all fiber categories have initial objects. We then have a functor $(-)^\star : \mathscr{T} \to \mathscr{A}$ that maps every object $\Sigma \in \mathscr{T}$ to the initial object 0_Σ in the fiber $\mathscr{A}_\Sigma$, and $(-)^\star$ maps every morphism $t : \Sigma \to \Gamma$ to the unique morphism $0_\Sigma \to t^* 0_\Gamma$ followed by the cartesian morphism over t. For every model transformer $M : \mathscr{A} \to \mathscr{A}$ of some functor $T : \mathscr{T} \to \mathscr{T}$, we then have a unique natural transformation $h^M : (T-)^\star \to M(-)^\star$:



that interprets the abstract syntax $(T\Sigma)^\star$ with the model $M\Sigma^\star$ for every $\Sigma \in \mathscr{T}$.

3.3.6∗5 Lemma (Fusion). For $i \in \{1, 2, 3\}$, let $P^i : \mathscr{A}^i \to \mathscr{T}^i$ be a cloven fibration such that every fiber category has initial objects. Given model transformers

$N : \mathscr{A}^1 \to \mathscr{A}^2$ of $S : \mathscr{T}^1 \to \mathscr{T}^2$ and $M : \mathscr{A}^2 \to \mathscr{A}^3$ of $T : \mathscr{T}^2 \to \mathscr{T}^3$, we have



i.e. $h^{M\circ N}_{\Sigma} = (Mh^N_{\Sigma}) \cdot h^M_{S\Sigma} : (TS\Sigma)^\star \to MN\Sigma^\star$ for every $\Sigma \in \mathscr{T}^1$.

Proof. The component at Σ of these two natural transformations are both the *unique* morphism out of the initial object of the fiber category over $TS\Sigma$. □

3.3.7 Modular Models in Symmetric Monoidal Categories

3.3.7∗1. In this subsection, we will have a look at some constructions of modular models that are only possible in symmetric monoidal categories $\mathscr{E}$, such as $\langle \mathscr{E}, \times, 1\rangle$ for cartesian monoids and $\langle \textsc{Endo}_\kappa(\textsc{Set}), *, \mathrm{Id}\rangle$ for applicative functors.

3.3.7∗2. To begin with, we can upgrade ordinary models of algebraic and scoped operations to modular models by using the fact that in a symmetric $\mathscr{E}$, two monoids $\langle A, \mu^A, \eta^A\rangle$ and $\langle B, \mu^B, \eta^B\rangle$ induces a monoid structure on $A \,\square\, B$.

3.3.7∗3 Theorem (Independent Combination). *Let $\mathscr{E}$ be a symmetric monoidal category and $\mathcal{F}$ be* $\textsc{Alg}(\mathscr{E})$ *or* $\textsc{Scp}_l(\mathscr{E})$*. For each $\ddot{\Psi} \in \mathcal{F}$, every $\dot{A} \in \ddot{\Psi}$-$\textsc{Alg}$ induces a strict modular model M of $\ddot{\Psi} \in \mathcal{F}$ such that $M\langle \ddot{\Sigma}, B, \beta\rangle$ is carried by $A \,\square\, B$, and M has an updater $u_{\ddot{\Sigma},B,\beta} = (B \cong I \,\square\, B \xrightarrow{\eta^A \square B} A \,\square\, B)$.*

Proof. Given a monoid $\langle B, \mu^B, \eta^B\rangle$, $A \,\square\, B$ has the following monoid structure:

$$\eta^{A\square B} = (I \cong I \,\square\, I \xrightarrow{\eta^A \square \eta^B} A \,\square\, B)$$
$$\mu^{A\square B} = \big((A \,\square\, B) \,\square\, (A \,\square\, B) \cong (A \,\square\, A) \,\square\, (B \,\square\, B) \xrightarrow{\mu^A \square \mu^B} A \,\square\, B\big)$$

Moreover, we can transform a scoped operation $\alpha : C \,\square\, A \,\square\, A \to A$ on A to a scoped operation $\alpha^\sharp$ on $A \,\square\, B$ as follows:

$$C \,\square\, (A \,\square\, B) \,\square\, (A \,\square\, B) \cong C \,\square\, (A \,\square\, A) \,\square\, (B \,\square\, B) \xrightarrow{\alpha \square \mu^B} A \,\square\, B$$

Symmetrically, every scoped operation on B can also be transformed to $A \,\square\, B$. Furthermore, algebraic operations are special cases of scoped operations, so they can be transformed in the same way. The preservation of (constant) equations of the operation transformation is the same as the proof of Theorem 3.3.4∗3. □

3.3.7∗4. For $\mathscr{E} = \langle \textsc{Endo}_\kappa(\textsc{Set}), *, \mathrm{Id}\rangle$, the intuition for $A * B$ is that two applicative-computations A and B are combined in the way that they execute *independently*, and operations act on $A * B$ pointwise.

There is another way to compose two applicatives, namely $A \circ B$ [McBride and Paterson 2008]. In this way, the B-computation can *depend* on the result of A.

3.3.7∗5 Theorem (Dependent Combination). *Let $\mathscr{E}$ be $\langle \textsc{Endo}_\kappa(\textsc{Set}), *, \mathrm{Id}\rangle$ and $\mathcal{F}$ be $\textsc{Alg}(\mathscr{E})$ or $\textsc{Scp}_l(\mathscr{E})$. For each $\ddot{\Psi} \in \mathcal{F}$, every $\dot{A} \in \ddot{\Psi}$-$\textsc{Alg}$ induces a strict modular model M of $\ddot{\Psi}$ such that $M_{\ddot{\Sigma}}\langle B, \beta\rangle$ is carried by $A \circ B$, and M has an updater $u_{\ddot{\Sigma},B,\beta} = \eta^A \circ B$.*

Proof sketch. Given two applicative functors $\langle A, \mu^A, \eta^A\rangle$ and $\langle B, \mu^B, \eta^B\rangle$, their composition $A \circ B$ as functors can be equipped with an applicative structure with unit $\eta^{A\circ B} = \eta^A \circ \eta^B$ and the following multiplication $\mu^{A\circ B}$:

$$\begin{aligned}((A \circ B) * (A \circ B))n &\cong \int^{m,k} A(Bm) \times A(Bk) \times n^{m\times k} \\ &\xrightarrow{f} \int^{m,k} A(Bm) \times A(Bk) \times (Bn)^{Bm\times Bk} \\ &\xrightarrow{g} \int^{m',k'} Am' \times Ak' \times (Bn)^{m'\times k'} \\ &\cong (A * A)(Bn) \xrightarrow{\mu^A} A(Bn)\end{aligned}$$

where the arrow g is the substitution of the bound variables of the coend $m' = Bm$ and $k' = Bk$; the arrow f uses functoriality of the coend and the morphism $n^{m\times k} \to (Bn)^{Bm\times Bk}$ given by the transpose of the following morphism:

$$n^{m\times k} \times Bm \times Bk \xrightarrow{\iota_{m,k}} \int^{m,k} Bm \times Bk \times n^{m,k} \cong (B * B)n \xrightarrow{\mu^B} Bn.$$

To transform a scoped operation $\alpha : S * A * A \to A$ to $A \circ B$, we use the fact that there is a canonical morphism $s : S * (A \circ B) \to (S * A) \circ B$ as follows:

$$\begin{aligned}(S * (A \circ B))n &\cong \int^{m,k} Sm \times A(Bk) \times n^{m\times k} \\ &\to \int^{m,k} Sm \times A(Bk) \times (Bn)^{m\times Bk} \\ &\to \int^{m,k'} Sm \times Ak' \times (Bn)^{m\times k'} \\ &\cong (S * A)(Bn)\end{aligned}$$

where the first step uses the action of the functor B on morphisms: $n^k \to (Bn)^{(Bk)}$, and the second steps is the substitution of the bound variable $k' = Bk$. We define the transformation of α to $A \circ B$ to be

$$\begin{aligned}S * (A \circ B) * (A \circ B) &\xrightarrow{s*(A\circ B)} ((S * A) \circ B) * (A \circ B) \\ &\xrightarrow{(\overline{\alpha}\circ B)*(A\circ B)} (A \circ B) * (A \circ B) \xrightarrow{\mu} A \circ B\end{aligned}$$

where $\overline{\alpha} = (S * A \xrightarrow{S*A*\eta^A} S * A * A \xrightarrow{\alpha} A)$.

To transform a scoped operation $\beta : G * B * B \to B$ to $A \circ B$, we need the

following canonical morphism $t : G * (A \circ B) \to A \circ (G * B)$:

$$\begin{aligned}(G * (A \circ B))n &\cong \textstyle\int^{m,k} Gm \times A(Bk) \times n^{m \times k} \\ &\to \textstyle\int^{m,k} A(Gm \times Bk \times n^{m \times k}) \\ &\to \textstyle\int^{m,k} A((G * B)n) \\ &\cong A((G * B)n)\end{aligned}$$

where the first step uses the canonical strength of A to push Gm and $n^{m \times k}$ inwards; the second step uses the coprojection $\iota_{m,k} : Gm \times Bk \times n^{m \times k} \to (G * B)n$. With t we define the transformed scoped operation on $A \circ B$:

$$\begin{aligned}G * (A \circ B) * (A \circ B) &\xrightarrow{t*(A \circ B)} (A \circ (G * B)) * (A \circ B) \\ &\xrightarrow{(A \circ \overline{\beta})*(A \circ B)} (A \circ B) * (A \circ B) \\ &\xrightarrow{\mu^{A \circ B}} A \circ B\end{aligned}$$

where $\overline{\beta} = (G * B \xrightarrow{G*B*\eta^B} G * B * B \xrightarrow{\beta} G)$. □

3.3.7∗6. To see the difference between Theorem 3.3.7∗3 and Theorem 3.3.7∗5, let $\dot{A}$ be the applicative functor induced by the exception monad $\bar{E} + \mathrm{Id}$. It is a model of the applicative version of the theory ET_E of exception *throwing*, equipped with an operation $throw : \bar{E} * (\bar{E} + \mathrm{Id}) \to (\bar{E} + \mathrm{Id})$. Using Theorem 3.3.7∗5, it can be extended to a modular model using $(\bar{E} + \mathrm{Id}) \circ B \cong (\bar{E} + B)$ for all applicatives B. In this model, it holds that for all elements $x, y \in (\bar{E} + B)X$, $throw\ \langle e, x\rangle = \iota_1 e = throw\ \langle e, y\rangle$, which means that exception throwing discards any B-computation. But it is not true for the independent composition $(\bar{E} + \mathrm{Id}) * B$.

3.3.7∗7. Our final example is an interesting modular model for *phased computation*, generalising the construction that Kidney and Wu [2021] and Gibbons et al. [2022] use for *breadth-first search*. The theory $\textsc{Pha} \in \textsc{Scp}(\mathscr{E})$ has a unary scoped operation *later*. The intention is that a program may have multiple phases of execution, and the operation *laterp* delays the execution of p to the next phase.

For example, if F is an applicative functor in Haskell with $later :: F\ a \to F\ a$, given $p_i :: F\ a$, the following Haskell program of type $F\ a$

```
do later (do later p3
             p21)
   p11
   later     p22
   p12
```

is supposed to execute p_{11} and p_{12} at phase 1, p_{22} and p_{21} at phase 2, and p_3 at phase 3. A standard example of such an applicative functor F is the nested

list functor $[[a]]$, where the i-th element of the outer list contains all possible outcomes of the computation at phase i, and *later* $xs = [[\,] : xs]$.

In a symmetric closed monoidal category $\mathscr{E}$ such that every object has a free monoid over it, given a monoid $\dot{A} = \langle A, \mu^A, \eta^A \rangle$, Kidney and Wu's [2021] idea can be abstracted as equipping (the carrier of) the free monoid $S_A = \mu X.\, A \square X + I$ over A with a nonstandard monoid structure $\langle S_A, \mu^{S_A}, \eta^{S_A} \rangle$ with $\eta^{S_A} : I \to \mu X.\, A \square X + I$ given by $\vdash in\ (\iota_2\ *)$ where $out : (S_A \cong A \square S_A + I) : in$ is the isomorphism for the initial algebra, and μ^{S_A} is denoted by $s : S_A, t : S_A \vdash m : S_A$ where m is

$$
\begin{array}{l}
case\ (out\ s, out\ t)\ of \\
\quad (\iota_1\ (a, x),\ \iota_1\ (a', y)) \mapsto in\ (\iota_1\ (\mu^A(a, a'), \mu^{S_A}(x, y)) \\
\quad (\iota_2\ *, \qquad y) \qquad\quad \mapsto in\ y \\
\quad (x, \qquad \iota_2\ *) \qquad\quad \mapsto in\ x
\end{array}
$$

Note that the use of variable x and a' in the first case does not match their order in the context, so we need a symmetric monoidal category, and we also need closedness for interpreting structural recursion on the initial algebra S_A. This construction is essentially the same idea as the list object in the category of (ordinary) monoids that we saw in 2.2.1∗16. The intuition is that $S_A = \mu X.A \square X + I$ is a list of A-computations at each phase, and μ^{S_A} merges two lists by multiplying computations at the same phase. The *later* operation on S_A is defined as

$$p : S_A, k : S_A \vdash \mu^{S_A}\ (in\ (\iota_1\ (\eta^A, p), k)) : S_A.$$

The construction $A \mapsto S_A$ is a functorial monoid transformer, so we can use Theorem 3.3.4∗10 to obtain a modular model of phasing in $\textsc{Scp}_I(\mathscr{E})$.

* * *

3.3.7∗8. To summarise this chapter, we have studied modular constructions of algebraic structures in the framework of lifting functors $T : \mathscr{T} \to \mathscr{T}'$ along two fibrations $P : \mathscr{A} \to \mathscr{T}'$ and $P' : \mathscr{A}' \to \mathscr{T}'$. The base categories $\mathscr{T}$ and $\mathscr{T}'$ of the fibrations contain some notion of algebraic theories, and the total categories $\mathscr{A}$ and $\mathscr{A}'$ contain models of all these theories. The functor $T : \mathscr{T} \to \mathscr{T}'$ transforms every theory in a certain way, for example, by combining it with another fixed theory. Liftings of T along P and P' sends an object in every fiber category $\mathscr{A}_\Sigma$ to an object in the fiber $\mathscr{A}'_{T P \Sigma}$, so we call them *model transformers*. We can intuitively think of a functor $T : \mathscr{T} \to \mathscr{T}$ as a theory $T\Sigma$ parameterised by some potential future extension with $\Sigma \in \mathscr{T}$, then a model transformer M of T can be thought of as a model of the 'parameterised theory' T.

We have also seen a handful of universal constructions of model transformers as well as some more concrete constructions, such as using monoid transformers. Lastly, we comment that liftings along fibrations have many other applications in

computer science, such as in logical relations for computational types (lifting a computational monad T along a fibration of predicates over sets) [Katsumata 2005], in Hoare logics (lifting a computation monad T along a fibration of specifications over types) [Aguirre et al. 2022], and in behaviour metrics of states of automata (lifting a coalgebra encoding an automaton along a fibrations of metric spaces over sets) [Baldan et al. 2014]. It is an interesting question for the future to find out whether the lifting techniques developed in these contexts give interesting modular models of algebraic theories.

Part II

Language

Chapter 4

A Logical Framework for LCCCs

4∗1. In the second part of this thesis, we switch our topic from categorical structures to programming languages of higher-order algebraic effects. The design space for such a language is vast:

- In the axis of foundational strength, does it have dependent types, impredicative polymorphism, general recursion, quotient types?
- In the axis of user interface, does it have an effect system? Does it provide pattern matching on computations (i.e. shallow handlers) or only the recursor of computations (i.e. deep handlers)? Does it call by value, call by push value, or call by name? Does it allow dependent LET-binding?

As a first step, in this thesis we explore a corner of the space, particularly, a (fine-grained) call-by-value language with impredicative polymorphism and possibly general recursion but no full dependent types, which we call *System* F^{ha}_{ω}.

4∗2. The author's motivation for studying this particular configuration of language features is that Nicolas Wu and the author are implementing a Haskell library of higher-order algebraic effects, which *uses more than 40 GHC extensions*, so the author hopes to clarify what exactly language features are needed for implementing higher-order algebraic effects in languages similar to Haskell.

The omission of dependent types implies that we will have only degenerated *equation-less* algebraic theories in $\mathrm{F}^{\mathrm{ha}}_{\omega}$, and it is not just the equations on effectful operations that we have to leave out; we cannot enforce the monadic laws on user-defined monads either. However, what is interesting, and surprising to the author, is that even though user-defined models of effects are lawless 'raw monads', the computation judgements of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ can still satisfy the monadic laws strictly and can be eliminated into lawless monads without causing inconsistency.

4∗3. Before we go into $\mathrm{F}^{\mathrm{ha}}_{\omega}$, in this chapter we introduce a *logical framework* (LF) that we will use for studying $\mathrm{F}^{\mathrm{ha}}_{\omega}$. Logical frameworks are type theories for defining logics and programming languages conveniently. In particular, the logical framework in this chapter will not only provide us with a concise notation

for defining programming languages, liberating us from the bureaucracy of dealing with variables and contexts that are common in programming languages, but also provide a notion of categorical models automatically for languages defined using the logical framework.

The categorical structure corresponding to the LF in this chapter is *locally cartesian closed categories* (LCCCs), just as monoidal categories correspond to *monoidal algebraic theories* in Section 2.3, and as categories with finite products correspond to multi-sorted algebraic theories, and as categories with finite limits correspond to essentially algebraic theories [Adámek and Rosicky 1994]. For this reason, we will refer to the LF by LccLF in this thesis.

4∗4. LccLF is originally introduced in Sterling's [2021, §1] thesis, as a simpler alternative to Uemura's [2021; 2023] *second-order generalised algebraic theories*. LccLF has been used in the study of several type theories [Grodin et al. 2024; Niu et al. 2022; Sterling and Angiuli 2021; Sterling and Harper 2022]. We refer the reader to [Sterling 2021, §0.1.2.2 and §1.2∗2] for an insightful discussion of the comparison of LccLF and other logical frameworks.

However, some crucial meta-theory of LccLF is still missing in the literature. Although readers well versed in categorical logic can probably see how it should work, in this chapter we take the chance to work it out in some detail.

4∗5. The structure of this chapter is as follows:

- In Section 4.1, we introduce the syntax of LccLF and explain how a programming language can be defined as a *signature* in LccLF (which is formally a context in LccLF) following the motto of *judgements as types*.
- In Section 4.2, we describe the *category of judgements* $\textsc{Jdg}\, S$ of a signature S. Functors $M : \textsc{Jdg}\, S \to \mathscr{C}$ into an LCCC $\mathscr{C}$ preserving the locally cartesian closed structure then provide *functorial models* of the signature S in $\mathscr{C}$ in the tradition of functorial semantics pioneered by Lawvere [1963].
- In Section 4.3, we define a notion of *diagrammatic models* of an LccLF-signature S. Unlike a functorial model $M : \textsc{Jdg}\, S \to \mathscr{C}$, which specifies the interpretation of *all* the judgements generated by the signature S in a coherent way, a diagrammatic model only needs to specify the interpretation of the generating operations in the signature S, so diagrammatic models are the notion that we actually want to work with when concretely defining a model of a signature. Due to type dependency in LccLF, the definition of diagrammatic models is much more involved than that of, e.g. monoidal algebraic theories in 2.3.1∗6. The main technical tool that we will need is the concept of *universes* in categories, and the theorem that for every LCCC

$\mathscr{C}$, there is a universe in $\textsc{Pr}\,\mathscr{C}$ that classifies exactly the Yoneda embedding of objects and morphisms of $\mathscr{C}$.

* In Section 4.4, we prove that there is an equivalence of groupoids

$$S\text{-}\textsc{Mod}(\mathscr{C}) \;\cong\; \mathrm{LCCC}_{\cong}(\textsc{Jdg}\,S, \mathscr{C}),$$

where S-$\textsc{Mod}(\mathscr{C})$ contains diagrammatic models of a signature S and isomorphisms between them, and $\mathrm{LCCC}_{\cong}(\textsc{Jdg}\,S, \mathscr{C})$ is the groupoid of functorial models and natural isomorphisms. To define the groupoid S-$\textsc{Mod}(\mathscr{C})$, a technique similar to, but at one dimensional higher than, Altenkirch et al.'s [2019] syntactic translation for *setoid type theory* is used. The reason that only isomorphisms instead of homomorphisms are considered that LccLF has dependent function types so type expressions are not necessarily covariant with respect to variables in it.

* In Section 4.5, we make some wrap-up comments on this chapter.

4∗6. In the rest of the thesis, we will assume familiarity with (extensional) dependent type theory and its categorical semantics. An exceptionally good exposition on dependent type theory is Angiuli and Gratzer [2024]; an elementary account of the categorical semantics is Hofmann [1997], which should be sufficient for our needs in this thesis; a comprehensive textbook account is Jacobs [1999]. Some basic knowledge of topos theory is also helpful, as all denotational models in this thesis are either in presheaf toposes or realizability toposes.

4.1 Syntax of the Logical Framework

4.1∗1. In brief, the logical framework LccLF is a dependent type theory with a unit type 1, Σ-types, and a Tarski-style universe type $\mathbb{J}$ such that

1. the universe $\mathbb{J}$ is closed under the unit type and Σ-types;
2. the universe $\mathbb{J}$ is closed under *extensional* equality types;
3. there are Π-types $\Pi\ A\ B$ provided that A is in the universe $\mathbb{J}$. If the codomain B is a type family valued in $\mathbb{J}$, the Π-type $\Pi\ A\ B$ is also in $\mathbb{J}$.

In other words, $\mathbb{J}$ is a universe having all connectives of extensional Martin-Löf type theory (MLTT), while types outside $\mathbb{J}$ have only the unit type, Σ-types, and restricted Π-types whose domain must be in $\mathbb{J}$.

The precise type formation rules of the logical framework are in Figure 4.1, and the term formation rules for the universe $\mathbb{J}$ are in Figure 4.2. All other rules, including the rules for contexts, substitutions, term formations, and judgemental equalities (β and η equalities for all type formers) are the same as the usual

$$\frac{}{\Gamma \vdash 1 \text{ type}} \qquad \frac{\Gamma \vdash A \text{ type} \qquad \Gamma, a : A \vdash B \text{ type}}{\Gamma \vdash \Sigma\ A\ B \text{ type}} \qquad \frac{}{\Gamma \vdash \mathbb{J} \text{ type}} \qquad \frac{\Gamma \vdash A : \mathbb{J}}{\Gamma \vdash \mathrm{El}\ A \text{ type}}$$

$$\frac{\Gamma \vdash A : \mathbb{J} \qquad \Gamma \vdash a : \mathrm{El}\ A \qquad \Gamma \vdash b : \mathrm{El}\ A}{\Gamma \vdash \mathrm{Eq}(a, b) \text{ type}} \qquad \frac{\Gamma \vdash A : \mathbb{J} \qquad \Gamma, a : \mathrm{El}\ A \vdash B \text{ type}}{\Gamma \vdash \Pi\ A\ B \text{ type}}$$

Figure 4.1: Type formation rules for the logical framework

$$\frac{}{\Gamma \vdash \widehat{1} : \mathbb{J}} \qquad \frac{\Gamma \vdash A : \mathbb{J} \qquad \Gamma, a : \mathrm{El}\ A \vdash B : \mathbb{J}}{\Gamma \vdash \widehat{\Sigma}\ A\ B : \mathbb{J}}$$

$$\frac{\Gamma \vdash A : \mathbb{J} \qquad \Gamma \vdash a : \mathrm{El}\ A \qquad \Gamma \vdash b : \mathrm{El}\ B}{\Gamma \vdash \widehat{\mathrm{Eq}}(a, b) : \mathbb{J}} \qquad \frac{\Gamma \vdash A : \mathbb{J} \qquad \Gamma, a : \mathrm{El}\ A \vdash B : \mathbb{J}}{\Gamma \vdash \widehat{\Pi}\ A\ B : \mathbb{J}}$$

Figure 4.2: Codes of types in the universe $\mathbb{J}$

extensional Martin-Löf type theory [Hofmann 1997; Martin-Löf 1984; Nordström et al. 1990] and thus omitted here.

A small difference between the framework defined here and the one in Sterling's thesis [2021] is that op. cit. *all types* have extensional equality types, not just those in $\mathbb{J}$. Those extensional equality types play the role of *sort equations* in Cartmell's [1986] generalised algebraic theories. Although they are sometimes handy when specifying type theories in the LF, they necessitate considerations of strict equalities between objects in a category when considering categorical models of theories specified in the LF, making the notion of models not invariant under equivalences of categories. They also complicate the definition of isomorphisms of models. For these reasons, they are left out in the LF here.

4.1∗2 Notation. In the rest of the thesis we will extensively use dependent type theories – the logical framework for presenting object type theories and internal languages of categories for studying the meta-theoretic properties of the object type theories. To make working with type theories as natural as working with ordinary maths, we impose the following notational conventions, which resemble the concrete syntax of Agda [Norell 2009].

- Dependent function types, i.e. Π-types, are written as $(a : A) \to B$, or $A \to B$ when B does not depend on A. Function abstraction is $\lambda(x : A).\ t$, or $\lambda x.\ t$ if A can be inferred; function application is $f\ a$ as usual.
- We use *implicit function types* $\{a : A\} \to B$, whose function application and abstraction are elided when they can be inferred or have a unique choice.

When they are hard to infer, we make their application and abstraction explicit by writing $f\ \{a\}$ and $\lambda\{a : A\}.\ t$ respectively.

When the type A of a can be inferred from the uses of a, we sometimes further abbreviate the notation $\{a : A\} \to B$ as $\{a\} \to B$.

* Dependent pair types, i.e. Σ-types, are written as $\Sigma(a : A).\ B$, or $A \times B$ if B does not depend on A. Pairing is (a, b) and projections are $\pi_1\ p$ and $\pi_2\ p$. We also use the record syntax for iterative Σ-types with named fields:

$$\begin{aligned}&\textsc{record}\ R : \mathscr{U}\ \textsc{where}\\ &\quad \mathit{fld}_1 : A_1\\ &\quad \dots\\ &\quad \mathit{fld}_n : A_n\end{aligned}$$

means the iterative Σ-type of the fields $A_1, \dots A_n$, where each field may depend on previous fields. A field of a record $r : R$ is accessed by $r.\mathit{fld}_i$. A record $r : R$ is constructed using the 'co-pattern matching' syntax which specifies each field of r by a list of declarations:

$$\begin{aligned}&r.\mathit{fld}_1 = t_1\\ &\dots\\ &r.\mathit{fld}_n = t_n\end{aligned}$$

* We will use the same notation for type formers and their codes in universes. The decoding operator El of universes will be also elided, as if we are working with Russell-style universes. For example, we write

$$A : \mathbb{J}, B : A \to \mathbb{J} \vdash (x : A) \to B : \mathbb{J}$$

to mean $A : \mathbb{J}, B : \mathrm{El}\ A \to \mathbb{J} \vdash \widehat{\Pi}\ A\ B : \mathbb{J}$.

* The extensional equality type $\mathrm{Eq}(a, b)$ will be written as simply $a = b$, and its only constructor is $\mathit{refl} : a = a$. There is no special notation for consuming extensional equality types since they are consumed by *equality reflection*: if we have an element of the type $a = b$, then a and b can be used interchangeably, i.e. they are judgementally equal.

* An identifier that contains underscores '_' is used as an operator. For example, if $\_+\_ : A \to A \to A$, we can write $a + b$ for $a, b : A$. Such operators do not have to be binary. For example, if $\_\langle\_\rangle\_ : A \to B \to A \to C$, we can write $a\langle b\rangle a' : C$ for $a, a' : A$ and $b : B$.

  However, a single underscore '_' that appears alone just means a 'wildcard' that take the place of something inferable or irrelevant.

4.1∗3. A type theory is defined in the logical framework as a context, or equivalently a closed type, since a context $(a_1 : A_1, \ldots, a_n : A_n)$ can be packed into a record type with fields $a_i : A_i$. The idea is the *judgements-as-types* principle of the Edinburgh Logical Framework [Harper et al. 1993]: *judgements* of the object type theory (e.g. something being a type) are declared as types in the universe $\mathbb{J}$ in the logical framework; *inference rules* are declared as functions between judgements; *deductions* are then terms of judgements that make use of the previously declared judgements and inference rules.

4.1∗4 Terminology. To avoid confusion with concepts in object type theories, henceforth we will call LF contexts *signatures* or occasionally *theories*. The variables of an LF contexts are referred to as *declarations* of the signature. LF types in the universe $\mathbb{J}$ will be called *judgements*.

4.1∗5 Example. The signature of *barebone type theory* has two declarations

$$ty : \mathbb{J} \qquad\qquad tm : ty \to \mathbb{J}$$

which are respectively the judgement for *something being a type* and the family of judgements for *something being a term of a type*. This signature alone is not very interesting but it serves as a basic building block for more complex type theories.

4.1∗6 Example. The signature of *simply typed λ-calculus* (STLC) extends barebone type theory (Example 4.1∗5) with the following declarations:

$$\begin{aligned}
\iota : ty \qquad\qquad & _\Rightarrow_ : ty \to ty \to ty \\
abs : \{a, b : ty\} &\to (tm\ a \to tm\ b) \to tm\ (a \Rightarrow b) \\
app : \{a, b : ty\} &\to tm\ (a \Rightarrow b) \to (tm\ a \to tm\ b) \\
_ : \{a, b : ty\} &\to \{f : tm\ a \to tm\ b\} \to app\ (abs\ f) = f \\
_ : \{a, b : ty\} &\to \{g : tm\ (a \Rightarrow b)\} \to abs\ (app\ g) = g
\end{aligned}$$

The declaration $\iota : ty$ corresponds to the inference rule of a base type in STLC, and the declaration $\_\Rightarrow\_$ corresponds to the inference rule of (non-dependent) function types. Terms of function types are specified using *higher-order abstract syntax* (HOAS) via an isomorphism with the function space in the logical framework, which is also the reason why we do not need a judgement of STLC *contexts*. With these declarations, we can define STLC terms such as

$$abs\ (\lambda f \to abs\ (\lambda x \to app\ f\ (app\ f\ x))) : tm\ ((\iota \Rightarrow \iota) \Rightarrow (\iota \Rightarrow \iota)).$$

4.1∗7. It is a common pattern that a type former in the object type theory is specified by internalising an LF judgement via an isomorphism, which is precisely the purpose of introducing logical frameworks. Thus for convenience we define the judgement of isomorphisms given two judgements A, B:

$$
\begin{array}{l}
\textsc{record}\ A \cong B : \mathbb{J}\ \textsc{where} \\
\quad \mathit{fwd}\ : A \to B \\
\quad \mathit{bwd} : B \to A \\
\quad \_\quad\ : (a : A) \to \mathit{bwd}\ (\mathit{fwd}\ a) = a \\
\quad \_\quad\ : (b : B) \to \mathit{fwd}\ (\mathit{bwd}\ b) = b
\end{array}
$$

Using isomorphisms to LF judgements to specify object type theories does not entail that object type theories are restricted to sublanguages of the logical framework. The following two examples shows how general recursion and impredicative polymorphism can be specified in this way, although the logical framework does not have general recursion or any impredicativity.

4.1∗8 Example. The signature of PCF [Plotkin 1977] extends STLC in Example 4.1∗6 with a fixed-point combinator at every type

$$\mathit{Y} : \{a : \mathit{ty}\} \to (\mathit{tm}\ a \to \mathit{tm}\ a) \to \mathit{tm}\ a$$

as well as some new base types and terms

$$
\begin{array}{c}
0 : \mathit{tm}\ \iota \qquad \mathit{succ}, \mathit{pred} : \mathit{tm}\ \iota \to \mathit{tm}\ \iota \qquad \mathit{o} : \mathit{ty} \qquad \mathit{tt}, \mathit{ff} : \mathit{tm}\ o \\
\mathit{iszero} : \mathit{tm}\ \iota \to \mathit{tm}\ o \qquad \supset : \{a : \mathit{ty}\} \to \mathit{tm}\ o \to \mathit{tm}\ a \to \mathit{tm}\ a
\end{array}
$$

and also the following equational declarations (whose names are irrelevant):

$$
\begin{array}{c}
\{n : \mathit{tm}\ \iota\} \to \mathit{pred}\ (\mathit{succ}\ n) = n \qquad \mathit{pred}\ 0 = 0 \qquad \mathit{iszero}\ 0 = \mathit{tt} \\
\{n : \mathit{tm}\ \iota\} \to \mathit{iszero}\ (\mathit{succ}\ n) = \mathit{ff} \\
\{a : \mathit{ty}\}\ \{x, y : \mathit{tm}\ a\} \to (\supset\ \mathit{tt}\ x\ y = x)\ \times\ (\supset\ \mathit{ff}\ x\ y = y) \\
\{a : \mathit{ty}\} \to \{f : \mathit{tm}\ a \to \mathit{tm}\ a\} \to f\ (\mathit{Y}\ f) = \mathit{Y}\ f
\end{array}
$$

Within the logical framework, in the context of this signature, we can write programs such as addition of numbers:

$$
\begin{array}{l}
\_+\_ : \mathit{tm}\ (\iota \Rightarrow \iota \Rightarrow \iota) \\
\_+\_ = \mathit{abs}\ (\lambda n \to \mathit{Y}\ (\lambda \mathit{rec} \to \mathit{abs}\ (\lambda m \to \\
\qquad\qquad \supset\ (\mathit{iszero}\ m)\ n\ (\mathit{succ}\ (\mathit{app}\ \mathit{rec}\ (\mathit{pred}\ m))))))
\end{array}
$$

The equational axioms in the signature implies, for example, that $0 + \mathit{succ}\ 0$ is judgementally equal to $\mathit{succ}\ 0$ in the logical framework. Note that in this way PCF is formulated as an equational theory rather than a reduction system of terms, viz a small-step operational semantics.

4.1∗9 Example. The signature of *System F* [Girard 1972; Reynolds 1974] extends the one of STLC in Example 4.1∗6 with the following declarations:

$$
\begin{array}{c}
\forall : (\mathit{ty} \to \mathit{ty}) \to \mathit{ty} \\
\forall\text{-}\mathit{iso} : \{A : \mathit{ty} \to \mathit{ty}\} \to \mathit{tm}\ (\forall\ A) \cong ((\alpha : \mathit{ty}) \to \mathit{tm}\ (A\ \alpha))
\end{array}
$$

where $\_\cong\_$ is the judgement of isomorphisms (4.1∗7). Since polymorphic (and ordinary) functions in this signature are specified by function types of the logical framework, they inherit the β and η equalities of LF function types.

As an example, letting $Abs = \forall\text{-}iso.bwd$, we can define Church numerals:

$$\begin{aligned}&CNum : ty\\&CNum = \forall\ (\lambda\ \alpha \to (\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha)\\&C_2 : tm\ CNum\\&C_2 = Abs\ (\lambda\ \alpha \to abs\ (\lambda\ f \to abs\ (\lambda\ x \to app\ f\ (app\ f\ x))))\end{aligned}$$

4.1∗10. Let us compare the examples of LF presentations of type theories above and their more traditional gamma-and-turnstile presentations.

(1) In the LF, Σ-types are used to pack two judgements together. This is implicit in traditional presentations. For example, $(\Sigma(a : ty).\ tm\ a) \to J$ is just

$$\frac{a\ \text{type} \qquad t : a}{J}$$

(2) Equality types of the LF are used to specify the equational theory of the object type theory, and since equality types are respected by all constructions of the LF, there is no need to manually specify any congruence rules.

(3) A more noticeable difference is that dependent function types in the universe $\mathbb{J}$ (which may be called *higher-order judgements* as we call elements of $\mathbb{J}$ judgements) uniformly handles two different things in traditional presentations: *contexts of hypotheses* $\Gamma \vdash J$ and *schematic inference rules* $\frac{J}{K}$. Taking the rule of function abstraction in STLC for example, the traditional presentation is

$$\frac{\Gamma, x : a \vdash t : b}{\Gamma \vdash \lambda x.\ t : a \Rightarrow b}$$

while in the LF presentation (Example 4.1∗6), this rule is

$$abs : \{a, b : ty\} \to (tm\ a \to tm\ b) \to tm\ (a \Rightarrow b)$$

The higher-order judgement $tm\ a \to tm\ b$ corresponds to a deduction of $t : b$ with a new hypothesis $x : a$ in the context, and $(tm\ a \to tm\ b) \to tm\ (a \Rightarrow b)$ corresponds to the inference rule. In traditional presentations, contexts Γ can only contain certain basic judgements, such as $x : a$, but not for example

$$\Gamma, x : (a\ \text{type}) \vdash \cdots \qquad \text{or even} \qquad \Gamma, x : (\Gamma, x : a \vdash t : b) \vdash \cdots,$$

so there need to be two layers of entailment, $\Gamma \vdash J$ and $\frac{J}{K}$. In contrast, both of them are handled as higher-order judgements in the logical framework.

However, introducing higher-order judgements raises the question that whether a type theory defined in the LF is the same as its traditional presenta-

tion, since higher-order judgements *a priori* may introduce new terms to base judgements. This question is called the *adequacy* of LF presentations [Harper et al. 1993]. Using a gluing argument similar to the one by Gratzer and Sterling [2021], adequacy of the logical framework that we use here can be proven with respect to Uemura's [2021; 2023] logical framework, which can be seen as a faithful formulation of traditional presentations of type theories.

4.1∗11. Because of *extensional* equalities in $\mathbb{J}$, LccLF does not enjoy decidable type checking, so the type theory of LccLF cannot be implemented as a mechanised proof assistant as is. This is not a problem for us since we only use LccLF as a paper-and-pencil logical framework. The undecidability of LccLF is no different from the undecidability of the word problem of universal algebra [Boone 1958].

However, if we are after mechanically checking the signatures and terms defined in LccLF, we can 'approximate' LccLF using existing proof assistants implementing intensional equality types, such as Agda, with the axioms of uniqueness of identity proofs (UIP) and function extensionality (FUNEXT) as postulates, since extensional Martin-Löf type theory is conservative with respect to intensional Martin-Löf type theory augmented with UIP and FUNEXT [Hofmann 1995a; Kapulkin and Li 2023]. The (unavoidable) cost of this is that some extra transportations along intensional equalities must be inserted manually.

4.2 Functorial Semantics of Signatures

4.2∗1. Type theories, like other flavours of algebraic theories, or languages in general, are invented for *talking about things*, either mathematical objects or intuitive objects in 'the physical world'. Therefore a logical framework is incomplete if it does not provide a notion of *models* of signatures defined in it. In this section, we show how this is done for LccLF in Section 4.1 by means of *functorial semantics* à la Lawvere theories [Lawvere 1963], except that categories with finite products are replaced by locally cartesian closed categories.

4.2∗2. The syntax of LccLF, quotiented by judgemental equalities, forms a *category with families* (CwF) with the extra structures in Figure 4.1, henceforth called an LF-CwF. It can further be proven to be *initial* among all LF-CwFs, similarly to the initiality results of many other dependent type theories in the literature [de Boer 2020; Kaposi et al. 2020; Pitts 2001; Streicher 1991]. In outline, we first define a *partial* interpretation of the raw syntax for every LF-CwF by induction on the raw syntax, and then we show that the partial interpretation is defined on well typed terms by induction on the typing derivation, and finally we show that the interpretation respects judgemental equalities.

Alternatively, from a more abstract point of view, we can view the typing rules of LccLF as a *generalised algebraic theory* (GAT) [Cartmell 1986; Sterling 2019] or the signature of a *quotient inductive-inductive type* (QIIT) [Altenkirch et al. 2018; Kaposi et al. 2019; Kovács 2023]. Then we can directly take the initial model of this GAT or this QIIT as the definition of the syntax of LccLF. The existence of the initial model is shown by Cartmell [1986] in a set-theoretic metatheory and by Kaposi et al. [2019] in a type-theoretic metatheory. In this way, there is no need to prove initiality manually, since we are essentially using existing logical frameworks (GATs or QIITs) to define our logical framework. In fact, it is folklore that every elementary topos with a natural number object has finitary quotient inductive-inductive types; see Kovács [2023, §4.6] for more discussion.

4.2∗3. Either (i) by constructing the abstract syntax of LccLF from raw syntax and proving the initiality manually or (ii) by taking the initial model as the definition of the abstract syntax of the LF, in what follows we write LfSig for the category of LF-contexts and substitutions between contexts, i.e.,

$$\textsc{Obj}\,\textsc{LfSig} = \{\vdash \Gamma\ \mathrm{ctx}\} \qquad \textsc{Hom}_{\textsc{LfSig}}(\Delta, \Gamma) = \{\Delta \vdash \gamma : \Gamma\},$$

and $Ty_{LF} : \textsc{LfSig}^{\mathrm{op}} \to \textsc{Set}$ for the presheaf of LF-types over contexts,

$$Ty_{LF}(\Gamma) = \{\Gamma \vdash A\ \mathrm{type}\},$$

and $Tm_{LF} : (\int Ty_{LF})^{\mathrm{op}} \to \textsc{Set}$ for the presheaf of terms over the category of elements of Ty_{LF}, i.e. for every context Γ and $A \in Tm_{LF}(\Gamma)$,

$$Tm_{LF}(\Gamma; A) = \{\Gamma \vdash a : A\}.$$

The context extension of $\Gamma \in \textsc{LfSig}$ with $A \in Ty_{LF}(\Gamma)$ is written as just $\Gamma.A \in \textsc{LfSig}$, together with the projection substitution $p : \Gamma.A \to \Gamma$.

4.2∗4 Definition. For every signature S, i.e. a context, of LccLF, its *category of judgements* $\textsc{Jdg}\,S$ is the full subcategory of the slice category $\textsc{LfSig}/S$ spanned by projection maps $p : S.A \to S$ of context extensions for $S \vdash A : \mathbb{J}$.

4.2∗5. The objects of $\textsc{Jdg}\,S$ can be identified with judgements $S \vdash A : \mathbb{J}$ in the context of S; the morphisms $t : A \to B$ can be identified with functions $S \vdash f : A \to B$. Since the universe $\mathbb{J}$ is closed under precisely the connectives of extensional MLTT (1, Σ, Π, and extensional equality types), the category $\textsc{Jdg}\,S$ is the category of types for extensional MLTT with the additional constants from S.

Consequently, $\textsc{Jdg}\,S$ is locally cartesian closed. In every slice category $\textsc{Jdg}\,S/A$,

- the terminal object is $(\lambda a.\ a) : A \to A$, where we omit $S \vdash$ for clarity;

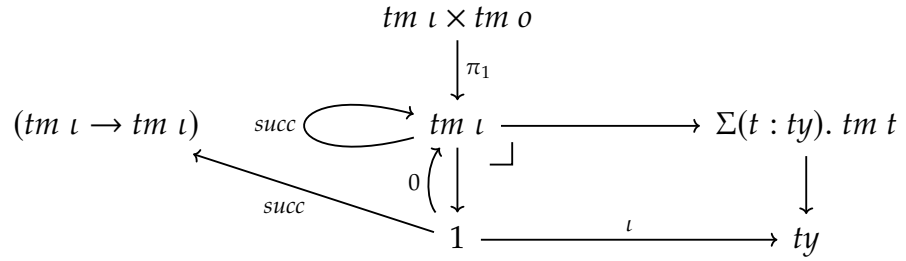
* the product of $f : B \to A$ and $g : C \to A$ is $\lambda p.\ f\ (\pi_1\ p) : P \to A$ where

$$P := \Sigma(b : B).\ \Sigma(c : C).\ (f\ b = g\ c);$$

* the exponential of $f : B \to A$ and $g : C \to A$ is $\pi_1 : E \to A$ where

$$\begin{aligned} E &:= \Sigma(a : A).\ B_a \to C_a, \\ B_a &:= \Sigma(b : B).\ f\ b = a, \\ C_a &:= \Sigma(c : C).\ g\ c = a. \end{aligned}$$

4.2∗6 Example. Consider the signature of PCF in Example 4.1∗8. Some examples of objects and morphisms of the category of judgements for PCF are



This category is not the same as the usual category of *contexts* for PCF, since it contains higher-order judgements such as $tm\ \iota \to tm\ \iota$ or $(tm\ \iota \to tm\ \iota) \to tm\ \iota$ that do not correspond to any PCF-contexts. However, the adequacy of the LF encoding of PCF implies that the category of PCF-contexts can be fully faithfully embedded in the category of judgements. Namely, it is the full subcategory spanned by finite products of objects in the set $\{tm\ A \mid A : ty\}$.

4.2∗7 Definition. Let S be a signature in LccLF and $\mathscr{C}$ a locally cartesian closed category (LCCC). A *(functorial) model of S in $\mathscr{C}$* is a functor $M : \textsc{Jdg}\, S \to \mathscr{C}$ that preserves the locally cartesian closed structure.

4.2∗8. Because the objects of the category of judgements $\textsc{Jdg}\, S$ are generated by a quite intricate induction, it is not straightforward to define a functorial model of S explicitly. We will solve this by using internal languages later, but for now let us sketch a partial example for some intuition.

The following lemma about presheaf categories is well known (see e.g. [nLab 2024d]) and will be used in the example.

4.2∗9 Lemma. For every small category $\mathscr{C}$ and presheaf $A : \mathscr{C}^{\mathrm{op}} \to \textsc{Set}$, there is an equivalence of categories $\textsc{Pr}\,\mathscr{C}/A \cong \textsc{Pr}\,(\int A)$ between the slice category $\textsc{Pr}\,\mathscr{C}/A$ and the presheaf category $\textsc{Pr}\,(\int A)$ over the category of elements of A.

4.2∗10 Example. Consider the signature of STLC in Example 4.1∗6. Let $\mathscr{C}$ be a small cartesian closed category. The category $\mathscr{C}$ has enough structure for

interpreting the category of *contexts* of STLC [Lambek and Scott 1986] but not enough for interpreting the category of *judgements* of STLC, since $\mathscr{C}$ may not be locally cartesian closed. However, we can interpret the category of judgement in the presheaf category $\textsc{Pr}\,\mathscr{C}$, which is always locally cartesian closed.

Firstly, we define $Ty \in \textsc{Pr}\,\mathscr{C}$ to be the constant presheaf

$$Ty(\Gamma) = \textsc{Obj}\,\mathscr{C} \qquad\qquad Ty(\gamma) = id,$$

and $Tm \in \textsc{Pr}\,\mathscr{C}$ to be the presheaf defined by

$$Tm(\Gamma) = \{(A, f) \mid A \in \textsc{Obj}\,\mathscr{C}, f : \Gamma \to A\}$$
$$Tm(\gamma) = (A, f) \mapsto (A, \Delta \xrightarrow{\gamma} \Gamma \xrightarrow{f} A)$$

for all $\Gamma, \Delta \in \mathscr{C}$ and $\gamma : \Delta \to \Gamma$. As the names suggest, the projection map $p : Tm \to Ty$ is going to be the interpretation of the morphism

$$\pi_1 : \Sigma(t : ty).\ tm\ t \to ty \in \textsc{Jdg}_{\text{STLC}}.$$

The interpretation of the declaration $\iota : ty$ can be any global element $1 \to Ty$, i.e. any object of $\mathscr{C}$. The interpretation of the declaration $\_\Rightarrow\_ : ty \to ty \to ty$ in $\textsc{Pr}\,\mathscr{C}$ is given by the adjunct of the natural transformation $F : Ty \times Ty \to Ty$:

$$F_\Gamma\,(A, B) = B^A \text{ for all } (A, B) \in (Ty \times Ty)\,(\Gamma).$$

To interpret the isomorphism pair *abs* and *app*, following the interpretation of MLTT in LCCCs [Hofmann 1997; Seely 1984], we need to construct in the slice category $\textsc{Pr}\,\mathscr{C}/Ty \times Ty$ an isomorphism between the object $F^*p : F^*Tm \to Ty \times Ty$, obtained by pulling back p along F,

$$\begin{array}{ccc} F^*Tm & \longrightarrow & Tm \\ {\scriptstyle F^*p}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ Ty \times Ty & \xrightarrow{F} & Ty \end{array}$$

and the object $\pi_1^*p \Rightarrow \pi_2^*p$, obtained by taking the exponential of π_1^*p and π_2^*p in $\textsc{Pr}\,\mathscr{C}/Ty \times Ty$, where π_i^*p is respectively obtained by the pullback

$$\begin{array}{ccc} \pi_i^*Tm & \longrightarrow & Tm \\ {\scriptstyle \pi_i^*p}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ Ty \times Ty & \xrightarrow{\pi_i} & Ty \end{array}$$

We can construct this isomorphism pointwise for each object $\Gamma \in \mathscr{C}$. An element of $Ty \times Ty\ (\Gamma)$ is a pair (A, B) of $\mathscr{C}$-objects, so the presheaf F^*Tm at Γ is the set

$$\{(A, B, f) \mid A, B \in \mathscr{C},\ f : \Gamma \to B^A\}. \tag{4.1}$$

The object $\pi_1^* p \Rightarrow \pi_2^* p$ is harder to compute, but by using Lemma 4.2∗9 and the end formula of exponentials in presheaf categories, we can compute that the fiber of $\pi_1^* p \Rightarrow \pi_2^* p$ over $(A, B) \in Ty \times Ty\,(\Gamma)$ is the set

$$
\begin{aligned}
& \textstyle\int_{\Delta \in \mathscr{C}} \prod_{\mathscr{C}(\Delta,\Gamma)} \mathscr{C}(\Delta, A) \Rightarrow \mathscr{C}(\Delta, B) \\
\cong \quad & \{\text{powering in } \textsc{Set} \text{ is the same as exponentiating}\} \\
& \textstyle\int_{\Delta \in \mathscr{C}} \mathscr{C}(\Delta, \Gamma) \Rightarrow \mathscr{C}(\Delta, A) \Rightarrow \mathscr{C}(\Delta, B) \\
\cong \quad & \{\text{by uncurrying}\} \\
& \textstyle\int_{\Delta \in \mathscr{C}} \mathscr{C}(\Delta, \Gamma) \times \mathscr{C}(\Delta, A) \Rightarrow \mathscr{C}(\Delta, B) \\
\cong \quad & \{\text{by the universal property of products}\} \\
& \textstyle\int_{\Delta \in \mathscr{C}} \mathscr{C}(\Delta, \Gamma \times A) \Rightarrow \mathscr{C}(\Delta, B) \\
\cong \quad & \{\text{by Yoneda embedding}\} \\
& \mathscr{C}(\Gamma \times A,\ B)
\end{aligned}
$$

which is indeed isomorphic to the fiber of F^*Tm (4.1) over (A, B) in a canonical way. We omit the verification of naturality here.

We have sketched the interpretations of the declarations of STLC from Example 4.1∗6 in $\textsc{Pr}\,\mathscr{C}$. These data can be in fact extended to an LCC-functor

$$M : \textsc{Jdg}_{\text{STLC}} \to \textsc{Pr}\,\mathscr{C},$$

but we will not do it here since we will study the general case below.

4.2∗11. If the reader is more interested in higher-order algebraic effects than the meta-theory of LccLF, they can now jump to Chapter 5, which will use Theorem 4.4∗10 below but only as a black box.

4.3 Diagrammatic Semantics of Signatures

4.3∗1. Since the category of judgement $\textsc{Jdg}\,S$ is comprised of syntactic entities that are inductively generated, it is natural to expect that $\textsc{Jdg}\,S$ has some universal property so that functorial models of S, i.e. LCC-functors $\textsc{Jdg}\,S \to \mathscr{C}$, correspond to certain structures in $\mathscr{C}$ that we may call *diagrammatic models* of S. The situation should generalise that of Lawvere theories – if $\textsc{Jdg}\,S$ is the Lawvere theory generated by a signature S, a product-preserving functor $\textsc{Jdg}\,S \to \mathscr{C}$ corresponds to an object in $\mathscr{C}$ equipped with the operations from the signature S.

Taking the example of STLC (Example 4.1∗6) again, it is natural to expect that

LCCC-functors $\textsc{Dg}_{\mathrm{STLC}} \to \mathscr{C}$ correspond to diagrams in $\mathscr{C}$ of the shape

$$\begin{array}{ccccc} \pi_1^* p \Rightarrow \pi_2^* p & \longrightarrow & Tm \\ \downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ Ty \times Ty & \xrightarrow{F} & Ty & \xleftarrow{\iota} & 1 \end{array}$$

where $\pi_1^* p \Rightarrow \pi_2^* p$ is the exponential of $\pi_1^* p$ and $\pi_2^* p$ in the slice $\mathscr{C}/Ty \times Ty$, and the square must be a pullback.

To define a notion of diagrammatic models in an LCCC $\mathscr{C}$ for every signature S, we need to perform an induction on the syntax of LccLF, since a signature S is nothing other than an LF context. An induction on the LF is the same as constructing a CwF with the type connectives of the LF.

However, we cannot directly use the LCCC $\mathscr{C}$ as the (underlying category of) the CwF, since $\mathscr{C}$ does not have the structure for interpreting the universe $\mathbb{J}$. The solution is, again, passing to the presheaf category $\textsc{Pr}\,\mathscr{C}$, in which we can construct a *universe* containing (the Yoneda embedding of) objects of $\mathscr{C}$.

4.3.1 Universes in Categories

4.3.1∗1. The concept of universes in *toposes* dates back to Bénabou [1973] and Maurer [1975], and was developed later by Streicher [2005]. A more general account of universes in categories is developed by Voevodsky [2015, 2017] and Kapulkin and Lumsdaine [2021] in the study of homotopy type theory; see also Gratzer [2023, §3.2, 3.3] for an excellent exposition.

In this subsection, we will have a brief digression on universes without going into much technical detail, ending with the theorem that a universe of $\mathscr{C}$-objects can be constructed in the presheaf category $\textsc{Pr}\,\mathscr{C}$ (Theorem 4.3.1∗11).

4.3.1∗2 Definition. A *universe* in a category $\mathscr{C}$ is simply a morphism $p : \tilde{U} \to U$ equipped with chosen pullbacks along every morphism $A : \Gamma \to U$:

$$\begin{array}{ccc} \Gamma.A & \longrightarrow & \tilde{U} \\ {\scriptstyle A^*p}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ \Gamma & \xrightarrow{A} & U \end{array}$$

A morphism $\Delta \to \Gamma$ in $\mathscr{C}$ is said to be *classified* by a universe $p : \tilde{U} \to U$ if it is a pullback of p along some (not necessarily unique) morphism $\Gamma \to U$.

A universe may additionally be equipped with logical structures such as Π-types and Σ-types; see [Kapulkin and Lumsdaine 2021, §1.4] or [Gratzer 2023, §3.2] for details. For example, *binary product* on a universe $p : \tilde{U} \to U$ is a pair of

morphisms $prod : U \times U \to U$ and $pair : \tilde{U} \times \tilde{U} \to \tilde{U}$ forming a pullback:

$$\begin{array}{ccc} \tilde{U} \times \tilde{U} & \xrightarrow{pair} & \tilde{U} \\ {\scriptstyle p \times p}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ U \times U & \xrightarrow{prod} & U \end{array} \tag{4.2}$$

4.3.1∗3. Universes abound in logic and type theory.

1. In the category of sets, every Grothendieck universe U determines a universe $\pi_1 : \tilde{U} \to U$ where $\tilde{U}$ is the set of pointed U-small sets:

   $$\tilde{U} = \{(A, a) \mid A \in U, a \in A\},$$

   and $\pi_1(A, a) = A$ is the projection function.

2. For a small category $\mathscr{C}$, every Grothendieck universe U of sets can be lifted to a universe $\pi_1 : \tilde{V} \to V$ in the presheaf category $\mathrm{Pr}\,\mathscr{C}$ by the *Hofmann-Streicher lifting* [Hofmann and Streicher 1999]: V maps every $\Gamma \in \mathscr{C}$ to the set of U-valued presheaves over $\mathscr{C}/\Gamma$, and $\tilde{V}$ maps every $\Gamma \in \mathscr{C}$ to the set

   $$\{(A, a) \mid A \in V(\Gamma), a : 1 \to A \in \mathrm{Pr}\,(\mathscr{C}/\Gamma)\}.$$

   The actions of V and $\tilde{V}$ on morphisms $\gamma : \Delta \to \Gamma$ is given by precomposing with the functor $(\gamma \cdot -) : (\mathscr{C}/\Delta)^{\mathrm{op}} \to (\mathscr{C}/\Gamma)^{\mathrm{op}}$.

   However, the universe V constructed in this way is not what we want for interpreting the universe of judgements $\mathbb{J}$, because V classifies all (U-small) presheaves rather than just (Yoneda-embedding of) $\mathscr{C}$-objects.

3. Liftings of Grothendieck universes of sets to sheaf toposes in general [Gratzer et al. 2022; Streicher 2005], categories of assemblies, and realizability toposes [Streicher 2005] also exist.

4. A syntactic example of universes is the map $\pi_1 : (A : \mathbb{J}, a : A) \to (A : \mathbb{J})$ in the category LFSig of LF-signatures (4.2∗3). The pullback of π_1 along an arbitrary morphism $B : S \to (A : \mathbb{J})$ can be chosen to be simply

   $$\begin{array}{ccc} S.B & \longrightarrow & (A : \mathbb{J}, a : A) \\ \downarrow & \lrcorner & \downarrow \\ S & \xrightarrow[B]{} & (A : \mathbb{J}) \end{array}$$

All the examples above can be equipped with structures of Σ-types, Π-types, and extensional equality types.

4.3.1∗4 Definition. For a universe $p : \tilde{U} \to U$ in a category $\mathscr{C}$, its *externalisation* $[p : \tilde{U} \to U]$, or simply $[U]$ when it causes no confusion, is the fibration over $\mathscr{C}$

whose fiber category $[U]_\Gamma$ over every object $\Gamma \in \mathscr{C}$ has as objects $\mathscr{C}$-morphisms $A : \Gamma \to U$. Morphisms $A \to B$ in the fiber $[U]_\Gamma$ are $\mathscr{C}$-morphisms $h : \Gamma.A \to \Gamma.B$ making the following diagram commute:

$$\begin{array}{ccc} \Gamma.A & \xrightarrow{h} & \Gamma.B \\ & {\scriptstyle A^*p}\searrow \quad \swarrow{\scriptstyle B^*p} & \\ & \Gamma & \end{array}$$

where $\Gamma.A$ and $\Gamma.B$ arise from pulling back p along A and B respectively. The reindexing functor $\gamma^* : [U]_\Gamma \to [U]_\Delta$ for a morphism $\gamma : \Delta \to \Gamma$ is precomposition:

$$y^*(\Gamma \xrightarrow{A} U) = (\Delta \xrightarrow{\gamma} \Gamma \xrightarrow{A} U).$$

4.3.1∗5. For an LF-signature S, the category $\textsc{Jdg}\, S$ of judgements of S from Definition 4.2∗4 is precisely the fiber over S of the externalisation of the universe

$$\pi_1 : (A : \mathbb{J}, a : A) \to (A : \mathbb{J})$$

in the category $\textsc{LfSig}$ of LF-contexts from 4.2∗3.

4.3.1∗6. When a universe is additionally equipped with logical structures, these structures can be carried over to the externalisation. For example, if the universe is equipped with binary product $prod : U \times U \to U$ as in (4.2), the cartesian product of two objects $A, B : \Gamma \to U$ in the fiber $[U]_\Gamma$ is

$$\Gamma \xrightarrow{\langle A,B\rangle} U \times U \xrightarrow{prod} U$$

Moreover, this choice of cartesian products are strictly preserved by the reindexing functor $\gamma^* : [U]_\Gamma \to [U]_\Delta$ since reindexing is defined to be precomposition.

If the universe U is equipped with Σ-types, Π-types, and extensional equality types, each fiber category of $[U]$ has an LCCC structure given in a way similar to 4.2∗5. The reindexing functor strictly preserves the LCCC structure.

4.3.1∗7. Example 4.2∗10 contains another example of universes: starting from a cartesian closed category $\mathscr{C}$, we constructed in the presheaf category $\textsc{Pr}\,\mathscr{C}$ a morphism $p : Tm \to Ty$, which can be succinctly defined as

$$\begin{array}{c} \coprod_{A \in \textsc{Obj}\,\mathscr{C}} \mathrm{Y}A \\ \Big\downarrow{\scriptstyle \coprod_{A \in \textsc{Obj}\,\mathscr{C}} !} \\ \coprod_{A \in \textsc{Obj}\,\mathscr{C}} 1 \end{array} \tag{4.3}$$

This morphism as a universe classifies (the Yoneda embedding of) all projection morphisms in $\mathscr{C}$: for every projection morphism $\Gamma \times A \to \Gamma$ in $\mathscr{C}$, we can choose

a morphism $\lceil A \rceil : \mathrm{Y}\Gamma \to Ty$ making a pullback square:

$$\begin{array}{ccc} \mathrm{Y}(\Gamma \times A) & \longrightarrow & Tm \\ {\scriptstyle \mathrm{Y}\pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ \mathrm{Y}\Gamma & \xrightarrow{\lceil A \rceil} & Ty \end{array}$$

The morphism $\lceil A \rceil$ can just be the one corresponding to the element $A \in Ty(\Gamma)$ by Yoneda lemma, but any object isomorphic to A in $\mathscr{C}$ is equally good, so p only *weakly* classifies projection morphisms of $\mathscr{C}$.

Conversely, for every $\Gamma \in \mathscr{C}$ and morphism $A : \mathrm{Y}\Gamma \to Ty$, the pullback of $p : Tm \to Ty$ along A is (isomorphic to) the Yoneda embedding of the projection map $\pi_1 : \Gamma \times A \to \Gamma$, where we identify the morphism A with an element of $Ty(\Gamma) = \textsc{Obj}\,\mathscr{C}$ by Yoneda lemma.

As a consequence of these observations, the fiber category $[Ty]_1$ of the externalisation over the terminal object is isomorphic to the category $\mathscr{C}$ itself, justifying the view of Ty as a universe of $\mathscr{C}$-objects.

Moreover, in Example 4.2∗10 we saw how exponentials in $\mathscr{C}$ can be lifted to the universe $p : Tm \to Ty$. However, p inherently only supports *simply typed* structures. For an arbitrary type family over Γ, i.e. an arbitrary morphism $f : B \to \Gamma$ in $\mathscr{C}$, there may be no morphism $\mathrm{Y}\Gamma \to Ty$ making a pullback square:

$$\begin{array}{ccc} \mathrm{Y}B & \longrightarrow & Tm \\ {\scriptstyle \mathrm{Y}f}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ \mathrm{Y}\Gamma & \dashrightarrow^{?} & Ty \end{array}$$

4.3.1∗8. The good news is that the dependently typed version of (4.3) exists. Perhaps unexpectedly, what we need is exactly Hofmann's [1995b] technique for turning locally cartesian closed categories into (strict) models of extensional MLTT. Hofmann op. cit. showed how to construct a CwF (with the type connectives of extensional MLTT) over an arbitrary LCCC $\mathscr{C}$, but as pointed out by Fiore [2012] and Awodey [2018], a CwF structure $\langle Ty, Tm \rangle$ over $\mathscr{C}$ is precisely a universe $p : Tm \to Ty$ in $\textsc{Pr}\,\mathscr{C}$ that is a *representable morphism*, a perspective known as *natural models* of dependent type theories [Awodey 2018; Newstead 2018].

4.3.1∗9 Definition. A morphism $p : Tm \to Ty$ in a presheaf category $\textsc{Pr}\,\mathscr{C}$ is called *representable* if for every $\Gamma \in \mathscr{C}$ and $A : \mathrm{Y}\Gamma \to Ty$, there is an object $\Gamma.A \in \mathscr{C}$ and a morphism $f : \Gamma.A \to \Gamma$ in $\mathscr{C}$ making a pullback square:

$$\begin{array}{ccc} \mathrm{Y}\Gamma.A & \longrightarrow & Tm \\ {\scriptstyle \mathrm{Y}f}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ \mathrm{Y}\Gamma & \xrightarrow{A} & Ty \end{array}$$

4.3.1∗10. Putting aside the type connectives, Hofmann's construction can be seen as an instance of Bénabou's [1975] construction from a fibration to an equivalent split fibration (see also Streicher [2023, Theorem 3.1] and Jacobs [1999, §5.2]), applied to the codomain fibration $\mathscr{C}^{\to} \to \mathscr{C}$ of an LCCC $\mathscr{C}$. This splitting technique is the right adjoint to the forgetful functor from split fibrations to fibrations. There is also a left adjoint due to Giraud [1965, I 2.4.3]; see also Streicher [2023, page 13]. Lumsdaine and Warren [2015] and Awodey [2018] showed that the left-adjoint splitting can also be used to construct (strict) models of type theories from weak models, known as the *local universe* construction.

Compared to Hofmann's construction, the local universe construction additionally supports inductive types without judgemental η-laws, such as *intensional* equality types. Although we do not need these types in this thesis, the local universe construction is worth a mention for its remarkable elegance: given a small category $\mathscr{C}$, the local universe is the following representable map:

$$\begin{array}{c} \coprod_{f \in \mathrm{Mor}\,\mathscr{C}} \mathrm{Y}(\mathrm{Dom}\, f) \\ \coprod_{f \in \mathrm{Mor}\,\mathscr{C}} \mathrm{Y}f \Big\downarrow \\ \coprod_{f \in \mathrm{Mor}\,\mathscr{C}} \mathrm{Y}(\mathrm{Cod}\, f) \end{array} \tag{4.4}$$

The logical structures on the LCCC $\mathscr{C}$, such as Π and Σ types, can be lifted to this representable map (as a universe) as well. Using either Hofmann's construction or the local universe construction, we have the following result.

4.3.1∗11 Theorem (Awodey 2018; Hofmann 1995b; Lumsdaine and Warren 2015). *For every small locally cartesian closed category $\mathscr{C}$, there is a universe $p : \tilde{U}_{\mathscr{C}} \to U_{\mathscr{C}}$ in* $\mathrm{Pr}\,\mathscr{C}$ *such that* (1) *p is representable*; (2) *every morphism $f : A \to B$ in $\mathscr{C}$ is classified by p*; (3) *p supports* 1, Σ, Π, *and extensional equality types.*

4.3.1∗12. In the situation of Theorem 4.3.1∗11, by the representability of p, the fiber $[U_{\mathscr{C}}]_1$ of the externalisation over the terminal object 1 contains only representable objects. And since p classifies (the Yoneda embedding of) all morphisms of $\mathscr{C}$, it in particularly classifies all morphisms $\mathrm{Y}A \to \mathrm{Y}1_{\mathscr{C}} = 1$. Therefore the fiber category $[U_{\mathscr{C}}]_1$ is equivalent to the LCCC $\mathscr{C}$.

4.3.2 Diagrammatic Models

4.3.2∗1. By Theorem 4.3.1∗11, $\mathrm{Pr}\,\mathscr{C}$ has the structure for interpreting the universe of judgements $\mathbb{J}$ and the type connectives on $\mathbb{J}$. The presheaf category $\mathrm{Pr}\,\mathscr{C}$ can interpret the unit type and Σ-types of the LF as usual [Hofmann 1997]. Thus we can turn $\mathrm{Pr}\,\mathscr{C}$ into an LF-CwF. Since the abstract syntax of the LF is initial among all LF-CwFs (4.2∗2), there is a unique LF-CwF homomorphism, which consists

of (1) a functor interpreting LF-signatures (i.e. LF-contexts) as $\mathscr{C}$-presheaves

$$[\![-]\!] : \textsc{LfSig} \longrightarrow \textsc{Pr}\,\mathscr{C} \tag{4.5}$$

and (2) mappings from LF-types/terms to $\textsc{Pr}\,\mathscr{C}$-types/terms that strictly preserve all operations, which we also denote by $[\![-]\!]$.

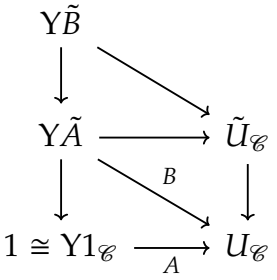
4.3.2∗2 Definition. A *diagrammatic model* of an LF-signature S in an LCCC $\mathscr{C}$ is a global element $m : 1 \to [\![S]\!]$ of the interpretation of S in $\textsc{Pr}\,\mathscr{C}$.

4.3.2∗3. Diagrammatic models are more ergonomic to work with than functorial models because $\textsc{Pr}\,\mathscr{C}$ as a presheaf topos has a very rich structure that we can manipulate using a type theoretic language. In this way, a diagrammatic model of S in $\mathscr{C}$ is a closed element of the record type $[\![S]\!]$ in the internal language of $\textsc{Pr}\,\mathscr{C}$, containing all fields of S and with $\mathbb{J}$ replaced by its interpretation $U_{\mathscr{C}}$.

4.3.2∗4 Example. The signature of barebone type theory from Example 4.1∗5 is interpreted as the presheaf $[\![\mathrm{BTT}]\!] \in \textsc{Pr}\,\mathscr{C}$ denoted by the record type

$$\begin{aligned}&\textsc{record}\ [\![\mathrm{BTT}]\!]\ \textsc{where}\\ &\quad ty\ : U_{\mathscr{C}}\\ &\quad tm : ty \to U_{\mathscr{C}}\end{aligned}$$

A closed element of this record consists of (1) a morphism $A : 1 \to U_{\mathscr{C}}$, which gives rise to an object $\tilde{A}$ in $\mathscr{C}$ by the representability of $\tilde{U}_{\mathscr{C}} \to U_{\mathscr{C}}$, and (2) a morphism $B : \mathrm{Y}\tilde{A} \to U_{\mathscr{C}}$ which gives rise to a morphism $\tilde{B} \to \tilde{A}$ in $\mathscr{C}$:



Thus a diagrammatic model of BTT in a locally cartesian closed category $\mathscr{C}$ gives rise to a morphism $\tilde{B} \to \tilde{A}$ in $\mathscr{C}$.

4.3.2∗5. In Definition 4.2∗4, the category $\textsc{Jdg}\,S$ of judgements for a signature S is defined to be the full subcategory of the slice $\textsc{LfSig}/S$ spanned by projections $S.A \to S$ for judgements $S \vdash A : \mathbb{J}$. Every such judgement is sent by the interpretation functor $[\![-]\!] : \textsc{LfSig} \to \textsc{Pr}\,\mathscr{C}$ (4.5) to a morphism $[\![A]\!] : [\![S]\!] \to U_{\mathscr{C}}$, which is exactly an object in the fiber of the externalisation $[U_{\mathscr{C}}]$ over $[\![S]\!]$. Since $[\![-]\!]$ is a homomorphism of LF-CwFs, it (strictly) preserves context extensions, so

the context projection $S.A \to S$ is sent to the morphism $[\![S]\!].[\![A]\!] \to [\![S]\!]$ in $\mathrm{Pr}\,\mathscr{C}$:

$$\begin{array}{ccc} S.A & \longrightarrow & (B : \mathbb{J}, b : B) \\ \downarrow & \lrcorner & \downarrow \\ S & \xrightarrow[A]{} & (B : \mathbb{J}) \end{array} \quad \overset{[\![-]\!]}{\longmapsto} \quad \begin{array}{ccc} [\![S]\!].[\![A]\!] & \longrightarrow & \tilde{U}_{\mathscr{C}} \\ \downarrow & \lrcorner & \downarrow \\ [\![S]\!] & \xrightarrow[[\![A]\!]]{} & U_{\mathscr{C}} \end{array}$$

Therefore $[\![-]\!]$ sends the morphisms of $\textsc{Jdg}\, S$ to morphisms of $[U_{\mathscr{C}}]_{[\![S]\!]}$ as well. In conclusion, we have a functor for every LCCC $\mathscr{C}$

$$\mathrm{G}_{\mathscr{C}} : \textsc{Jdg}\, S \longrightarrow [U_{\mathscr{C}}]_{[\![S]\!]},$$

which preserves LCCC structures since the type connectives giving the LCCC structures of $\textsc{Jdg}\, S$ 4.2∗5 are preserved by interpretation.

4.3.2∗6 Definition. Every diagrammatic model $m : 1 \to [\![S]\!]$ in an LCCC $\mathscr{C}$ determines a functorial model $F(m) := m^* \circ \mathrm{G}_{\mathscr{C}} : \textsc{Jdg}\, S \to \mathscr{C}$ by composing $\mathrm{G}_{\mathscr{C}}$ with the reindexing functor $m^* : [U_{\mathscr{C}}]_{[\![S]\!]} \to [U_{\mathscr{C}}]_1 \cong \mathscr{C}$:

$$\begin{array}{ccc} \textsc{Jdg}\, S & \xrightarrow{\mathrm{G}_{\mathscr{C}}} & [U_{\mathscr{C}}]_{[\![S]\!]} \\ & \searrow^{F(m)} & \downarrow m^* \\ & & \mathscr{C} \cong [U_{\mathscr{C}}]_1 \end{array}$$

4.3.2∗7. Using $F(m)$ will be our go-to way of constructing functorial models in the rest of the thesis. Of course, if the LCCC $\mathscr{C}$ that we start with already has a universe U that can model $\mathbb{J}$, for example, when $\mathscr{C}$ is a presheaf topos, then we may also directly construct diagrammatic models classified by U in the language of $\mathscr{C}$ itself, skipping the step of embedding $\mathscr{C}$ into $\mathrm{Pr}\,\mathscr{C}$.

4.3.3 From Functorial Models to Diagrammatic Models

4.3.3∗1. Unsurprisingly, from the other direction a functorial model $\textsc{Jdg}\, S \to \mathscr{C}$ induces a diagrammatic model as well, although this direction is more involved.

4.3.3∗2 Lemma. Let S be a signature and $\mathscr{C}$ an LCCC. Every functorial model $M : \textsc{Jdg}\, S \to \mathscr{C}$ determines a diagrammatic model $D(M) : 1 \to [\![S]\!]$ and a natural isomorphism $\phi_M : F(D(M)) \cong M : \textsc{Jdg}\, S \to \mathscr{C}$.

4.3.3∗3. Before proving 4.3.3∗2, we first observe that every LF-signature S is isomorphic (in the category $\textsc{LfSig}$ of LF-signatures) to a *standard signature*, which is inductively defined to be either

1. the empty signature,
2. a signature $(T, a : A)$ for some standard T and $T \vdash A : \mathbb{J}$, or

3. a signature $(T, B : A \to \mathbb{J})$ for some standard T and $T \vdash A : \mathbb{J}$.

Moreover, every judgement $S \vdash A : \mathbb{J}$ in the context of a standard signature S is equal (in the equational theory of the LF) to a *standard judgement*, which is inductively defined to be either

1. $S \vdash B\ a : \mathbb{J}$ for some declaration $B : A \to \mathbb{J}$ in S and $S \vdash a : A$, or
2. the type formers 1, Σ, Π, $a = b$ of $\mathbb{J}$ applied to standard judgements.

These claims can be shown by induction on the syntax of the LF. Note that they are not the same as *normalisation* of the LF – we do not claim terms have any standard or normal form (which is in indeed not true because *extensional* equality types do not enjoy normalisation [Hofmann 1995a, §3.2.2]).

When proving statements $P(S)$ about LF-signatures S that are invariant under isomorphisms or defining constructions $C(S)$ for LF-signatures that can be transported along isomorphisms, we conveniently only need to prove or construct for standard signatures and consider only standard judgements.

Proof of 4.3.3∗2. The required constructions $D(M)$ and ϕ_M in the statement of Lemma 4.3.3∗2 can be transported along isomorphisms $S \cong S'$of LF-signatures, so we can assume S is standard. Then we construct $D(M)$, the components of ϕ_M, and show the naturality of ϕ_M by a simultaneous induction on the structure of standard signatures S, standard judgements A in S, and terms $S \vdash a : A$.

Part 1. We first construct D for every signature S.

Case 1.1. If S is the empty signature, $[\![S]\!]$ is the terminal presheaf, and thus there is a unique choice for $D(M) : 1 \to [\![S]\!]$.

Case 1.2. If $S = (T, a : A)$ for some $T \vdash A : \mathbb{J}$, we have an inclusion functor $i : \textsc{Jdg}\, T \to \textsc{Jdg}\, S$ that sends every judgement $(T \vdash B : \mathbb{J}) \in \textsc{Jdg}\, T$ to its weakening $(S \vdash B : \mathbb{J}) \in \textsc{Jdg}\, S$. By composing this functor with $M : \textsc{Jdg}\, S \to \mathscr{C}$ we have a functorial model $M \circ i : \textsc{Jdg}\, T \to \mathscr{C}$ of T, which further gives rise to a diagrammatic model $D(M \circ i)$ of T by induction. Our goal is to construct a morphism $D(M) : 1 \to [\![S]\!]$ making the left triangle below commute:

$$\begin{array}{ccccc} & & [\![S]\!] = [\![T]\!].[\![A]\!] & \longrightarrow & \tilde{U}_{\mathscr{C}} \\ & \overset{\cdots\nearrow}{} & \downarrow{\scriptstyle p_S} \lrcorner & & \downarrow{\scriptstyle p} \\ 1 & \xrightarrow[D(M\circ i)]{} & [\![T]\!] & \xrightarrow[[\![A]\!]]{} & U_{\mathscr{C}} \end{array} \tag{4.6}$$

In the category $\textsc{Jdg}\, S$, we have a morphism

$$(T, a : A, 1 \vdash a : A) : (S \vdash 1 : \mathbb{J}) \to (S \vdash A : \mathbb{J}) \tag{4.7}$$

which is mapped by $M : \textsc{Jdg}\, S \to \mathscr{C}$ to a morphism

$$M(a) : 1_{\mathscr{C}} \cong M(S \vdash 1 : \mathbb{J}) \to M(S \vdash A : \mathbb{J})$$

in $\mathscr{C} \cong [U_{\mathscr{C}}]_1$. By the inductive hypothesis, we have a natural isomorphism

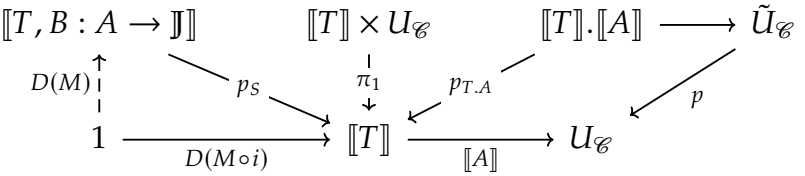
$$\phi_{M \circ i} : F(D(M \circ i)) \cong M \circ i,$$

so $M(i(T \vdash A : \mathbb{J}))$, which is exactly $M(S \vdash A : \mathbb{J})$, is isomorphic to the pullback of $p : \tilde{U}_{\mathscr{C}} \to U_{\mathscr{C}}$ along $[\![A]\!] \cdot D(M \circ i)$, giving rise to pullback squares as below:

$$\begin{array}{ccccc} M(S \vdash A : \mathbb{J}) & \xrightarrow{g} & [\![S]\!] = [\![T]\!].[\![A]\!] & \longrightarrow & \tilde{U}_{\mathscr{C}} \\ {\scriptstyle M(a)}\uparrow \downarrow & \lrcorner & \downarrow{\scriptstyle p_S} & \lrcorner & \downarrow{\scriptstyle p} \\ M(S \vdash 1 : \mathbb{J}) \cong 1 & \xrightarrow[D(M \circ i)]{} & [\![T]\!] & \xrightarrow[[\![A]\!]]{} & U_{\mathscr{C}} \end{array} \tag{4.8}$$

We let the desired diagrammatic model $D(M)$ be $g \cdot M(a) : 1 \to [\![S]\!]$.

Case 1.3. If $S = (T, B : A \to \mathbb{J})$ for some $T \vdash A : \mathbb{J}$, the projection morphism $p_S : [\![T, B : A \to \mathbb{J}]\!] \to [\![T]\!]$ is the exponential object in the slice category $\mathrm{Pr}\,\mathscr{C}/[\![T]\!]$ from $p_{T.A} : [\![T]\!].[\![A]\!] \to [\![T]\!]$ to $\pi_1 : [\![T]\!] \times U_{\mathscr{C}} \to [\![T]\!]$. We would like to define $D(M) : 1 \to [\![T, B : A \to \mathbb{J}]\!]$ making the left triangle below commute,



where $i : \textsc{Jdg}\, T \to \textsc{Jdg}\, S$ is the weakening functor, and $D(M \circ i)$ is the diagrammatic model of the smaller context T obtained from the inductive hypothesis. By the universal property of $[\![T, B : A \to \mathbb{J}]\!]$ as the exponential, we need to construct a morphism of $D(M \circ i) \times p_{T.A} \to \pi_1$ in the slice category over $[\![T]\!]$. The product $D(M \circ i) \times p_{T.A}$ is the pullback of $p_{T.A}$ along $D(M \circ i)$, which is isomorphic to the object $M(i(T \vdash A : \mathbb{J}))$ via $\phi_{M \circ i} : F(D(M \circ i)) \cong M \circ i$:

$$\begin{array}{ccccc} M(i(T \vdash A : \mathbb{J})) & \longrightarrow & [\![S]\!] = [\![T]\!].[\![A]\!] & \longrightarrow & \tilde{U}_{\mathscr{C}} \\ \downarrow & \lrcorner & {\scriptstyle p_{T.A}}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ 1 & \xrightarrow[D(M \circ i)]{} & [\![T]\!] & \xrightarrow[[\![A]\!]]{} & U_{\mathscr{C}} \end{array}$$

Now our goal is to construct a morphism $M(i(T \vdash A : \mathbb{J})) \to U_{\mathscr{C}}$.

Back in the category $\textsc{Jdg}\, S$, we have the judgement $S \vdash \Sigma\, A\, B : \mathbb{J}$, and the projection $\pi_1^{\Sigma AB} : \Sigma\, A\, B \to A$; henceforth we omit the context $S \vdash$ on objects. The projection map is sent by $M : \textsc{Jdg}\, S \to \mathscr{C} \cong [U_{\mathscr{C}}]_1$ to a morphism in $\mathscr{C}$. Since the universe $p : \tilde{U}_{\mathscr{C}} \to U_{\mathscr{C}}$ (weakly) classifies all $\mathscr{C}$-morphisms, the morphism

$M\pi_1^{\Sigma AB}$ gives us some $\ulcorner B \urcorner : M(A) \to U_{\mathscr{C}}$ and a pullback square:

$$\begin{array}{ccc} M(\Sigma\ A\ B) & \longrightarrow & \tilde{U}_{\mathscr{C}} \\ {\scriptstyle M\pi_1^{\Sigma AB}}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ M(A) & \xrightarrow[\ulcorner B \urcorner]{} & U_{\mathscr{C}} \end{array} \tag{4.9}$$

The morphism $\ulcorner B \urcorner$ fulfils our goal $M(i(T \vdash A : \mathbb{J})) = M(A) \to U_{\mathscr{C}}$.

Part 2. Now we define the component of $\phi_M : FDM \cong M$ at every (standard) judgement in a (standard) signature S.

Case 2.1. For the judgement $S \vdash B\ a : \mathbb{J}$ where $S = (T, B : A \to \mathbb{J}, R)$ for some T and R, $T \vdash A : \mathbb{J}$, and $S \vdash a : A$, we have a pullback diagram in $\textsc{Jdg}\, S$:

$$\begin{array}{ccc} B\ a & \longrightarrow & \Sigma\ A\ B \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \pi_1^{\Sigma AB}} \\ 1 & \xrightarrow[a]{} & A \end{array}$$

Since M is an LCCC functor, it preserves pullbacks and the terminal object, so we have a pullback square in $\mathscr{C}$:

$$\begin{array}{ccc} M(B\ a) & \longrightarrow & M(\Sigma\ A\ B) \\ \downarrow & \lrcorner & \downarrow{\scriptstyle M\pi_1^{\Sigma AB}} \\ 1 & \xrightarrow[Ma]{} & M(A) \end{array}$$

On the other hand, in Case 1.3 above, we have defined the B-component of the diagrammatic model $D(M)$ to be the code $\ulcorner B \urcorner$ of the morphism $M\pi_1^{\Sigma AB}$ as in the diagram (4.9). Hence, unfolding the definition of F, $FDM(B\ a)$ will be the pullback of $p : \tilde{U}_{\mathscr{C}} \to U_{\mathscr{C}}$ along the following morphism:

$$1 \xrightarrow{F(D(M))(a)} (FDM)(A) \xrightarrow{(\phi_M)_A} MA \xrightarrow{\ulcorner B \urcorner} U_{\mathscr{C}}$$

Using the naturality of $(\phi_M)_A$, this morphism is the same as $1 \xrightarrow{M(a)} MA \xrightarrow{\ulcorner B \urcorner} U_{\mathscr{C}}$. Therefore both $M(A)$ and $FDM(A)$ are the pullback of p along $\ulcorner B \urcorner \cdot M(a)$, so they are isomorphic in a canonical way.

Case 2.2. For a judgement A that is some type former of $\mathbb{J}$ applied to (smaller) standard judgements, $M(A)$ and $D(F(M))(A)$ are both LCC-functors so they preserve these type formers. By the universal properties of these type formers, $M(A)$ and $D(F(M))(A)$ are isomorphic in a canonical way.

Part 3. We also need to show that the family of morphisms

$$\phi_M(A) : FDM(A) \cong M(A)$$

is natural in $A \in \textsc{Jdg}\, S$. Because $\textsc{Jdg}\, S$ has exponentials, which are preserved by

M and FDM, it is sufficient to show that for every $S \vdash A : \mathbb{J}$ and $S \vdash a : A$, there is a commutative triangle:

$$\begin{array}{ccc} 1 & \xrightarrow{FDM(a)} & FDM(A) \\ & \searrow^{M(a)} & \downarrow{\scriptstyle (\phi_M)_A} \\ & & M(A) \end{array}$$

If a is a variable, this follows from the definition of $D(T, a : A)$ in Case 1.2 above. If a is other term formers, it follows from the fact that M and FDM as LCC-functors both preserve these term formers. □

4.4 Equivalence of Functorial and Diagrammatic Models

4.4∗1. We have now mappings F (Definition 4.3.2∗6) from diagrammatic models to functorial models, and vice versa D (Lemma 4.3.3∗2). Moreover, there is an isomorphism $F(D(M)) \cong M$ for every functorial model $M : \textsc{Jdg}\, S \to \mathscr{C}$. Naturally, we would expect $D(F(m)) \cong m$ for every diagrammatic model m as well, and then diagrammatic and functorial models will be in bijection up to isomorphisms. But we do not have a notion of isomorphisms of diagrammatic models yet, so we will define it in this section, which turns out to be more interesting a task than it sounds.

4.4∗2 Example. Let us still begin with some small examples for intuition. Consider the signature $(A : \mathbb{J})$ of one judgement and nothing else. Diagrammatic models of it in an LCCC $\mathscr{C}$ are morphisms $m : 1 \to U_{\mathscr{C}}$ in $\textsc{Pr}\,\mathscr{C}$, which give rise to objects A in $\mathscr{C}$ by Theorem 4.3.1∗11 in a surjective way (but different m may give rise to the same object in $\mathscr{C}$). An isomorphism between two models m_1 and m_2 in this case ought to be an isomorphism $i : A_1 \to A_2$ in $\mathscr{C}$ between the $\mathscr{C}$-objects A_1 and A_2 induced by m_1 and m_2 respectively.

Now suppose the signature is extended to $(A : \mathbb{J}, f : (A \to A) \to A)$. Then every diagrammatic model gives rise to an object A together with a morphism $f : (A \Rightarrow A) \to A$ in $\mathscr{C}$, where $A \Rightarrow A$ denotes the exponential. Now an isomorphism between two diagrammatic models should be a $\mathscr{C}$-isomorphism $i : A_1 \to A_2$ that commutes with f:

$$\begin{array}{ccc} A_1 \Rightarrow A_1 & \xrightarrow{f_1} & A_1 \\ {\scriptstyle i^{-1} \Rightarrow i}\downarrow & & \downarrow{\scriptstyle i} \\ A_2 \Rightarrow A_2 & \xrightarrow[f_2]{} & A_2 \end{array}$$

The fact that we have dependent functions in LF signatures is why we only consider isomorphisms rather than homomorphisms of diagrammatic models.

Suppose that the signature is further extended with a family of judgements $B : A \to \mathbb{J}$ indexed by A. A diagrammatic model now further induces an $\mathscr{C}$-object B with a morphism $g : B \to A$. An isomorphism of diagrammatic models should now further include a $\mathscr{C}$-isomorphism $j : B_1 \to B_2$ that commutes with i:

$$\begin{array}{ccc} B_1 & \xrightarrow{j} & B_2 \\ {\scriptstyle g_1}\downarrow & & \downarrow{\scriptstyle g_2} \\ A_1 & \xrightarrow[i]{} & A_2 \end{array}$$

Finally, if a signature is extended with a declaration of an equation, the notion of isomorphisms between diagrammatic models should remain unchanged, since in LCCCs there is not any higher-dimensional coherence between equalities.

4.4∗3. Diagrammatic models in an LCCC $\mathscr{C}$ are defined by interpreting an LF-signature S as a presheaf $[\![S]\!] \in \mathrm{Pr}\,\mathscr{C}$ (Definition 4.3.2∗2). In the internal language of the presheaf topos $\mathrm{Pr}\,\mathscr{C}$, a presheaf is a 'set'. Now that we are interested in isomorphisms of models, sets are no longer sufficient, and instead, we would like to use groupoids as our interpretation. More precisely, we plan to interpret every LF-signature S as an *internal groupoid* in $\mathrm{Pr}\,\mathscr{C}$:

$$\begin{array}{ccccc} & & \overset{inv}{\circlearrowright} & & \\ [\![S]\!]^{\cong} \times_{[\![S]\!]} [\![S]\!]^{\cong} & \xrightarrow[comp]{} & [\![S]\!]^{\cong} & \xleftarrow[id]{} & [\![S]\!] \\ & & \downarrow{\scriptstyle \langle s,t \rangle} & & \\ & & [\![S]\!] \times [\![S]\!] & & \end{array}$$

whose object part is exactly the presheaf $[\![S]\!]$ in the earlier interpretation (4.3.2∗1). Then, global elements $i : 1 \to [\![S]\!]^{\cong}$ of the morphism part of the groupoid will be defined as isomorphisms between diagrammatic models $s \cdot i$ and $t \cdot i : 1 \to [\![S]\!]$.

4.4∗4. One way to carry out the plan above is to follow Hofmann and Streicher's [1998] celebrated *groupoid model* of Martin-Löf type theory internally in the language of the presheaf topos $\mathrm{Pr}\,\mathscr{C}$, rather than in the ambient set theory.

In outline, in the language of $\mathrm{Pr}\,\mathscr{C}$, there is a groupoid $U_{\mathscr{C}}^{\cong}$ whose objects have the type $U_{\mathscr{C}}$ and the morphisms between $A, B : U_{\mathscr{C}}$ have the type of isomorphisms $\tilde{U}_{\mathscr{C}}(A) \cong \tilde{U}_{\mathscr{C}}(B)$, where $\tilde{U}_{\mathscr{C}}$ is the decoding type family for the universe $p : \tilde{U}_{\mathscr{C}} \to U_{\mathscr{C}}$. This groupoid provides the interpretation for the universe $\mathbb{J}$. For every $A : U_{\mathscr{C}}$, the type $\tilde{U}_{\mathscr{C}}(A)$ can be regarded as a *discrete* groupoid, so $U_{\mathscr{C}}^{\cong}$ still models Π, Σ, and *extensional* equalities. Π and Σ types outside $\mathbb{J}$ in the LF are interpreted in the same way as Hofmann and Streicher [1998]. The result of this construction would then be an *internal* LF-CwF in $\mathrm{Pr}\,\mathscr{C}$,

whose externalisation over the terminal object $1 \in \mathrm{Pr}\,\mathscr{C}$ will then the (ordinary) LF-CwF that interprets an LF-signature S as a groupoid of diagrammatic models.

4.4∗5. While the approach outlined above is feasible, there is a more direct approach that we will follow instead. First, we notice that in virtually all (many-sorted) logical frameworks the notions of homomorphisms/isomorphisms of models of a theory S can be expressed as another theory that contains (1) two copies of the declarations of S and (2) new declarations for the homomorphisms/isomorphisms between the basic sorts of the two copies of S, together with (3) equations asserting the homomorphic properties. This is also the case for LccLF. Take the signature $S = (A : \mathbb{J}, f : (A \to A) \to A)$ for example; in the LF we have the following signature $S^{\cong}$ of *isomorphisms of S-models*:

$$A_1 : \mathbb{J} \qquad f_1 : (A_1 \to A_1) \to A_1 \qquad A_2 : \mathbb{J} \qquad f_2 : (A_2 \to A_2) \to A_2$$
$$i : A_1 \cong A_2 \qquad\qquad \_ : (i.fwd \cdot f_1) = (\lambda g.\ f_2\ (i.fwd \cdot g \cdot i.bwd))$$

where $A_1 \cong A_2$ is the judgement of isomorphisms defined in 4.1∗7 and $f \cdot g$ means function composition $\lambda x.\ f\ (g\ x)$. In the category LfSig of LF-signatures, there are two morphisms $s, t : S^{\cong} \to S$ that project out (A_1, f_1) and (A_2, f_2) respectively. Similarly, we have morphisms *inv*, *id*, *comp* in LfSig for the inverse, identity, composition of S-isomorphisms, assembling to an internal groupoid:

$$S^{\cong} \times_S S^{\cong} \xrightarrow[comp]{} S^{\cong} \xleftarrow[id]{} S \qquad (inv : S^{\cong} \to S^{\cong}) \qquad S^{\cong} \xrightarrow{\langle s,t \rangle} S \times S$$

where $S^{\cong} \times_S S^{\cong}$ is the signature of three copies of S and two isomorphisms in between. The interpretation of this internal groupoid by $[\![-]\!] : \mathrm{LfSig} \to \mathrm{Pr}\,\mathscr{C}$ for every LCCC $\mathscr{C}$ is precisely the internal groupoid that we wanted in 4.4∗3.

Motivated by the discussion above, we would like to define an internal groupoid $\langle S, S^{\cong} \rangle$ for every LF-signature S, which can be done inductively together with some related conditions on judgements and terms in a signature.

4.4∗6 Lemma. The syntax of LccLF satisfies the following statements:

1. Every LF-signature S determines a judgement of *S-isomorphisms*

   $$M_1 : S,\ M_2 : S \vdash S^{\cong} : \mathbb{J}$$

   in which we reuse the name S for the iterative Σ-type of the fields of S. Moreover, there are terms *inv*, *id*, *comp* of the following types and they

satisfy (definitionally in the LF) the axioms of groupoids:

$$\begin{aligned} M_1, M_2 : S \vdash \ & inv_S \ : S^{\cong}[M_1, M_2] \to S^{\cong}[M_2, M_1] \\ M_1 : S \vdash \ & id_S \ : S^{\cong}[M_1, M_1] \\ M_1, M_2, M_3 : S \vdash \ & comp_S : S^{\cong}[M_1, M_2] \to S^{\cong}[M_2, M_3] \to S^{\cong}[M_1, M_3] \end{aligned}$$

where square brackets mean substitution.

2. Every judgement $S \vdash A : \mathbb{J}$ over a signature S determines a term coe_A for *coercing* along S-isomorphisms

$$M_1, M_2 : S \vdash coe_A : S^{\cong}[M_1, M_2] \to A[M_1] \to A[M_2]$$

such that coe_A is functorial, i.e., it preserves id_S and $comp_S$ of $S^{\cong}$. Conceptually, this amounts to say that every judgement in S determines a $\mathbb{J}$-valued presheaf over the groupoid $S^{\cong}$ in the LF.

3. Every term $S \vdash a : A$ of a judgement $S \vdash A : \mathbb{J}$ over a signature S determines a term coh_a showing the *coherence* of coercion:

$$M_1, M_2 : S \vdash coh_a : (i : S^{\cong}[M_1, M_2]) \to coe_A \ i \ a[M_1] = a[M_2]$$

Conceptually, this amounts to say that every term a determines a natural transformation from the constant $\mathbb{J}$-presheaf 1 to the presheaf A.

Proof. Following 4.3.3*3, it is sufficient to show these statements only for standard signatures S and standard judgements, since the judgements or terms required by these statements can be transported along isomorphisms of LF-signatures. We show these statements simultaneously by induction on the structure of standard signatures, standard judgements, and terms.

Part 1. We start with defining a groupoid $S^{\cong}$ for every signature S, with the inductive hypothesis that all the claims above about signatures, judgements and terms are already shown for structurally smaller cases.

Case 1.1. If S is the empty signature, we let $S^{\cong}$ be the unit type 1 together with the trivial groupoid structure.

Case 1.2. If $S = (T, a : A)$ for some $T \vdash A : \mathbb{J}$, in the context of $M_1, M_2 : S$, we will write $M_i.T$ and $M_i.a$ for the first and second projections of $M_i : S$, and we let $M_1, M_2 : S \vdash S^{\cong} : \mathbb{J}$ be the judgement

$$\Sigma(i : T^{\cong}[M_1.T, M_2.T]). \ \big(coe_A[M_1.T, M_2.T] \ i \ M_1.a = M_2.a\big).$$

The groupoid structure on this $S^{\cong}$ is defined by that of $T^{\cong}$ and the assumption that coe_A preserves the structure id_T and $comp_T$ of $T^{\cong}$. Conceptually, this definition of $S^{\cong}$ is the category of elements for the presheaf determined by $T \vdash A : \mathbb{J}$.

Case 1.3. If $S = (T, B : A \to \mathbb{J})$ for some $T \vdash A : \mathbb{J}$, in the context of $M_1, M_2 : S$, we will write $A_1, A_2 : \mathbb{J}$ for $A[M_1]$ and $A[M_2]$ respectively and similarly B_1 and

B_2 for $B[M_1]$ and $B[M_2]$ respectively. We define $M_1, M_2 : S \vdash S^{\cong} : \mathbb{J}$ to be

$$\Sigma(i : T^{\cong}[M_1.T, M_2.T]).\ \big((a_1 : A_1) \to (B_1\ a_1 \cong B_2\ (coe_A\ i\ a_1))\big)$$

The groupoid structure on this $S^{\cong}$ is defined using that of $T^{\cong}$ and the evident groupoid structure on isomorphisms of judgements in $\mathbb{J}$.

Part 2. Now we construct coe_A for every possibility of standard judgements.

Case 2.1. If the judgement is $S \vdash B\ a : \mathbb{J}$ for some variable $B : A \to \mathbb{J}$ in the context S and $S \vdash a : A$, we need to define the coercion:

$$M_1, M_2 : S \vdash coe_{B\ a} : S^{\cong} \to B_1\ a_1 \to B_2\ a_2$$

where a_i and B_i are $a[M_i]$ and $B[M_i]$ as usual. Let $S = (T, B : A \to \mathbb{J}, S')$. In the context of $M_1, M_2 : S$, given any $i : S^{\cong}$, by the definition of $(T, B : A \to \mathbb{J})^{\cong}$ in 1.3, we can project out from $i : S^{\cong}$ to an element

$$p_i : \Sigma(j : T^{\cong}).\ (a_1 : A_1) \to (B_1\ a_1 \cong B_2\ (coe_A\ j\ a_1))$$

Now given any $b_1 : B_1\ a_1$, using $(\pi_2\ p_i\ a_1).fwd$, we get an element of type $B_2\ (coe_A\ (\pi_1\ p_i)\ a_1)$. Use the coherence $coh_a\ i : coe_A\ (\pi_1\ p_i)\ a_1 = a_2$, we get an element of $B_2\ a_2$ as needed.

Case 2.2. The case of unit judgement is simple. There is a unique choice of

$$M_1, M_2 : S \vdash coe_1 : S^{\cong}[M_1, M_2] \to 1 \to 1$$

which is functorial because 1 has a unique element.

Case 2.3. For the case $S \vdash \Sigma\ A\ B : \mathbb{J}$ for some $S \vdash A : \mathbb{J}$ and $S.A \vdash B : \mathbb{J}$, we need to define a term that coerce elements of Σ-types along isomorphisms:

$$M_1, M_2 : S \vdash coe_{\Sigma AB} : S^{\cong}[M_1, M_2] \to \Sigma\ A_1\ B_1 \to \Sigma\ A_2\ B_2$$

where A_i and B_i stand for $A[M_i]$ and $(M_1, M_2 : S, a_i : A_i \vdash B[M_i, a_i])$ respectively. By the inductive hypotheses, we can use terms

$$M_1, M_2 : S \vdash coe_A : S^{\cong}[M_1, M_2] \to A_1 \to A_2$$
$$M_1', M_2' : S.A \vdash coe_B : (S.A)^{\cong}[M_1', M_2'] \to B[M_1'] \to B[M_2']$$

We can use coe_A to coerce the first component of $\Sigma\ A\ B$:

$$coe_{\Sigma AB}\ i\ (a_1, b_1) = (coe_A\ i\ a_1,\ ?0 : B[M_2, coe_A\ i\ a_1]\ )$$

Now to use coe_B to fill out the hole $?0$, we recall that the judgement $(S.A)^{\cong}$ of $(S.A)$-isomorphisms is defined earlier in 1.2 to be

$$M_1', M_2' : S.A \vdash \Sigma(i : S^{\cong}[M_1'.S, M_2'.S]).\ \big(coe_A\ i\ (\pi_2\ M_1') = \pi_2\ M_2'\big) : \mathbb{J}$$

Thus we fill out the hole ?0 by putting

$$?0 = coe_B[(M_1, a_1), (M_2, coe_A\ i\ a_1)]\ (i, refl)\ b_1$$

and the resulting coercion term is

$$coe_{\Sigma AB}\ i\ (a_1, b_1) = (coe_A\ i\ a_1,\ coe_B[(M_1, a_1), (M_2, coe_A\ i\ a_1)]\ (i, refl)\ b_1).$$

whose functoriality follows from that of coe_A and coe_B.

Case 2.4. The case of $\Pi\ A\ B$ is similar to the one of $\Sigma\ A\ B$ above, except that we need to use the backward direction of an isomorphism to coerce A_2 to A_1 in order to coerce a function $f : (a_1 : A_1) \to B_1$ to a function $(a_2 : A_2) \to B_2$:

$$coe_{\Pi AB}\ i\ f = \lambda(a_2 : A_2).\ coe_B[\sigma]\ \ (i, refl)\ \ (f\ (coe_A\ (inv_S\ i)\ a_2))$$

where the substitution σ is $[(M_1, coe_A\ (inv_S\ i)\ a_2),\ (M_2, a_2)]$. This definition is well typed because coe_A is functorial, so

$$coe_A\ i\ (coe_A\ (inv_S\ i)\ a_2) = a_2$$

and thus $(i, refl)$ is indeed an element of $(S.A)^{\cong}[\sigma]$.

Case 2.5. For $S \vdash a = b : \mathbb{J}$, where $S \vdash A : \mathbb{J}$ and $S \vdash a, b : A$, we need

$$M_1, M_2 : S \vdash coe_{a=b} : S^{\cong}[M_1, M_2] \to (a_1 = b_1) \to (a_2 = b_2)$$

where a_i and b_i stand for $a[M_i]$ and $b[M_i]$ as usual. Given $i : S^{\cong}[M_1, M_2]$ and $a_1 = b_1$, we have $coe_A\ i\ a_1 = coe_A\ i\ b_1$, and the inductive hypotheses give us

$$M_1, M_2 : S \vdash coh_a\ i : (coe_A\ i\ a_1) = a_2$$
$$M_1, M_2 : S \vdash coh_b\ i : (coe_A\ i\ b_1) = b_2$$

Hence we have $a_2 = b_2$ as required.

Part 3. Finally, we need to verify that every term $S \vdash a : A$ for any signature S and judgement $S \vdash A : \mathbb{J}$ satisfies the coherence condition:

$$M_1, M_2 : S \vdash coh_a : (i : S^{\cong}[M_1, M_2]) \to coe_A\ i\ a[M_1] = a[M_2]$$

We omit the details here because it is relatively routine verification: for the case where a is a variable in the S, the coherence is guaranteed by the definition of $(T, a : A)^{\cong}$ earlier in 1.2; the case for other term formers follow from the inductive hypotheses of subterms and the $\beta\eta$-equalities of the connectives. □

4.4∗7 Remark. The constructions $\langle S^{\cong}, coe, coh\rangle$ in the preceding proof are essentially the same idea as *observational equality* [Altenkirch et al. 2007, 2019], but at one homotopy level higher. Altenkirch et al. [2007] constructed a universe of sets (with extensional principles such as function extensionality and uniqueness of identity proofs) in intensional type theory with proof-irrelevant propositions,

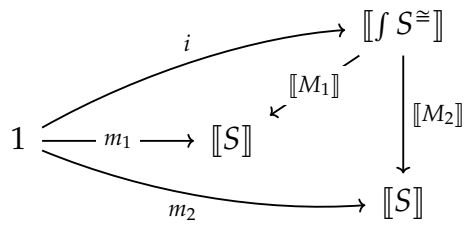
while here we constructed groupoids from sets. But the idea of defining equalities or isomorphisms *structurally for each type former* remains the same.

4.4∗8. Given an signature S, again, we will reuse S as the name of the iterative Σ-type of the fields of S, and denote the signature $(M_1 : S, M_2 : S, i : S^{\cong})$ by $\int S^{\cong}$. In the category LFSIG of LF-contexts, there are two projections morphisms

$$M_1, M_2 : \int S^{\cong} \longrightarrow S$$

which are interpreted by $[\![-]\!]$ (4.3.2∗1) in the presheaf category $\mathrm{PR}\,\mathscr{C}$ for every LCCC $\mathscr{C}$ as two morphisms $[\![M_1]\!], [\![M_2]\!] : [\![\int S^{\cong}]\!] \to [\![S]\!]$.

4.4∗9 Definition. Given a signature S and an LCCC $\mathscr{C}$, an *isomorphism* of diagrammatic models $m_1, m_2 : 1 \to [\![S]\!]$ in $\mathscr{C}$ is a morphism $i : 1 \to [\![\int S^{\cong}]\!]$ in $\mathrm{PR}\,\mathscr{C}$ such that $m_1 = [\![M_1]\!] \cdot i$ and $m_2 = [\![M_2]\!] \cdot i$:



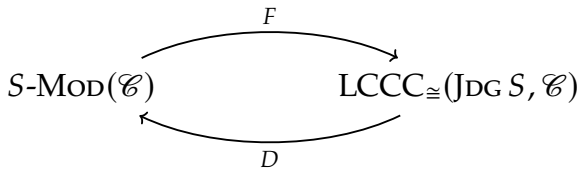
The groupoid of diagrammatic models and isomorphisms, induced by the internal groupoid structure on $S^{\cong}$, is denoted as S-MOD($\mathscr{C}$).

4.4∗10 Theorem. *For every LCCC $\mathscr{C}$ and signature S, the mappings F and D between diagrammatic and functorial models from* 4.3.2∗6 *and* 4.3.3∗2 *extend to an equivalence*:

$$F : S\text{-MOD}(\mathscr{C}) \cong \mathrm{LCCC}_{\cong}(\mathrm{JDG}\, S, \mathscr{C}) : D$$

where $\mathrm{LCCC}_{\cong}(\mathrm{JDG}\, S, \mathscr{C})$ *is the groupoid of* (1) *functors* $\mathrm{JDG}\, S \to \mathscr{C}$ *that preserve locally cartesian closed structures and* (2) *natural isomorphisms.*

Proof sketch. So far F and D are merely functions between diagrammatic and functorial models, so we first need to extend them to functors



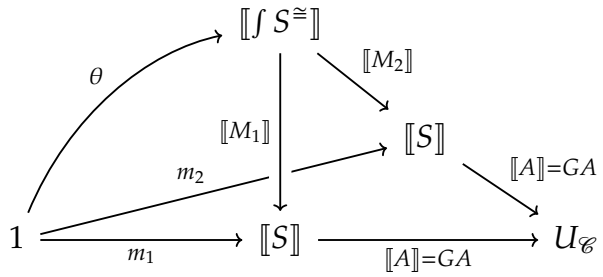
and then show that they form a pair of equivalence.

Part 1. Let us start with extending F to a functor. Let $m_1, m_2 : 1 \to [\![S]\!]$ be two diagrammatic models and $\theta : 1 \to [\![\int S^{\cong}]\!]$ be an isomorphism between them. By

Definition 4.3.2∗6, $F(m_i) : \textsc{Jdg}\, S \to \mathscr{C}$ is defined to be the composite

$$\textsc{Jdg}\, S \xrightarrow{G} [U_{\mathscr{C}}]_{[\![S]\!]} \xrightarrow{m_i^*} [U_{\mathscr{C}}]_1 \xrightarrow{\cong} \mathscr{C},$$

where G maps every $S \vdash A : \mathbb{J}$ to its interpretation $[\![A]\!] : [\![S]\!] \to U_{\mathscr{C}}$, viewed as an object of the fiber category $[U_{\mathscr{C}}]_{[\![S]\!]}$. To define $F(\theta)$, it is sufficient to define a natural isomorphism $m_1^* G \cong m_2^* G$; by the definition of diagrammatic models, $m_i = [\![M_i]\!] \cdot \theta$, so it is furthermore sufficient to define a natural isomorphism $[\![M_1]\!]^* G \cong [\![M_2]\!]^* G : \textsc{Jdg}\, S \to [U_{\mathscr{C}}]_{[\![\int S^{\cong}]\!]}$. The situation is as follows:



To define the required natural transformation at every judgement $S \vdash A : \mathbb{J}$, we use the coerce morphism of the judgement A from Lemma 4.4∗6:

$$M_1, M_2 : S \vdash coe_A : S^{\cong}[M_1, M_2] \to A[M_1] \to A[M_2],$$

or more precisely, its uncurry morphism:

$$M_1, M_2 : S, i : S^{\cong}[M_1, M_2] \vdash \overline{coe_A} : A[M_1] \to A[M_2].$$

The interpretation of this morphism in $\textsc{Pr}\,\mathscr{C}$ is a morphism

$$[\![\overline{coe_A}]\!] : [\![M_1]\!]^*[\![A]\!] \to [\![M_2]\!]^*[\![A]\!]$$

in the fiber category $[U_{\mathscr{C}}]_{[\![\int S^{\cong}]\!]}$ that we are looking for. The invertibility of this map follows from the invertibility inv_S of $S^{\cong}$ in Lemma 4.4∗6.

Naturality of the family of morphisms defined above comes from the coherence terms in Lemma 4.4∗6: given a morphism $f : A \to B$ in $\textsc{Jdg}\, S$, it induces a term $S \vdash \lambda f : A \to B$ in the LF. Now using the coherence $coh_{\lambda f}$ from Lemma 4.4∗6, we have a commutative diagram in the category $\textsc{Jdg}(\int S^{\cong})$:

$$\begin{array}{ccc} A[M_1] & \xrightarrow{\overline{coe_A}} & A[M_2] \\ {\scriptstyle f[M_1]}\downarrow & & \downarrow{\scriptstyle f[M_2]} \\ B[M_1] & \xrightarrow[\overline{coe_B}]{} & B[M_2] \end{array}$$

whose interpretation in $[U_{\mathscr{C}}]_{[\![\int S^{\cong}]\!]}$ is exactly the needed naturality square.

Part 2. Now to extend D to a functor $\textsc{Lccc}_{\cong}(\textsc{Jdg}\, S, \mathscr{C}) \to S\text{-}\textsc{Mod}(\mathscr{C})$, given two LCC-functors $N_1, N_2 : \textsc{Jdg}\, S \to \mathscr{C}$ and a natural isomorphism $\sigma : N_1 \cong N_2$,

we would like to define $D(\sigma) : 1 \to [\![\int S^{\cong}]\!]$ in $\mathrm{Pr}\,\mathscr{C}$ that lies over $D(N_1)$ and $D(N_2)$. Following the structure of the proofs of Lemma 4.3.3*2 and Lemma 4.4*6, we define $D(\sigma)$ by induction on the structure of standard signatures S, and maintain the additional invariant that the coercion $\int S^{\cong} \vdash \overline{coe_A} : A[M_1] \to A[M_2]$ for every judgement A is mapped by the functor $F(D(\sigma)) : \mathrm{Jdg}(\int S^{\cong}) \to \mathscr{C}$ to exactly the component of the natural isomorphism $\sigma_A : N_1 A \to N_2 A$ at A, modulo the isomorphisms $N_i A \cong F(D(N_i))$ in Lemma 4.3.3*2. There is no need to impose any invariant for the coherence terms coh_a in this proof, since in LCCCs two morphisms can be equal in at most one way. This is a rather tedious inductive proof so we only provide a sketch here.

Case 2.1. The case for the empty signature is trivial, as $F(\sigma)$ is unique.

Case 2.2. For $S = (T, a : A)$ where $T \vdash A : \mathbb{J}$, according to Case 1.2 of the proof of Lemma 4.4*6, we need a global element of the interpretation of the type

$$\Sigma(i : T^{\cong}[M_1.T, M_2.T]).\ \big(coe_A[M_1.T, M_2.T]\ i\ M_1.a = M_2.a\big).$$

The first component is obtained by induction for the signature T, and the second component is obtained from the invariant that coercion morphisms are interpreted as the components of the natural isomorphism.

Case 2.3. For $S = (T, B : A \to \mathbb{J})$ where $T \vdash A : \mathbb{J}$, by Case 1.3 of the proof of Lemma 4.4*6, we need a global element of the interpretation of the type

$$\Sigma(i : T^{\cong}[M_1.T, M_2.T]).\ \big((a_1 : A_1) \to (B_1\ a_1 \cong B_2\ (coe_A\ i\ a_1))\big). \tag{4.10}$$

The first component is still obtained by induction, and the second component is obtained from the component of $\sigma : N_1 \to N_2$ at the judgement $\Sigma\ A\ B$.

We also need to check that all judgements in a signature S maintain our invariant that the coercion map coe_A is interpreted as the component σ_A. This is again shown by induction on the structure of (standard) judgements, mimicking the structure of the definition of coe_A.

Case 2.a. The case for $S \vdash B\ a : \mathbb{J}$ for some variable $(B : A \to \mathbb{J}) \in S$ follows from the fact that coercion for this case is defined in Case 2.1 of the proof of Lemma 4.4*6 to be invoking the forward direction of the second component of (4.10), which we have defined to be a component of the natural isomorphism σ.

Case 2.b. The case for the unit type is trivial. The cases for Σ and Γ types uses the fact that N_1 and N_2 preserve LCCC structures and σ is natural, so its components at Σ-types and Π-types behave the same as how the coercion maps of Σ-types and Π-types are defined in the proof of Lemma 4.4*6.

Case 2.c. The case for equality types is also trivial, because the interpretation of extensional equality types in LCCCs have at most one element.

Part 3. Next we show that F and D form a pair of equivalence of groupoids. First recall that in Lemma 4.3.3*2, we have already a family of isomorphisms

$M \cong F(D(M))$ for all $M : \textsc{Jdg}\, S \to \mathscr{C}$. If we examine the proof of Lemma 4.3.3*2, we can notice that every component of the isomorphism $M \cong F(D(M))$ is defined using the unique morphism into some construct with a universal property. Thus the family of isomorphisms $M \cong F(D(M))$ is necessarily natural in M.

It remains to construct a family of isomorphisms $m \cong D(F(m))$, natural in $m \in S\text{-}\textsc{Mod}(\mathscr{C})$, for all signatures S. Again, this is constructed by induction on the structure of standard signatures S, with the additional invariant that the natural isomorphism $F(D(F(m))) \cong F(m)$ constructed in Lemma 4.3.3*2 coincides with the interpretation of coercion morphisms coe_A in $\mathscr{C}$ for every judgement $S \vdash A : \mathbb{J}$. We will only provide a sketch here.

Case 3.1. For the empty signature, $m \cong D(F(m))$ trivially holds because there is a unique diagrammatic model and there is a unique isomorphism.

Case 3.2. For the signature $S = (T, a : A)$ where $T \vdash A : \mathbb{J}$, by Case 1.2 of Lemma 4.4*6, $(M_1, M_2 : S \vdash \int S^{\cong} : \mathbb{J})$ is the judgement

$$\Sigma(i : T^{\cong}[M_1.T, M_2.T]).\ \big(coe_A[M_1.T, M_2.T]\ i\ M_1.a = M_2.a\big).$$

We need to construct a global element of $[\![\int S^{\cong}]\!]$ that lies over m and $D(F(m))$. The first component $(i : T^{\cong}[M_1.T, M_2.T])$ can be obtained by the inductive hypothesis for the signature T. The second component $coe_A[M_1.T, M_2.T]\ i\ M_1.a = M_2.a$ follows from the additional invariant that the interpretation of coe_A coincides with the isomorphism $F(D(F(m)))(A) \to F(m)(A)$ in Lemma 4.3.3*2, which was used to define the $(a : A)$ component of $D(F(m))$ in Case 1.2 of Lemma 4.3.3*2.

Case 3.3. For the signature $S = (T, B : A \to \mathbb{J})$ where $T \vdash A : \mathbb{J}$, by Case 1.3 of Lemma 4.4*6, $(M_1, M_2 : S \vdash \int S^{\cong} : \mathbb{J})$ is the judgement

$$\Sigma(i : T^{\cong}[M_1.T, M_2.T]).\ \big((a_1 : A_1) \to (B_1\ a_1 \cong B_2\ (coe_A\ i\ a_1))\big)$$

The first component is still obtained by induction. To construct the second component, we recall that Case 1.3 of the proof of Lemma 4.3.3*2 defines the component $B : A \to \mathbb{J}$ of the diagrammatic model $D(F(m))$ to be a code of

$$F(m)(\pi_1) : F(m)(\Sigma\ A\ B) \to F(m)(A) \tag{4.11}$$

in the universe $p : \tilde{U}_{\mathscr{C}} \to U_{\mathscr{C}}$. By the definition of $F(m)$, the B component of m is the classifying map of a morphism isomorphic to (4.11) in $\textsc{Pr}\,\mathscr{C}/F(m)(A)$, so we have the required $(a_1 : A_1) \to B_1\ a_1 \cong B_2\ (coe_A\ i\ a_1)$. □

4.5 Discussion

4.5*1. In this chapter we have developed of a logical framework LccLF and its categorical semantics in locally cartesian closed categories. Our development follows the tradition of categorical logic pioneered by Lawvere [1963]: a syntactic

presentation S of a theory generates a *classifying category* C with certain structures, and structure-preserving functors $C \to D$ are equivalent to models of S in D.

4.5∗2. The development in this chapter is presented in the traditional set-theoretic language, but we did not rely on anything specific to material set theories or the axiom of choice, as long as concepts such as LCCCs are all defined as categories with chosen structures rather than just mere existence of structures. Therefore the development in this chapter is valid internally in any elementary topos with an NNO and universes. This in particular includes the effective topos Eff [Hyland 1982; Oosten 2008], which will be useful when we study type theories with impredicative polymorphisms and effects in future chapters.

4.5∗3. Unlike in untyped or simply typed settings [Crole 1994; Jacobs 1999], the definition of diagrammatic models of LccLF is significantly harder than that of functorial models. But our effort of building the bridge between these two sides will pay off in the upcoming chapters: functorial semantics unlocks the opportunity for using *abstract* categorical tools, while diagrammatic semantics allows us to use *concrete* type-theoretic internal languages of categories to define models of type theories. Such a synthesis of the abstract and the concrete greatly simplifies our task of defining type theories for higher-order computational effects and proving their meta-theoretic properties in the upcoming chapters.

4.5∗4. The development of LccLF in this chapter is by no means the end of the story. The following are some possible directions of future work.

1. Signatures of LccLF in this chapters are finitary in two aspects: firstly, signatures of LccLF can only contain finitely many declarations; secondly, every operation has finitely many operands. It is worthwhile to relax both restrictions, so that, for example, type theories with countably many universes or infinitary products can be accommodated.

2. We have shown models of every signature S are equivalent to LCC-functors out of $\textsc{Jdg}\, S$, which is called a *semantic* categorical algebraic theory correspondence by Fiore and Mahmoud [2014], but we did not show the *syntactic* correspondence: the category of LF signatures is equivalent to the category of LCCCs – of course we need to first define morphisms of LF signatures before showing such an equivalence holds.

Chapter 5

A Polymorphic Language with Higher-Order Effects

5∗1. Using the logical framework LccLF that we have developed in Chapter 4, in this chapter we present an extension to *higher-order polymorphic λ-calculus*, also known as System F_ω [Girard 1972, 1986], with *fine-grain call-by-value* computations with (monadic and equation-less) higher-order algebraic effects and handlers. We refer to the extended type theory as *System* F^{ha}_ω.

System $\mathrm{F}^{\mathrm{ha}}_\omega$ is a quite simple extension of F_ω, but expressive type-and-effect systems can be recovered by further extending the kind sub-language of $\mathrm{F}^{\mathrm{ha}}_\omega$ with standard connectives. Therefore $\mathrm{F}^{\mathrm{ha}}_\omega$ is an ideal core type theory for programming languages with impredicative polymorphism and higher-order effects.

5∗2. The plan of this chapter is as follows. In Section 5.1, we define the language $\mathrm{F}^{\mathrm{ha}}_\omega$ in three steps. First we define System F_ω as a signature in LccLF. Then we define some standard concepts related to computational effects such as (raw) functors and monads in F_ω. Finally we add fine-grain call-by-value computation judgements, parameterised by a (raw) higher-order endofunctor H that is the signature of the effectful operations. Each computation judgement is equipped with H-operations and an evaluation/handling construct that interprets a computation by a (raw) monad equipped with H-operations.

5∗3. In Section 5.2, we give a denotational semantics to $\mathrm{F}^{\mathrm{ha}}_\omega$ by extending the standard realizability model of F_ω to $\mathrm{F}^{\mathrm{ha}}_\omega$, which establishes the consistency of $\mathrm{F}^{\mathrm{ha}}_\omega$ in the sense that the two Boolean terms tt and ff are not judgementally equal. The main technical difficulty is to reconcile a mismatch between computation judgements and user-defined (raw) monads: the former satisfy monadic laws but the latter do not. Moreover, the realizability model essentially interprets a program t of $\mathrm{F}^{\mathrm{ha}}_\omega$ as an element $[\![t]\!]$ of an untyped computation model $\mathbb{A}$, e.g. untyped λ-calculus or Turing machines. Therefore the realizability model implicitly provides a way to run programs of $\mathrm{F}^{\mathrm{ha}}_\omega$ without an operational semantics.

5∗4. We then study the meta-theoretic properties of $\mathrm{F}^{\mathrm{ha}}_\omega$. First in Section 5.3, we give a brief tutorial to Sterling's [2021] *synthetic Tait computability* (STC), a

type/category-theoretic reincarnation of Tait's [1967] *computability method* (also known as *logical relations* or the *reducibility method*). We first explain how defining logical relations and establishing the so-called 'fundamental lemma' can be viewed as constructing a model of the programming language in a certain (presheaf) topos from Artin gluing. Then a type theory TTstc is introduced for conveniently manipulating objects in this topos, and defining a logical relation for a programming language and proving the fundamental lemma is reduced to defining a model of the language in TTstc. This can be summarised as the slogan 'logical relations as types' [Sterling and Harper 2021].

In Section 5.4, we use STC to prove the (closed term) *canonicity* of $\mathrm{F}_\omega^{\mathrm{ha}}$, which states that every (closed) Boolean term t in $\mathrm{F}_\omega^{\mathrm{ha}}$ is judgementally equal to one of the canonical Boolean values: tt and ff. The most interesting part of this proof is the logical predicate for computations, which uses a form of $\top\top$*-lifting* [Katsumata 2005; Lindley and Stark 2005] to circumvent the mismatch between (lawful) computations and (lawless) user-defined raw monads mentioned in 5∗3.

In Section 5.5, we note an additional benefit of using an axiomatised language (namely, TTstc) to do our logical relation proof: by re-interpreting our (unary) logical predicate model for canonicity in a slightly different glued topos, we obtain *binary parametricity* for closed $\mathrm{F}_\omega^{\mathrm{ha}}$-terms for free.

5∗5. Finally, in Section 5.6 we extend $\mathrm{F}_\omega^{\mathrm{ha}}$ with general recursion and give it a denotational model using ideas from *synthetic domain theory*.

5.1 The Signature of System $\mathrm{F}_\omega^{\mathrm{ha}}$

5.1∗1. We present the signature of $\mathrm{F}_\omega^{\mathrm{ha}}$ in two steps: in this section we first define Girard's [1972; 1986] System F_ω in LccLF, and in the next section we bring in computations. The complete signature for $\mathrm{F}_\omega^{\mathrm{ha}}$ is collected in Appendix A.1.

5.1.1 The Signature of System F_ω

5.1.1∗1 (Kinds). Kinds of F_ω are structurally the same as types of STLC from Example 4.1∗6, so F_ω has the following declarations for *kinds*:

$$ki : \mathbb{J} \qquad el : ki \to \mathbb{J} \qquad ty : ki \qquad \_\Rightarrow_k\_ : ki \to ki \to ki \tag{5.1}$$

where we have a judgement ki for kinds, a family of judgements el for elements of kinds, a base kind $ty : ki$ classifying *types*, and function kinds $k_1 \Rightarrow_k k_2$.

Elements of the base kind $ty : ki$ includes a unit type *unit*, a (weak) Boolean type *bool*, function types $A \Rightarrow_t B$, and impredicative polymorphic function types

All k A where *k* can be of any kind:

$$\begin{array}{ccc} \mathit{unit} : \mathit{el}\ \mathit{ty} & \mathit{bool} : \mathit{el}\ \mathit{ty} & \_\Rightarrow_t\_ : \mathit{el}\ \mathit{ty} \to \mathit{el}\ \mathit{ty} \to \mathit{el}\ \mathit{ty} \\ \multicolumn{3}{c}{\mathit{All} : (k : \mathit{ki}) \to (\mathit{el}\ k \to \mathit{el}\ \mathit{ty}) \to \mathit{el}\ \mathit{ty}} \end{array}$$

Elements of function kinds are specified via an isomorphism to LF-functions as we did in Example 4.1∗6 for STLC function types:

$$\Rightarrow_k\text{-}\mathit{iso} : \{A, B : \mathit{ki}\} \to \mathit{el}\ (A \Rightarrow_k B) \cong (\mathit{el}\ A \to \mathit{el}\ B) \tag{5.2}$$

5.1.1∗2 Notation. For brevity, in the future we will sometimes elide the use of the forward and backward direction of an isomorphism that specifies a judgement, as if the isomorphism is a strict equality. For example, given $C : \mathit{el}\ (A \Rightarrow_k B)$ and $A : \mathit{el}\ \mathit{ty}$, we may write $C\ A : \mathit{el}\ \mathit{ty}$ to mean $\Rightarrow_k\text{-}\mathit{iso}.\mathit{fwd}\ C\ A$.

5.1.1∗3 (Types). For terms of types, there is a judgement $\mathit{tm} : \mathit{el}\ \mathit{ty} \to \mathbb{J}$. Terms of (polymorphic) function types and the unit type are still specified by HOAS:

$$\begin{array}{rl} \mathit{tm} : \mathit{el}\ \mathit{ty} \to \mathbb{J} & \mathit{unit}\text{-}\mathit{iso} : \mathit{tm}\ \mathit{unit} \cong 1 \\ \Rightarrow_t\text{-}\mathit{iso} : \{A, B : \mathit{el}\ \mathit{ty}\} \to \mathit{tm}\ (A \Rightarrow_t B) \cong (\mathit{tm}\ A \to \mathit{tm}\ B) & \\ \mathit{All}\text{-}\mathit{iso} : \{k : \_\}\ \{A : \_\} \to \mathit{tm}\ (\mathit{All}\ k\ A) \cong ((\alpha : \mathit{el}\ k) \to \mathit{tm}\ (A\ \alpha)) & \end{array} \tag{5.3}$$

For the weak Boolean type, we only include two terms *tt* and *ff*:

$$\mathit{tt} : \mathit{tm}\ \mathit{bool} \qquad\qquad \mathit{ff} : \mathit{tm}\ \mathit{bool}$$

This completes the signature of System F_ω. We have set it up in a minimal way for simplicity when proving meta-theoretic properties of $\mathrm{F}^{\mathrm{ha}}_\omega$ later, but more useful types/kinds, such as a strong Boolean type with the correct universal property, products and lists, can be added easily and can be found in Appendix A.2.

5.1.1∗4 Example. The Church encoding of lists in F_ω can be defined as follows:

$$\begin{array}{l} \mathit{CList} : \mathit{el}\ (\mathit{ty} \Rightarrow_k \mathit{ty}) \\ \mathit{CList} = \lambda(A : \mathit{el}\ \mathit{ty}).\ \mathit{All}\ \mathit{ty}\ (\lambda(\alpha : \mathit{el}\ \mathit{ty}).\ \alpha \Rightarrow_t (A \Rightarrow_t \alpha \Rightarrow_t \alpha) \Rightarrow_t \alpha)) \end{array}$$

where we have omitted the application of $\Rightarrow_k\text{-}\mathit{iso}.\mathit{bwd}$ following 5.1.1∗2. The term for empty lists, polymorphic in the element type, is then

$$\begin{array}{l} \mathit{CNil} : \mathit{tm}\ (\mathit{All}\ \mathit{ty}\ (\lambda(A : \mathit{el}\ \mathit{ty}).\ \mathit{CList}\ A)) \\ \mathit{CNil} = \lambda(A : \mathit{el}\ \mathit{ty}).\ \lambda(\alpha : \mathit{el}\ \mathit{ty}).\ \lambda(\mathit{nil} : \mathit{tm}\ \alpha).\ \lambda\mathit{cons}.\ \mathit{nil} \end{array}$$

5.1.2 Derived Concepts in System F_ω

5.1.2∗1. System F_ω^{ha} roughly adds to F_ω an initial monad M equipped with a natural transformation $H\ M \to M$ for every higher-order endofunctor

$$H : (ty \Rightarrow_k ty) \Rightarrow_k (ty \Rightarrow_k ty)$$

over F_ω-functors $ty \Rightarrow_k ty$. In the following, let us define these derived concepts in F_ω, such as functors, monads, natural transformations, which will be ingredients for the judgements for computations in Section 5.1.3.

5.1.2∗2. A (raw) F_ω-functor is a type constructor $F_0 : ty \Rightarrow_k ty$ equipped with a term F_1 acting on F_ω-functions, similar to the situation in Haskell:

$$\begin{array}{l}
tyco : ki \\
tyco = (ty \Rightarrow_k ty) \\
fmap\text{-}ty : (F : el\ tyco) \to el\ ty \\
fmap\text{-}ty\ F = All\ ty\ (\lambda\alpha.\ All\ ty\ (\lambda\beta.\ (\alpha \Rightarrow_t \beta) \Rightarrow_t (F\ \alpha \Rightarrow_t F\ \beta))) \\
\textsc{record}\ RawFunctor : \mathbb{J}\ \textsc{where} \\
\quad F_0 : el\ tyco \\
\quad F_1 : tm\ (fmap\text{-}ty\ F_0)
\end{array}$$

The judgement *RawFunctor* is called *raw* because F_0 and F_1 are not required to satisfy the laws of functors.

5.1.2∗3. There are two other ways to remedy the absence of equality types:

1. We may include an axiom or axiom schema of *relational parametricity* [Plotkin and Abadi 1993; Reynolds 1983] or some weaker form of it, such as dinaturality, in our language as judgemental equalities. Then all raw functors would satisfy the functor laws judgementally following parametricity.

2. Although F_ω does not have equality types, we do have equality types in LccLF; for example, we have the following judgement in LccLF:

   $$\begin{array}{l}
   \textsc{record}\ Functor : \mathbb{J}\ \textsc{where} \\
   \quad \textsc{include}\ RawFunctor \\
   \quad _ : \{A : el\ ty\} \to F_1\ A\ A\ id = id \\
   \quad _ : \{A, B, C, f, g\} \to F_1\ B\ C\ g \circ F_1\ A\ B\ f = F_1\ A\ C\ (g \circ f)
   \end{array}$$

   Using this judgement of functors in F_ω^{ha} amounts to requiring functors to satisfy the its laws judgementally.

However, approach (1) is not strong enough to eliminate all needs of equality types in F_ω^{ha}: even with parametricity, raw monads do not necessarily satisfy the monad laws, let alone arbitrary equational theories on effects.

On the other hand, approach (2) is overly strong because checking whether a raw functor satisfies the functor laws judgementally is highly nontrivial (if decidable at all). This will be a problem later when we have rules in $\mathrm{F}_\omega^{\mathrm{ha}}$ that take in elements of *Functor* as input, which would mean that a type checker of *preterms* of $\mathrm{F}_\omega^{\mathrm{ha}}$ will need to decide if a raw functor is lawful judgementally (unless we make the programmer to supply the proof on the preterms, but this would undermine our intention of $\mathrm{F}_\omega^{\mathrm{ha}}$ as a core type theory for a practical programming language with higher-order algebraic effects).

For these reasons, we choose to work with lawless functors, monads, and so on in $\mathrm{F}_\omega^{\mathrm{ha}}$, which is a faithful model of current functional programming languages such as Haskell and OCaml anyway.

5.1.2∗4. We also need (raw) monads *RawMonad* later in $\mathrm{F}_\omega^{\mathrm{ha}}$, whose definition is the same as its counterpart in Haskell:

$$
\begin{array}{l}
\textsc{record}\ \mathit{RawMonad} : \mathbb{J}\ \textsc{where} \\
\quad M_0 \ : \mathit{el}\ \mathit{tyco} \\
\quad \mathit{ret} \ : \mathit{tm}\ (\mathit{All}\ \mathit{ty}\ (\lambda\alpha.\ \alpha \Rightarrow_t M_0\ \alpha)) \\
\quad \mathit{bind} : \mathit{tm}\ (\mathit{All}\ \mathit{ty}\ (\lambda\alpha.\ \mathit{All}\ \mathit{ty}\ (\lambda\beta.\ M_0\ \alpha \Rightarrow_t (\alpha \Rightarrow_t M_0\ \beta) \Rightarrow_t M_0\ \beta)))
\end{array}
$$

5.1.2∗5. Finally, we will need (raw) higher-order functors *RawHFunctor*, which consists of (1) a higher-order type constructors H_0 mapping type constructors to type constructors, (2) a function *hfmap* mapping an *fmap* for a type constructor F to an *fmap* for $H_0\ F$, and (3) a function *hmap* mapping a transformation $F \to G$ to another $H\ F \to H\ G$:

$$
\begin{array}{l}
\mathit{htyco} : \mathit{ki} \\
\mathit{htyco} = \mathit{tyco} \Rightarrow_k \mathit{tyco} \\
\mathit{trans} : (F, G : \mathit{el}\ \mathit{tyco}) \to \mathit{el}\ \mathit{ty} \\
\mathit{trans}\ F\ G = \mathit{All}\ \mathit{ty}\ (\lambda\alpha.\ F\ \alpha \Rightarrow_t G\ \alpha) \\
\textsc{record}\ \mathit{RawHFunctor} : \mathbb{J}\ \textsc{where} \\
\quad H_0 \ : \mathit{el}\ \mathit{htyco} \\
\quad \mathit{hfmap} : (F : \mathit{RawFunctor}) \to \mathit{tm}\ (\mathit{fmap\text{-}ty}\ (H_0\ (F\ .F_0))) \\
\quad \mathit{hmap} \ : (F, G : \mathit{RawFunctor}) \to \mathit{tm}\ (\mathit{trans}\ (F\ .F_0)\ (G\ .F_0)) \\
\qquad\quad \to \mathit{tm}\ (\mathit{trans}\ (H_0\ (F\ .F_0))\ (H_0\ (G\ .F_0)))
\end{array}
$$

5.1.2∗6 Notation. We will typically suppress the field accessors F_0, M_0 and H_0 for readability, so given $F : \mathit{RawFunctor}$ and $X : \mathit{el}\ \mathit{ty}$, we write $F\ X$ for $F.F_0\ X$.

5.1.3 Computation Types for Higher-Order Algebraic Effects

5.1.3∗1. Now we introduce computations to F_ω to obtain $\mathrm{F}_\omega^{\mathrm{ha}}$. We follow the *fine-grain call-by-value* (FGCBV) approach [Lassen 1998; Levy et al. 2003]. For all

$H : \mathit{RawHFunctor}$ and $A : \mathit{el}\ \mathit{ty}$, there are judgements $\mathit{co}\ H\ A$ for *computations* of A-values with effectful operations specified by H:

$$\mathit{co} : (H : \mathit{RawHFunctor}) \to (A : \mathit{el}\ \mathit{ty}) \to \mathbb{J}$$

The judgement has the following two rules for pure computations and sequential composition of computations respectively:

$$\begin{aligned}
&\mathit{val} \quad : \{H, A\} \to \mathit{tm}\ A \to \mathit{co}\ H\ A \\
&\mathit{let\text{-}in} : \{H, A, B\} \to \mathit{co}\ H\ A \to (\mathit{tm}\ A \to \mathit{co}\ H\ B) \to \mathit{co}\ H\ B
\end{aligned}$$

The interaction of *val* and *let-in* is axiomatised by the following declarations, which are essentially the monad laws:

$$\begin{aligned}
&\mathit{val\text{-}let} \quad : \{H, A, B\} \to (a : \mathit{tm}\ A) \to (k : \mathit{tm}\ A \to \mathit{co}\ H\ B) \\
&\qquad \to \mathit{let\text{-}in}\ (\mathit{val}\ a)\ k = k\ a \\
&\mathit{let\text{-}val} \quad : \{H, A\} \to (m : \mathit{co}\ H\ A) \to \mathit{let\text{-}in}\ m\ \mathit{val} = m \\
&\mathit{let\text{-}assoc} : \{H, A, B, C\} \to (m_1 : \mathit{co}\ H\ A) \\
&\qquad \to (m_2 : \mathit{tm}\ A \to \mathit{co}\ H\ B) \to (m_3 : \mathit{tm}\ B \to \mathit{co}\ H\ C) \\
&\qquad \to \mathit{let\text{-}in}\ (\mathit{let\text{-}in}\ m_1\ m_2)\ m_3 = \mathit{let\text{-}in}\ m_1\ (\lambda a.\ \mathit{let\text{-}in}\ (m_2\ a)\ m_3)
\end{aligned} \tag{5.4}$$

5.1.3∗2. We also introduce a new type former $\mathit{th}\ H\ A$ for *thunks* of computations of A-values with effects of H, whose terms are isomorphic to computations:

$$\begin{aligned}
\mathit{th} &: \mathit{RawHFunctor} \to \mathit{el}\ \mathit{ty} \to \mathit{el}\ \mathit{ty} \\
\mathit{th\text{-}iso} &: \{H, A\} \to \mathit{tm}\ (\mathit{th}\ H\ A) \cong \mathit{co}\ H\ A
\end{aligned}$$

The two directions of the isomorphism *th-iso* will be called $\Uparrow$ and $\Downarrow$ respectively:

$$\Uparrow : \mathit{tm}\ (\mathit{th}\ H\ A) \to \mathit{co}\ H\ A \qquad\qquad \Downarrow : \mathit{co}\ H\ A \to \mathit{tm}\ (\mathit{th}\ H\ A)$$

Thunks can be packed into a monad:

$$\begin{aligned}
&\mathit{th\text{-}mnd} : \mathit{RawHFunctor} \to \mathit{RawMonad} \\
&\mathit{th\text{-}mnd}\ H\ .M_0 \;= \mathit{th}\ H \\
&\mathit{th\text{-}mnd}\ H\ .\mathit{ret} \;= \lambda A\ x.\ \Downarrow (\mathit{val}\ x) \\
&\mathit{th\text{-}mnd}\ H\ .\mathit{bind} = \lambda A\ B\ m\ k.\ \Downarrow (\mathit{let\text{-}in}\ (\Uparrow m)\ (\lambda a.\ \Uparrow (k\ a)))
\end{aligned}$$

Following from equations (5.4), *th-mnd* satisfies the monad laws too.

Levy et al. [2003] presented the FGCBV calculus using *effectful functions*:

$$\begin{aligned}
_ \Rightarrow [_] _ &: \mathit{el}\ \mathit{ty} \to \mathit{RawHFunctor} \to \mathit{el}\ \mathit{ty} \to \mathit{el}\ \mathit{ty} \\
\mathit{ef\text{-}iso} &: \{A, H, B\} \to \mathit{tm}\ (A \Rightarrow [H]\ B) \cong (\mathit{tm}\ A \to \mathit{co}\ H\ B)
\end{aligned}$$

But since we already have pure functions in the language, it is sufficient to have the thunk type, and define effectful functions as $(A \Rightarrow [H]\ B) := (A \Rightarrow_t \mathit{th}\ H\ B)$.

5.1.3∗3 (Operations). Effectful operations that computations can perform are introduced by the following declaration:

$$op : \{H, A, B\} \to tm\ (H\ (th\ H)\ A) \to (tm\ A \to co\ H\ B) \to co\ H\ B$$

The first argument $o : tm\ (H\ (th\ H)\ A)$ of op is the input to an operation call, such as some parameters or computations that the operation call acts on. The second argument $k : tm\ A \to co\ H\ B$ of op is the 'continuation' of the computation after this operation call, where the argument $tm\ A$ of k is the result of the operation call. The result $op\ o\ k$ is understood as the computation that first makes an operation call o, which returns an A-value, and then continues as k.

The interaction of operation calls and sequential composition of computations is the following, which is sometimes called *algebraicity* [Plotkin and Power 2001b]:

$$\begin{aligned} &let\text{-}op : \{H, A, B, C\} \to (p : tm\ (H\ (th\ H)\ A)) \\ &\quad \to (k : tm\ A \to co\ H\ B) \to (k' : tm\ B \to co\ H\ C) \\ &\quad \to let\text{-}in\ (op\ p\ k)\ k' = op\ p\ (\lambda a.\ let\text{-}in\ (k\ a)\ k') \end{aligned} \tag{5.5}$$

5.1.3∗4 (Evaluation). Now we axiomatise that computations $co\ H\ A$ can be *evaluated*, or *handled*, by any monads supporting the operations from H. We define the following structure for monads supporting operations from H:

$$\begin{aligned} &\textsc{record}\ MonadAlg\ (H : RawHFunctor) : \mathbb{J},\ \textsc{where} \\ &\quad \textsc{include}\ RawMonad\ \textsc{as}\ M \\ &\quad malg : tm\ (trans\ (H\ M_0)\ M_0) \end{aligned}$$

where by '$\textsc{include}\ RawMonad\ \textsc{as}\ M$', we mean that $MonadAlg$ has all the fields of the record $RawMonad$ from 5.1.2∗4 – namely, M_0, ret, and $bind$. Moreover, for every $m : MonadAlg\ H$, there is a projection $m.M : RawMonad$.

An example of such a monad is $th\text{-}mnd$ from 5.1.3∗2:

$$\begin{aligned} &th\text{-}alg : (H : RawHFunctor) \to MonadAlg\ H \\ &th\text{-}alg\ H\ .M \quad\ = th\text{-}mnd\ H \\ &th\text{-}alg\ H\ .malg = \lambda\alpha\ o.\ \Downarrow (op\ o\ val) \end{aligned}$$

We then add to $\mathrm{F}^{\mathrm{ha}}_{\omega}$ the following declaration that evaluates a computation with effect H with any monad that supports the effect:

$$eval : \{H\} \to (m : MonadAlg\ H) \to (A : el\ ty) \to co\ H\ A \to tm\ (m\ A)$$

5.1.3∗5. The last piece of the signature of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ is the computation rules for $eval$, which are similar to the small-step operational semantics of the handlers in conventional algebraic effects [Plotkin and Pretnar 2009, 2013]:

1. When the computation is a value, it is handled by the ret of the monad,

$$\begin{aligned} &eval\text{-}val : \{H, A\} \to (m : MonadAlg\ H) \to (a : tm\ A) \\ &\quad \to eval\ m\ A\ (val\ a) = m\ .ret\ A\ a \end{aligned} \tag{5.6}$$

2. When the computation is an operation call, it is handled by the corresponding operation on the monad, with all subterms recursively evaluated:

$$\begin{aligned}
&\mathit{eval\text{-}op} : \{H, A, B\} \to (m : \mathit{MonadAlg}\ H) \\
&\qquad \to (p : \mathit{tm}\ (H\ (\mathit{th}\ H)\ A)) \to (k : \mathit{tm}\ A \to \mathit{co}\ H\ B) \\
&\qquad \to \textsc{let}\ \mathit{bind} = m\ .\mathit{bind}\ A\ B \\
&\qquad\qquad\ \ \mathit{malg} = m\ .\mathit{malg}\ A \\
&\qquad\qquad\ \ T \quad = \mathit{fct\text{-}of\text{-}mnd}\ (\mathit{th\text{-}mnd}\ H) \\
&\qquad\qquad\ \ M \quad = \mathit{fct\text{-}of\text{-}mnd}\ (m\ .M) \\
&\qquad\ \ \textsc{in}\ \mathit{eval}\ m\ B\ (\mathit{op}\ p\ k) \\
&\qquad\qquad = \mathit{bind}\ (\mathit{malg}\ (H\ .\mathit{hmap}\ T\ M\ (\lambda\alpha\ c.\ \mathit{eval}\ m\ \alpha\ (\Uparrow c))\ A\ p)) \\
&\qquad\qquad\qquad\ (\lambda a.\ \mathit{eval}\ m\ B\ (k\ a))
\end{aligned} \tag{5.7}$$

where *fct-of-mnd* is the canonical functor structure of a monad:

$$\begin{aligned}
&\mathit{fct\text{-}of\text{-}mnd} : \mathit{RawMonad} \to \mathit{RawFunctor} \\
&\mathit{fct\text{-}of\text{-}mnd}\ m\ .F_0 = m\ .M_0 \\
&\mathit{fct\text{-}of\text{-}mnd}\ m\ .F_1\ \alpha\ \beta\ f\ ma = m\ .\mathit{bind}\ \alpha\ \beta\ ma\ (\lambda a.\ m\ .\mathit{ret}\ _\ (f\ a))
\end{aligned}$$

This completes the signature of $\mathrm{F}_\omega^{\mathrm{ha}}$. For easy reference, the full signature of $\mathrm{F}_\omega^{\mathrm{ha}}$ is collected in Appendix A.1.

5.1.4 Some Remarks on the Choice of the Rules

5.1.4∗1. In view of the undecidability of the equational theory of System T with a natural number type *with judgemental η-rule* [Okada and Scott 2000], we do not include in $\mathrm{F}_\omega^{\mathrm{ha}}$ the η-rule for *eval* asserting that all $f : \mathit{co}\ H\ A \to \mathit{tm}\ (m\ .M_0\ A)$ satisfying the properties similar to *eval-val* and *eval-op* are judgementally equal to *eval*. Otherwise we would not have normalisation for (open terms of) $\mathrm{F}_\omega^{\mathrm{ha}}$.

5.1.4∗2. We do not include in $\mathrm{F}_\omega^{\mathrm{ha}}$ the following equation asserting that *eval* also commutes with *let-in*:

$$\begin{aligned}
&\mathit{eval\text{-}let} : \{H, A, B\} \to (m : \mathit{MonadAlg}\ H) \\
&\quad \to (c : \mathit{co}\ H\ A) \to (f : \mathit{tm}\ A \to \mathit{co}\ H\ B) \\
&\quad \to \mathit{eval}\ m\ B\ (\mathit{let\text{-}in}\ c\ f) = m\ .\mathit{bind}\ A\ B\ (\mathit{eval}\ m\ A\ c)\ (\lambda a.\ \mathit{eval}\ m\ B\ (f\ a))
\end{aligned}$$

This is because we have chosen to work with *raw* monads that may not validate the monad laws, whereas computations *co H A* are axiomatised to always satisfy these laws (5.4). Consequently, we can freely re-associate let-bindings in computations but not in raw monads, so having *eval-let* would result in inconsistency.

Although *eval-let* is left out, later we will prove the *canonicity* of $\mathrm{F}_\omega^{\mathrm{ha}}$ – evaluating *closed* elements of computations never get stuck. This is intuitively because in the empty context, every computation is always equal to a computation without *let-in* because of the equations *let-val*, *let-assoc* and *let-op*.

5.1.4∗3. We did not declare in System $\mathrm{F}^{\mathrm{ha}}_{\omega}$ any judgements for *modular handlers* or *effect systems* [Bauer and Pretnar 2014; Kammar and Plotkin 2012; Lucassen and Gifford 1988] that track the effect operations that a computation may perform, because both of them can be *derived concepts* in $\mathrm{F}^{\mathrm{ha}}_{\omega}$.

Firstly, the judgement for *effect families* is the following record in $\mathrm{F}^{\mathrm{ha}}_{\omega}$:

$$\begin{aligned}&\textsc{record}\ \mathit{Fam} : \mathbb{J}\ \textsc{where}\\ &\quad \mathit{eff}\ \ : \mathit{ki}\\ &\quad \mathit{sig}\ \ : \mathit{el}\ \mathit{eff} \to \mathit{RawHFunctor}\\ &\quad \mathit{add} : \mathit{el}\ \mathit{eff} \to \mathit{el}\ \mathit{eff} \to \mathit{el}\ \mathit{eff}\end{aligned}$$

The elements of the kind $\mathit{eff} : \mathit{ki}$ are effect signatures in this family, each of them determining a higher-order functor via *sig*. Additionally, there is a way *add* to combine two effects in a family.

Then we have the following definitions for monads and computations for an effect e in a family F, which can be viewed as a generic effect system parameterised by an effect family F:

$$\begin{aligned}&\mathit{MonadEff} : (F : \mathit{Fam}) \to (e : \mathit{el}\ (F\ .\mathit{eff})) \to \mathbb{J}\\ &\mathit{MonadEff}\ F\ e = \mathit{MonadAlg}\ (F\ .\mathit{sig}\ e)\\ &\mathit{co}[\_\ni\_] : (F : \mathit{Fam}) \to (e : \mathit{el}\ (F\ .\mathit{eff})) \to \mathit{el}\ \mathit{ty} \to \mathbb{J}\\ &\mathit{co}[F \ni e] = \mathit{co}\ (F\ .\mathit{sig}\ e)\end{aligned}$$

A modular handler processing the effect e in a family F and outputting the effect o is the following structure:

$$\begin{aligned}&\textsc{record}\ \mathit{Hdl}\ (F : \mathit{Fam})\ (e\ o : \mathit{el}\ (F\ .\mathit{eff})) : \mathbb{J}\ \textsc{where}\\ &\quad \mathit{alg}\ \ : (\mu : \mathit{el}\ (F\ .\mathit{eff})) \to \mathit{MonadEff}\ F\ (F\ .\mathit{add}\ o\ \mu)\\ &\qquad \to \mathit{MonadEff}\ F\ (F\ .\mathit{add}\ e\ \mu)\\ &\quad \mathit{res}\ \ : \mathit{el}\ (\mathit{ty} \Rightarrow_k \mathit{ty})\\ &\quad \mathit{run} : (\mu : \mathit{el}\ (F\ .\mathit{eff})) \to (\mathit{Mo} : \mathit{MonadEff}\ F\ (F\ .\mathit{add}\ o\ \mu))\\ &\qquad \to \mathit{tm}\ (\mathit{trans}\ (\mathit{alg}\ \mu\ \mathit{Mo})\ (\lambda A.\ \mathit{Mo}\ (\mathit{res}\ A)))\end{aligned}$$

Modular handlers as such can be applied to computations $\mathit{co}[F \ni (F\ .\mathit{add}\ e\ \mu)]\ A$, for all 'ambient' effects μ, removing the effect e and generating the effect o, yielding computations $\mathit{co}[F \ni (F\ .\mathit{add}\ o\ \mu)]\ (h\ .\mathit{res}\ A)$:

$$\begin{aligned}&\mathit{handle} : \{F, e, o, \mu, A\} \to (h : \mathit{Hdl}\ F\ e\ o) \to \mathit{co}[F \ni (F\ .\mathit{add}\ e\ \mu)]\ A\\ &\qquad \to \mathit{co}[F \ni (F\ .\mathit{add}\ o\ \mu)]\ (h\ .\mathit{res}\ A)\\ &\mathit{handle}\ h\ c = \Uparrow (h\ .\mathit{run}\ \mu\ T\ A\ c')\ \textsc{where}\\ &\quad T : \mathit{MonadEff}\ F\ (F\ .\mathit{add}\ o\ \mu)\\ &\quad T = \mathit{th\text{-}alg}\ (F\ .\mathit{sig}\ (F\ .\mathit{add}\ o\ \mu))\\ &\quad c' : \mathit{tm}\ (h\ .\mathit{alg}\ \mu\ T\ A)\\ &\quad c' = \mathit{eval}\ (h\ .\mathit{alg}\ \mu\ (\mathit{th\text{-}alg}\ (F\ .\mathit{sig}\ (F\ .\mathit{add}\ o\ \mu))))\ \_\ c\end{aligned}$$

5.1.4∗4 Example. An effect system *algFam* : *Fam* of algebraic operations that is comparable to row-polymorphism-based effect systems of algebraic effects [Leijen 2014; Lindley and Cheney 2012] can be defined provided that we extend F_ω with the following standard connectives:

- ∗ type-level and kind-level products ($\times_t$ and $\times_k$),
- ∗ the empty type and type-level coproducts (+),
- ∗ kind-level lists $\mathit{list}_k :: \mathit{ki} \to \mathit{ki}$ with elimination to *modules* in the sense of ML-family languages – a *signature* is a kind $k : \mathit{ki}$ with a type $t : \mathit{el}\ k \to \mathit{el}\ \mathit{ty}$ that may depend on an element of k; a *module* of the signature (k, t) is a pair of an element $\alpha : \mathit{el}\ k$ and a term $e : \mathit{tm}\ (t\ \alpha)$ [Harper 2016].

We define $\mathit{algFam}\ .\mathit{eff} = \mathit{list}_k\ (\mathit{ty}\ \times_k\ \mathit{ty})$, i.e. an effect is a list of pairs of types. Every element (P, A) of such a list is understood as an algebraic operation of parameter type P and arity type A, and induces a (constant) higher-order functor:

$$\lambda F.\ \lambda X.\ P\ \times_t\ (A \Rightarrow_t X)$$

The field *sig* of *algFam* then takes the coproduct of the higher-order functor above for each element (P, A) in its argument of kind $\mathit{list}_k\ (\mathit{ty}\ \times_k\ \mathit{ty})$ – this is why we need elimination of kind-level lists to modules. The *add* field of *algFam* combines two effects by list appending.

The complete definition of this example, together with the needed extra type connectives, can be found in Appendix A.2.

5.1.4∗5 Example. Similar to the last example, we can define a family *scpFam* : *Fam* of *scoped operations* with $\mathit{scpFam}\ .\mathit{eff}$ still being $\mathit{list}_k\ (\mathit{ty}\ \times_k\ \mathit{ty})$, but each element P, A of the list is to be interpreted as the higher-order functor for a scoped operation with a parameter of type P and A-many scopes:

$$\lambda F.\ \lambda X.\ P\ \times_t\ (A \Rightarrow_t F\ X) \tag{5.8}$$

The component *sig* of *scpFam* also takes the coproduct of the higher-order functor of the element of its argument, and *add* is still list appending.

5.2 Realizability Model of $\mathrm{F}^{\mathrm{ha}}_{\omega}$

5.2∗1. In this section, we establish the *consistency* of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ by a realizability model, which is the standard way to model type theories with impredicative polymorphism [Asperti and Martini 1992; Bainbridge et al. 1990; Crole 1994; Jacobs 1999]. A useful by-product of this model is that every program is interpreted as a function that is realised by some Turing machine (or an element of some other *partial combinatory algebras* such as untyped λ-calculus), so if we carry out the

interpretation in a constructive meta-theory, the realizability model also gives us a compiler for $\mathrm{F}^{\mathrm{ha}}_{\omega}$: every closed program $t : \mathit{bool}$ of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ is compiled to a halting Turing machine that outputs either true or false. In the next section, we will also see that this compiler is adequate with respect to the equational theory of $\mathrm{F}^{\mathrm{ha}}_{\omega}$: the Turing machine for $t : \mathit{bool}$ outputs true (or false) if and only if $t = tt$ (or $t = \mathit{ff}$) judgementally in the equational theory of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ (Corollary 5.4.7∗4).

5.2∗2. We will construct a model of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ in the locally cartesian closed category Asm of *assemblies* over an arbitrary partial combinatory algebra $\mathbb{A}$ [Oosten 2008]. We will also use a type-theoretic internal language of Asm to describe constructions in Asm. The language we use and its interpretation in Asm are close to Luo's [1994] *extended calculus of constructions* and its interpretation in ω-sets [Luo 1994, Chapter 7], except that we only need two, rather than countably many, predicative universes V_i, and we use extensional rather than intensional equality types, since we are not concerned with mechanical type checking here.

1. The category Asm has a universe $\pi_P : \tilde{P} \to P$ of *modest sets*, also known as *partial equivalence relations* (PERs), which is closed under finite product and coproduct types, dependent pairs, extensional equality types, inductive types. Moreover, the universe P is *impredicative* in the sense that it is closed under dependent products $\Pi\ A\ B$ for arbitrary *assemblies* A and families of modest sets B, or in more standard phrasing, the internal category determined by the universe P is *complete* [Hyland 1988].

2. Any Grothendieck universe U of sets in the ambient set theory gives rise to a universe $\pi_V : \tilde{V} \to V$ of *U-small assemblies* in Asm: the underlying set $|V|$ of the assembly V contains all assemblies A such that $|A| \in U$, and the existence predicate of the assembly V is the codiscrete one:

$$r \Vdash_V A \quad \text{for all} \quad r \in \mathbb{A}, A \in |V|.$$

The assembly $\tilde{V}$ has *pointed U-small assemblies* as elements:

$$|\tilde{V}| = \{(A, a) \mid A \in |V|, a \in |A|\};$$

with realizers $r \Vdash_{\tilde{V}} (A, a)$ iff $r \Vdash_A a$. The morphism $\pi_V : \tilde{V} \to V$ is the evident projection. The universe V is also closed under finite (co)product types, dependent functions, dependent pairs, extensional equality types, inductive types, and moreover, the universe P of modest sets. However, V is not impredicative (otherwise it would be inconsistent as Coquand [1986] shows that one impredicative universe cannot contain another).

In this section we assume two Grothendieck universes $U_1 \in U_2$ in the ambient set theory, so in the internal language of Asm, we have a hierarchy of three cumulative universes $P : V_1 : V_2$ with P impredicative and V_i predicative.

5.2∗3. According to Section 4.3, a model of an LF-signature S in an LCCC $\mathscr{C}$ is a global element $1 \to [\![S]\!]$ of the interpretation of S in $\mathrm{Pr}\,\mathscr{C}$ with $\mathbb{J}$ interpreted by the universe $U_{\mathscr{C}}$ that classifies $\mathscr{C}$-morphism, or in the internal language of $\mathrm{Pr}\,\mathscr{C}$, a closed element $m : [\![S]\!]$ of the record type containing all the declaration S with $\mathbb{J}$ replaced by $U_{\mathscr{C}}$. More specially, if the category $\mathscr{C}$ already has a universe U that supports the type connectives of $\mathbb{J}$, a *U-small* model of S in $\mathscr{C}$ can be given by a closed element $m : [\![S]\!]_U$ of the record type in the language of $\mathscr{C}$ that contains the declarations of S with $\mathbb{J}$ replaced by U.

5.2∗4. In the following, we construct a V_2-small model $M : [\![\mathrm{F}^{\mathrm{ha}}_{\omega}]\!]_{V_2}$ in the internal language of Asm. Kinds are interpreted as the predicative universe V_1:

$$\begin{array}{ll} M.ki : V_2 & M.el : M.ki \to V_2 \\ M.ki = V_1 & M.el\ k = k \end{array}$$

Function kinds $M.\_\Rightarrow_k\_ : M.ki \to M.ki \to M.ki$ are interpreted by function types in V_1, and $M.\Rightarrow_k$*-iso* is the identity isomorphism.

5.2∗5. The base kind $ty : ki$ is interpreted as the impredicative universe P:

$$\begin{array}{ll} M.ty : M.ki & M.tm : M.el\ M.ty \to V_2 \\ M.ty = P & M.tm\ A = A \end{array}$$

The unit, Boolean, function types of *ty* in $\mathrm{F}^{\mathrm{ha}}_{\omega}$ are interpreted as the corresponding type formers in the universe P. The impredicative polymorphic function type *All* is interpreted as dependent function type:

$$\begin{array}{l} M.All : (k : ki) \to (M.el\ k \to M.el\ M.ty) \to M.ty \\ M.All\ k\ A = \Pi\ k\ A \end{array}$$

This is well typed because $M.ty$, i.e. P, is an impredicative universe.

5.2∗6. The meaning of derived concepts, such as *M.RawFunctor*, from Section 5.1.2 is fixed by other declarations, so we do not need to model them.

5.2∗7. The model of the computation judgement *co H A* is less obvious because of the mismatch between computations and raw monads in $\mathrm{F}^{\mathrm{ha}}_{\omega}$: computations satisfy the monadic laws strictly (*let-val*, *val-let*, *let-assoc* from 5.1.3∗2), while raw monads do not. Consequently, we cannot model *co H* as the initial *raw monad* equipped with H-operations because it then would not satisfy the monadic laws. Conversely, we cannot model it as the initial *monad* equipped with H-operations either because then it cannot be evaluated into raw monads.

5.2∗8 Remark. Following the formula $\mu X.\ I + A \,\square\, X + \Sigma X$ of free Σ-monoid over A in 2.4∗6, the author's first attempt to resolve the dilemma was to model

computations as the *initial pointed raw functor with H-operations*, which can be equipped with a *law-abiding* monad structure, and moreover may be evaluated into every raw monad M with H-operations, since M is a pointed functor as well. Unfortunately, it turns out that this approach validates the equations on computations only when $H.hmap$ preserves composition of transformations for every raw higher-order functor H, but it is not the case in our model.

One possible way to resolve this is to shift our model category from Asm to a category where impredicative polymorphism is always *parametric*, such as the category of reflexive graphs by Atkey et al. [2014], then every $H : M.RawHFunctor$ will always be lawful higher-order functors.

5.2∗9. There is a more lightweight solution that allows us to stay in Asm: we model computations by a combination of impredicative encoding and continuation-passing transformation:

$$\begin{aligned}&M.co : M.RawHFunctor \to M.el\ M.ty \to P\\&M.co\ H\ A = (T : M.MonadAlg\ H) \to (B : P) \to (A \to T\ B) \to T\ B\end{aligned}$$

Thunking $th\ H\ A$ can be modelled as the identity, because in the model M, computations and values live in the same universe P:

$$\begin{aligned}&M.th : M.RawHFunctor \to M.el\ M.ty \to M.el\ M.ty\\&M.th\ H\ A = M.co\ H\ A\end{aligned}$$

5.2∗10. The computation formers and *eval* are defined as follows:

$$\begin{aligned}&M.val : \{H, A\} \to A \to M.co\ H\ A\\&M.val\ a = \lambda T\ B\ (r : A \to T\ B).\ r\ a\end{aligned}$$

$$\begin{aligned}&M.let\text{-}in : \{H, A, B\} \to M.co\ H\ A \to (A \to M.co\ H\ B) \to M.co\ H\ B\\&M.let\text{-}in\ \{A, B\}\ c\ k = \lambda T\ C\ (r : B \to T\ C).\ c\ T\ C\ (\lambda a.\ k\ a\ T\ C\ r)\end{aligned}$$

We need to check that they satisfy the monadic laws of computations (5.4):

∗ For *val-let*, given any $a : M.tm\ A$ and $k : M.tm\ A \to M.co\ H\ B$,

$$\begin{aligned}&let\text{-}in\ (val\ a)\ k\\=\quad&\{\text{by definition of } M.let\text{-}in\}\\&\lambda T\ C\ r.\ val\ a\ T\ C\ (\lambda a.\ k\ a\ T\ C\ r)\\=\quad&\{\text{by definition of } M.val\}\\&\lambda T\ C\ r.\ k\ a\ T\ C\ r\\=\quad&\{\eta\text{-rule for functions}\}\\&k\ a\end{aligned}$$

* The case for *let-val* is very similar. Given any $c : M.co\ H\ A$,

$$
\begin{aligned}
& \mathit{let\text{-}in}\ c\ \mathit{val} \\
= \quad & \{\text{by definition of } M.\mathit{let\text{-}in}\} \\
& \lambda T\ C\ r.\ c\ T\ C\ (\lambda a.\ \mathit{val}\ a\ T\ C\ r) \\
= \quad & \{\text{by definition of } M.\mathit{val}\} \\
& \lambda T\ C\ r.\ c\ T\ C\ (\lambda a.\ r\ a) \\
= \quad & \{\eta\text{-rule for functions}\} \\
& c
\end{aligned}
$$

* For *let-assoc*, given any c_1, c_2, and c_3, we have

$$
\begin{aligned}
& \mathit{let\text{-}in}\ (\mathit{let\text{-}in}\ c_1\ c_2)\ c_3 \\
= {} & \lambda T\ C\ r.\ (\mathit{let\text{-}in}\ c_1\ c_2)\ T\ C\ (\lambda b.\ c_3\ b\ T\ C\ r) \\
= {} & \lambda T\ C\ r.\ c_1\ T\ C\ (\lambda a.\ c_2\ a\ T\ C\ (\lambda b.\ c_3\ b\ T\ C\ r)) \\
= {} & \lambda T\ C\ r.\ c_1\ T\ C\ (\lambda a.\ \mathit{let\text{-}in}\ (c_2\ a)\ c_3) \\
= {} & \mathit{let\text{-}in}\ c_1\ (\lambda a.\ \mathit{let\text{-}in}\ (c_2\ a)\ c_3)
\end{aligned}
$$

5.2∗11. The model of *evaluation* directly follows from the definition of $M.co$:

$$
\begin{aligned}
& M.\mathit{eval} : \{H\} \to (T : M.\mathit{MonadAlg}\ H) \to (A : P) \to M.co\ H\ A \to T\ A \\
& M.\mathit{eval}\ \{H\}\ T\ A\ c = c\ T\ A\ (T.\mathit{ret})
\end{aligned}
$$

We check that the equation *eval-val* (5.6) is satisfied: for all $H : M.\mathit{RawHFunctor}$, $A : M.el\ M.ty$, $T : M.\mathit{MonadAlg}\ H$ and $a : A$,

$$
\begin{aligned}
& M.\mathit{eval}\ T\ A\ (M.\mathit{val}\ a) \\
= \quad & \{\text{by definition of } M.\mathit{eval}\} \\
& M.\mathit{val}\ a\ T\ A\ T.\mathit{ret} \\
= \quad & \{\text{by definition of } M.\mathit{val}\} \\
& (\lambda T\ B\ r.\ r\ a)\ T\ A\ T.\mathit{ret} \\
= {} & T.\mathit{ret}\ a
\end{aligned}
$$

5.2∗12. The model of operations is defined as follows:

$$
\begin{aligned}
& M.\mathit{op} : \{H, A, B\} \to H\ (\mathit{th}\ H)\ A \to (A \to M.co\ H\ B) \to M.co\ H\ B \\
& M.\mathit{op}\ o\ k = \lambda T\ C\ r. \\
& \quad T.\mathit{bind}\ A\ C \\
& \qquad (T.\mathit{malg}\ A\ (H.\mathit{hmap}\ (\mathit{th}\ H)\ T\ (M.\mathit{eval}\ T)\ A\ o)) \\
& \qquad (\lambda a.\ k\ a\ T\ C\ r)
\end{aligned}
$$

It remains to check that the equations *let-op* (5.5) and *eval-op* (5.7) are satisfied.

For *let-op*, given arbitrary $o : H\ (th\ H)\ A$, $k : A \to co\ H\ B$, $k' : B \to co\ H\ C$,

$$
\begin{aligned}
& \mathit{let\text{-}in}\ (\mathit{op}\ o\ k)\ k' \\
= \quad & \{\text{by definition of } M.\mathit{let\text{-}in}\} \\
& \lambda T\ C\ r.\ (\mathit{op}\ o\ k)\ T\ C\ (\lambda b.\ k'\ b\ T\ C\ r) \\
= \quad & \left\{ \begin{array}{l} \text{by definition of } M.\mathit{op} \text{ and let } o' \text{ be} \\ \quad T.\mathit{malg}\ \_\ (H.\mathit{hmap}\ \_\ \_\ (M.\mathit{eval}\ T)\ \_\ o) \end{array} \right\} \\
& T.\mathit{bind}\ \_\ \_\ o'\ (\lambda a.\ k\ a\ T\ \_\ (\lambda b.\ k'\ b\ T\ \_\ r)) \\
= \quad & \{\text{by definition of } M.\mathit{let\text{-}in}\ (k\ a)\ k'\} \\
& \mathit{op}\ o\ (\lambda a.\ \mathit{let\text{-}in}\ (k\ a)\ k')
\end{aligned}
\tag{5.9}
$$

For *eval-op*, given any $T : MonadAlg\ H$, $o : H\ (th\ H)\ A$ and $k : A \to co\ H\ B$,

$$
\begin{aligned}
& \mathit{eval}\ T\ (\mathit{op}\ o\ k) \\
= \quad & \{\text{by definition of } M.\mathit{eval}\} \\
& (\mathit{op}\ o\ k)\ T\ \_\ T.\mathit{ret} \\
= \quad & \{\text{by definition of } M.\mathit{op} \text{ and let } o' \text{ be the same as in (5.9)}\} \\
& T.\mathit{bind}\ \_\ \_\ o'\ (\lambda a.\ k\ a\ T\ \_\ T.\mathit{ret}) \\
= \quad & \{\text{by definition of } M.\mathit{eval}\ T\ \_\ k\ a\} \\
& T.\mathit{bind}\ \_\ \_\ o'\ (\lambda a.\ \mathit{eval}\ T\ \_\ (k\ a))
\end{aligned}
$$

This completes our definition of the model $M : [\![\mathrm{F}^{\mathrm{ha}}_{\omega}]\!]_{V_2}$ in Asm. An immediate consequence is the *consistency* of System $\mathrm{F}^{\mathrm{ha}}_{\omega}$.

5.2∗13 Theorem. *The equational theory of System* $\mathrm{F}^{\mathrm{ha}}_{\omega}$ *is consistent, in the sense that the closed terms tt and ff : bool are not judgementally equal.*

Proof. Models of the logical framework respect judgemental equalities strictly, and the interpretation of *tt* and *ff* in the realizability model M are different, so they cannot be judgementally equal. □

5.2∗14. Another consequence of the realizability model is that it provides a way to *compute* terms of $\mathrm{F}^{\mathrm{ha}}_{\omega}$: terms and computations of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ are interpreted as morphisms in the category Asm of assemblies, which are realized by elements of the underlying partial combinatory algebra (PCA) $\mathbb{A}$. If we choose $\mathbb{A}$ to be the PCA of Turning machines or the PCA of untyped λ-calculus, we can compute an $\mathrm{F}^{\mathrm{ha}}_{\omega}$ term by computing the realizer of the interpretation. (This also requires the process of interpreting $\mathrm{F}^{\mathrm{ha}}_{\omega}$ into Asm to be constructive, which is indeed the case since the theory of the logical framework LccLF is constructively valid.)

For example, let $\mathrm{F}^{\mathrm{ha}}_{\omega} \vdash p : tm\ bool$ be any closed $\mathrm{F}^{\mathrm{ha}}_{\omega}$-term of the Boolean type, its interpretation in Asm is a morphism $[\![p]\!] : 1 \to 1 + 1$, which is realized by

some total Turing machine. By executing this machine, we get a Boolean answer.

5.2∗15. Note that above 'compute' merely means getting a Boolean answer from an $\mathrm{F}_\omega^{\mathrm{ha}}$-program $p : \mathit{bool}$, and a priori we have not ruled out the possibility that the computation may be inadequate. For example, it might be possible that for all p, the result is constantly true. In the next section, we will see that the realizability model is in fact *adequate* with respect to the equational theory of $\mathrm{F}_\omega^{\mathrm{ha}}$: if the interpretation $[\![p]\!]$ of a closed Boolean term $p : \mathit{tm}\ \mathit{bool}$ in the realizability model is true (resp. false), then $p = \mathit{tt}$ (resp. $p = \mathit{ff}$) in the equational theory of $\mathrm{F}_\omega^{\mathrm{ha}}$. Adequacy of a denotational model is usually proved by a logical relation model relating the syntax and the semantics [Plotkin 1977]. However, a consequence of not having general recursion in $\mathrm{F}_\omega^{\mathrm{ha}}$ is that the adequacy of the realizability model directly follows from (1) the fact that the denotation of tt and ff in the realizability model is adequate and (2) the *canonicity* of $\mathrm{F}_\omega^{\mathrm{ha}}$: for every closed $\mathrm{F}_\omega^{\mathrm{ha}}$-term $p : \mathit{tm}\ \mathit{bool}$, either $p = \mathit{tt}$ or $p = \mathit{ff}$.

5.3 Logical Relations, Categorically and Synthetically

5.3∗1. Proving canonicity of $\mathrm{F}_\omega^{\mathrm{ha}}$ is the subject of the rest of this chapter. Although the statement of canonicity only concerns about the base type *bool*, terms of *bool* may be obtained from applications of functions or evaluations of computations. Therefore it is necessary to prove something stronger for all types and kinds.

5.3∗2. The common proof strategy for such meta-theoretic properties of type theories is *logical relations* [Plotkin 1973, 1980], also known as *the computability method* or the *reducibility method* in the literature [Girard 1972; Martin-Löf 1975a,b; Statman 1985; Tait 1967].

We will use a type-theoretic approach to logical relations called *synthetic Tait computability* (STC) due to Sterling [2021] to do our proof. In this section, we first briefly overview the technique of logical relations in Section 5.3.1 and then introduce the language of STC in Section 5.3.2 and 5.3.3.

5.3.1 A Brief Overview of Logical Relations

5.3.1∗1. Given a model $M : \textsc{Jdg}\, S \to \mathscr{C}$ of a theory S, suppose that we would like to show that the interpretation Mt of every closed deduction $t : 1 \to X \in \textsc{Jdg}\, S$ of some judgement X satisfies certain property. The proof strategy of *unary logical relations (i.e. logical predicates) on the model M* proceeds as follows:

1. For every judgement $A \in \textsc{Jdg}\, S$, a predicate $P_A \subseteq \mathscr{C}(1, MA)$ on global elements of MA is defined by induction on the structure of the judgement

A, so that P_1 on the unit judgement is constantly true and that the predicate P_X implies the desired property.

2. Then the interpretation Mt of every morphism $t : A \to B \in \textsc{Jdg}\, S$ is shown to preserve the predicates, $a \in P_A$ implies $(Mt \cdot a) \in P_B$, by induction on the structure of t. This is usually called 'the fundamental lemma'.

3. Consequently, the interpretation Mt of every closed term $t : 1 \to X$ satisfies P_X (and thus the desired property), since $* : 1 \to 1$ always satisfies P_1.

The special case of $M = \mathrm{Id} : \textsc{Jdg}\, S \to \textsc{Jdg}\, S$ is called *syntactic logical relations*.

5.3.1∗2. Generalisations of this proof strategy abound in the literature:

- The unary logical predicates P_A can be obviously generalised to be n-ary relations, so that they can be used for proving relational properties such as *parametricity* [Reynolds 1983]. Moreover they can relate different models, useful for proving properties such as *adequacy* of a denotational model respect to operational semantics [Plotkin 1977].

- The predicates P_A may be generalised to be a family of predicates P^n_A indexed by natural numbers n to handle recursive types and general store, known as *step-indexed logical relations* [Ahmed 2006].

- The predicates P_A may be generalised to be *proof-relevant*, so that P_A becomes a family of sets indexed by deductions of A [Altenkirch et al. 1995; Coquand 2023; Fiore 2022]. Such proof-relevant logical predicates handle *universes* in dependent type theories smoothly and allow an algorithm to be extracted from the logical-relation proof if the proof is done constructively.

- The predicates P_A may be generalised to be a family of predicates $P_{\Gamma \vdash A}$ on *open* deductions $\mathscr{C}(M\Gamma, MA)$, indexed by a category or poset of contexts Γ, known as *Kripke logical relations* [Jung and Tiuryn 1993]. This is essential for proving meta-theoretic properties concerning about open terms rather than just closed terms, for example, normalisation of open terms and definability of a morphism in the semantic model by an open term.

5.3.1∗3. Logical relations on a model $M : \textsc{Jdg}\, S \to \mathscr{C}$, and all the generalisations mentioned above, can be elegantly and fruitfully understood as constructing a model $M^* : \textsc{Jdg}\, S \to \mathscr{E} \downarrow F$ of the object theory S in the comma category $\mathscr{E} \downarrow F$ for some functor $F : \mathscr{C} \to \mathscr{E}$ from the category $\mathscr{C}$ to another category $\mathscr{E}$, such that there is a commutative triangle for π the projection functor $\mathscr{E} \downarrow F \to \mathscr{C}$

[Altenkirch et al. 1995; Fiore 2022; Freyd 1978; Sterling and Spitters 2018]:

$$\begin{array}{ccc} \textsc{Jdg}\, S & \xrightarrow{M^*} & \mathscr{E} \downarrow F \\ & \searrow^{M} & \downarrow \pi \\ & & \mathscr{C} \end{array}$$

5.3.1∗4 Definition. Given a functor $F : \mathscr{C} \to \mathscr{E}$, the comma category $\mathscr{C} \downarrow \mathscr{E}$ has as objects $\langle A \in \mathscr{C},\ P \in \mathscr{E},\ p : P \to FA \rangle$. Morphisms from $\langle A, P, p \rangle$ to $\langle A', P', p' \rangle$ are pairs $\langle g : A \to A', h : P \to P' \rangle$ making the following square commute:

$$\begin{array}{ccc} P & \xrightarrow{h} & P' \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle p'} \\ FA & \xrightarrow[Fg]{} & FA' \end{array}$$

The category $\mathscr{E} \downarrow F$ is also called the *Artin gluing* of $\mathscr{C}$ and $\mathscr{E}$ along the functor F, especially when F is a left-exact functor between two toposes $\mathscr{C}$ and $\mathscr{E}$.

5.3.1∗5 Example. For instance, to prove some meta-theoretic property about closed terms of a theory S (such as canonicity), we let the category $\mathscr{C}$ be the category of judgements $\textsc{Jdg}\, S$, the model M be $\mathrm{Id} : \textsc{Jdg}\, S \to \textsc{Jdg}\, S$, the category $\mathscr{E}$ be the category of sets, and the functor $F : \mathscr{C} \to \mathscr{E}$ be the global section functor

$$\Gamma := \textsc{Hom}(1, -) : \textsc{Jdg}\, S \to \textsc{Set}.$$

The glued category $\textsc{Set} \downarrow \Gamma$ then has as objects tuples

$$\langle A \in \textsc{Jdg}\, S,\ P \in \textsc{Set},\ p : P \to \Gamma A \rangle,$$

which can be viewed as (proof-relevant) predicates on closed deductions of judgements A: for closed deduction $x \in \Gamma A$ of the judgement A, the set $p^{-1}(x) \subseteq P$ is viewed as the set of evidence that x satisfies this logical predicate. Defining a model $M^* : \textsc{Jdg}\, S \to \textsc{Set} \downarrow \Gamma$ making the following triangle commute

$$\begin{array}{ccc} \textsc{Jdg}\, S & \xrightarrow{M^*} & \textsc{Set} \downarrow \Gamma \\ & \searrow^{\mathrm{Id}} & \downarrow \pi \\ & & \textsc{Jdg}\, S \end{array}$$

amounts to (1) defining a syntactic logical relation and (2) proving the 'fundamental lemma' (every term satisfies the logical relation of its judgement).

5.3.1∗6. When the object theory S is complex, the gluing perspective provides very useful guidance on logical-relation proofs, as it reduces a proof to constructing a model in a certain category. However, manually manipulating the objects in the glued category can sometimes still be cumbersome.

Synthetic Tait computability (STC) [Sterling 2021], also known as *logical relation as types* [Sterling and Harper 2021], aims precisely to abstract away manually manipulating the glued category. The ideas of STC are that

1. instead of gluing the category $\mathscr{C}$ with the topos $\mathscr{E}$, we glue the presheaf topos $\textsc{Pr}\,\mathscr{C}$ with $\mathscr{E}$, so the resulting glued category $\mathscr{G}$ is always a topos;
2. as a topos $\mathscr{G}$ has expressive internal languages based on dependent type theories [Maietti 2005] or higher-order logic [Lambek and Scott 1986]. Sterling moreover designed a handful of type connectives for $\mathscr{G}$ for manipulating objects $\mathscr{G}$ as logical predicates conveniently.

Hence, constructing models in $\mathscr{G}$ can be done entirely in a type-theoretic language, reducing a logical-relation proof to a programming exercise.

5.3.2 Basic Categorical Properties of the Glued Topos

5.3.2∗1. Before we delve into STC, let us first familiarise ourselves with some basic properties of the glued topos for logical predicates. The purpose of this subsection is mainly to develop some intuition about the glued topos by seeing how it works *concretely*, so the details are somewhat irrelevant for future sections.

5.3.2∗2 Notation. In this subsection we work in general with a small category $\mathscr{S}$, and denote the glued category of $\textsc{Pr}\,\mathscr{S}$ along the global section functor $\Gamma : \textsc{Pr}\,\mathscr{S} \to \textsc{Set}$ by $\mathscr{G}$. When it causes no confusion, we will sometimes refer to an object $\langle A, S, p\rangle \in \mathscr{G}$ by $p : S \to \Gamma A$. Later in our canonicity proof, $\mathscr{S}$ is going to be instantiated as the category of judgement $\textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}$ for System $\mathrm{F}^{\mathrm{ha}}_{\omega}$.

5.3.2∗3. Let $F : \mathscr{C} \to \mathscr{E}$ be a functor between two categories. It is known in topos theory that the Artin gluing $\mathscr{E} \downarrow F$ has the following closure properties:

1. $\mathscr{E} \downarrow F$ is an elementary topos if $\mathscr{C}$ and $\mathscr{E}$ are elementary toposes and F is left-exact [Johnstone 2002, Example A2.1.12];
2. $\mathscr{E} \downarrow F$ is a presheaf topos if $\mathscr{C}$ and $\mathscr{E}$ are presheaf toposes and F preserves *wide pullbacks*, i.e. small products in slice categories [Carboni and Johnstone 1995, Theorem 4.1; Leinster 2004, C.3.4];
3. $\mathscr{E} \downarrow F$ is a Grothendieck topos if $\mathscr{C}$ and $\mathscr{E}$ are and F is left-exact and κ-accessible for some regular cardinal κ [Artin et al. 1972, Theorem 9.5.4].

In this thesis, we only need a very simple case for presheaf toposes: the gluing $\mathscr{G}$ along the global section functor $\Gamma : \textsc{Pr}\,\mathscr{S} \to \mathscr{S}$ is equivalent to the presheaf topos over $\mathscr{S}_{\top}$, the category that adjoins a new terminal object $\top$ to $\mathscr{S}$.

5.3.2∗4 Lemma. There is an equivalence $F : \mathscr{G} \cong \mathrm{Pr}\,\mathcal{S}_\top : F^{-1}$ between the glued category and the presheaf category over $\mathcal{S}_\top$.

Proof. Explicitly, F maps every $(p : P \to \Gamma A) \in \mathscr{G}$ to the presheaf $Fp : \mathcal{S}_\top^{\mathrm{op}} \to \mathrm{Set}$ whose action on the objects of $\mathcal{S}_\top$ is

$$(Fp)\ \top = P, \qquad\qquad (Fp)\ X = AX \text{ for all } X \in \mathcal{S},$$

and its action on the morphisms $! : X \to \top$ to the adjoined terminal $\top$ is

$$(Fp)\ (! : X \to \top) = (\lambda e.\ (p\ e)_X\ *) : P \to AX$$

where $p\ e$ is a natural transformation $1 \to A \in \mathrm{Pr}\,\mathcal{S}$, so $(p\ e)_X\ *$ is an element of AX; the action of Fp on the other morphisms $a : X \to Y \in \mathcal{S}_\top$ is

$$(Fp)\ (a : X \to Y) = Aa : AY \to AX.$$

The definition of Fp preserves identity trivially and preserves composition because of the naturality of $p\ e : 1 \to A \in \mathrm{Pr}\,\mathcal{S}$.

For the other direction, F^{-1} maps a functor $H : \mathcal{S}_\top^{\mathrm{op}} \to \mathrm{Set}$ to the object

$$\langle H \circ i : \mathcal{S}^{\mathrm{op}} \to \mathrm{Set},\ H\ \top \in \mathrm{Set},\ h : H\ \top \to \Gamma(H \circ i)\rangle,$$

where $i : \mathcal{S}^{\mathrm{op}} \to \mathcal{S}_\top^{\mathrm{op}}$ is the inclusion functor, h is the function sending every $e \in H\ \top$ to the natural transformation $h(e) : 1 \to H \circ i$ satisfying

$$h(e)_X = (\lambda *.\ (H\ !)\ e) : \{*\} \to HX,$$

whose naturality follows from the functoriality of H.

The natural isomorphisms $\eta : \mathrm{Id} \cong FF^{-1}$ and $\epsilon : F^{-1}F \cong \mathrm{Id}$ are simply the identities, so this equivalence is in fact a strict isomorphism of categories. □

5.3.2∗5 Remark. The lemma above has a slightly more general form. Let $\mathscr{R}$ be a small category and $\rho : \mathscr{R} \to \mathrm{Pr}\,\mathcal{S}$ be an arbitrary functor (which is the same as a profunctor $\mathcal{S}^{\mathrm{op}} \times \mathscr{R} \to \mathrm{Set}$). Define a functor $N_\rho : \mathrm{Pr}\,\mathcal{S} \to \mathrm{Pr}\,\mathscr{R}$ by sending every $A \in \mathrm{Pr}\,\mathcal{S}$ to $\mathrm{Hom}_{\mathrm{Pr}\,\mathcal{S}}(\rho -, A) \in \mathrm{Pr}\,\mathscr{R}$. The Artin gluing along N_ρ is equivalent to the presheaf category over the category $\mathscr{C}$ (called the *collage* of the profunctor ρ) whose objects are $\mathrm{Obj}\,\mathcal{S} + \mathrm{Obj}\,\mathscr{R}$ and whose morphisms are

$$\mathscr{C}(\iota_1 s', \iota_1 s) := \mathcal{S}(s', s) \qquad \mathscr{C}(\iota_2 r, \iota_2 r') := \mathscr{R}(r, r')$$
$$\mathscr{C}(\iota_1 s, \iota_2 r) := \rho(r)(s) \qquad \mathscr{C}(\iota_2 r, \iota_1 s) := \emptyset$$

with the evident composition operation. This recovers the lemma above by setting $\mathscr{R}$ to be the terminal category and ρ to be $* \mapsto 1_{\mathrm{Pr}\,\mathcal{S}}$.

5.3.2∗6. The topos $\mathrm{Pr}\,\mathcal{S}$ is equivalent to the slice category $\mathscr{G}/\mathfrak{ob}$ over

$$\mathfrak{ob} := \langle 1 \in \mathrm{Pr}\,\mathcal{S},\ 0 \in \mathrm{Set},\ ! : 0 \to \Gamma 1\rangle \in \mathscr{G}.$$

The terminal object of $\mathscr{G}$ is $id : 1_{\mathrm{Set}} \to \Gamma 1_{\mathrm{Pr}\,\mathcal{S}}$, so $\mathfrak{ob}$ is a subterminal object:

$$\begin{array}{ccc} 0 & \xrightarrow{!} & 1 \\ {\scriptstyle !}\downarrow & & \downarrow{\scriptstyle id} \\ \Gamma 1_{\mathrm{Pr}\,\mathcal{S}} & \xrightarrow{\Gamma id} & \Gamma 1_{\mathrm{Pr}\,\mathcal{S}} \end{array}$$

Therefore $\mathrm{Pr}\,\mathcal{S}$ is the *open subtopos* of $\mathscr{G}$ determined by the subterminal $\mathfrak{ob}$. The geometric embedding $(j^* \dashv j_*) : \mathrm{Pr}\,\mathcal{S} \to \mathscr{G}$ is concretely given by

$$j^*(p : S \to \Gamma A) = A \in \mathrm{Pr}\,\mathcal{S}, \qquad j_* A = (id : \Gamma A \to \Gamma A) \in \mathscr{G}.$$

5.3.2∗7. The *closed subtopos* of $\mathscr{G}$ determined by $\mathfrak{ob}$ is precisely the topos Set, which embeds into $\mathscr{G}$ by the following geometric morphism $(i^* \dashv i_*) : \mathrm{Set} \to \mathscr{G}$:

$$i^*(p : S \to \Gamma A) = S \in \mathrm{Set}, \qquad i_* S = (! : S \to \Gamma 1) \in \mathscr{G}.$$

5.3.2∗8. In the context of synthetic Tait computability, the open subtopos $\mathrm{Pr}\,\mathcal{S}$ is called the *object space*, since $\mathcal{S}$ is the category of judgements of the object theory. On the other hand, the closed subtopos Set is called the *meta space*, since it is where the meta-theoretic properties live.

5.3.2∗9. The idempotent monad $j_* j^*$ arising from the geometric embedding $j : \mathrm{Pr}\,\mathcal{S} \to \mathscr{G}$ will be denoted by $\bigcirc : \mathscr{G} \to \mathscr{G}$ and called the *open modality*:

$$\bigcirc(p : P \to \Gamma A) = (id : \Gamma A \to \Gamma A).$$

Intuitively, $\bigcirc$ erases the meta-theoretic information P from a logical predicate. Moreover, the open modality $\bigcirc$ is precisely the exponential functor $(-)^{\mathfrak{ob}}$ by $\mathfrak{ob}$.

5.3.2∗10. The idempotent monad $i_* i^*$ arising from the geometric embedding $i : \mathrm{Set} \to \mathscr{G}$ will be denote by $\bullet : \mathscr{G} \to \mathscr{G}$ and called the *closed modality*:

$$\bullet(p : P \to \Gamma A) = (! : P \to \Gamma 1).$$

Intuitively, $\bullet$ erases the object-theory-level information A from a logical predicate. The closed modality is also related to the subterminal $\mathfrak{ob}$: for every object $p \in \mathscr{G}$, the object $\bullet\, p$ is the following pushout in $\mathscr{G}$:

$$\begin{array}{ccc} \mathfrak{ob} \times p & \xrightarrow{\pi_2} & p \\ {\scriptstyle \pi_1}\downarrow & \ulcorner & \downarrow \\ \mathfrak{ob} & \longrightarrow & \bullet\, p \end{array}$$

5.3.2∗11. An object $p \in \mathscr{G}$ is called $\bigcirc$-*modal* if $\eta^\circ : p \to \bigcirc p$ is an isomorphism, and correspondingly $\bullet$-*modal* if $\eta^\bullet : p \to \bullet\, p$ is an isomorphism. The full subcategory of $\mathscr{G}$ spanned by $\bigcirc$-modal (resp. $\bullet$-modal) objects are equivalent to

$\mathrm{Pr}\,\mathcal{S}$ (resp. Set). Therefore, in the language of $\mathcal{G}$, we can talk about things from the object space and meta space by talking about $\bigcirc$-modal and $\bullet$-modal objects.

5.3.2∗12. Every presheaf topos $\mathrm{Pr}\,\mathscr{C}$ has a subobject classifier Ω, which can be logically viewed as a universe of propositions. Concretely, the subobject classifier Ω maps every object $A \in \mathscr{C}$ to the set of *sieves* on A:

$$\Omega(A) = \{S \subseteq \mathrm{Obj}\,(\mathscr{C}/A) \mid \forall (f : B \to A) \in S, \forall (g : C \to B), (f \cdot g) \in S\}, \quad (5.10)$$

and on morphisms $f : B \to A$, Ω maps f to

$$(S \in \Omega(A)) \quad \mapsto \quad \{g : C \to B \mid (f \cdot g) \in S\} \in \Omega(B).$$

The global truth morphism $t : 1 \to \Omega$ is $t_A = (\lambda *.\ \mathrm{Obj}\,(\mathscr{C}/A))$.

If we intuitively view a morphism $f : B \to A$ as the world B evolves to the world A via f, then the intuition for every $S \in \Omega(A)$ is a truth value in world A that tells us not just 'true or false' but more informatively *when* it is true: $(f : B \to A) \in S$ means that S is true in all worlds before the world B, including B itself, which evolves to A via f.

5.3.2∗13. Transporting the formula (5.10) along the equivalence in Lemma 5.3.2∗4, the glued category $\mathcal{G}$ has a subobject classifier

$$\Omega_{\mathcal{G}} := \langle \Omega_{\mathrm{Pr}\,\mathcal{S}} \in \mathrm{Pr}\,\mathcal{S},\ (\Gamma\Omega_{\mathrm{Pr}\,\mathcal{S}}) + 1_{\mathrm{Set}} \in \mathrm{Set},\ [id, (\lambda *.\ t_{\Omega_{\mathrm{Pr}\,\mathcal{S}}})]\rangle$$

with the truth morphism being $\langle t_{\Omega_{\mathrm{Pr}\,\mathcal{S}}} : 1_{\mathrm{Pr}\,\mathcal{S}} \to \Omega_{\mathrm{Pr}\,\mathcal{S}},\ \lambda *.\ \mathrm{inr}\ *\rangle : 1_{\mathcal{G}} \to \Omega_{\mathcal{G}}$.

5.3.2∗14. Finally, we recall that every Grothendieck universe U in Set can be lifted to presheaf categories [Hofmann and Streicher 1999]. Let $\pi : \tilde{V} \to V$ be the lifting of U to $\mathrm{Pr}\,\mathcal{S}$ as given in 4.3.1∗3. Let also $\mathrm{Obj}_U(\mathcal{G})$ be the set of *U-small $\mathcal{G}$-objects* $\langle A, P, p\rangle$, by which we mean that the image of the functor $A : S^{\mathrm{op}} \to \mathrm{Set}$ and the set P are in U. Transporting along the equivalence of Lemma 5.3.2∗4, the lifting of U to $\mathcal{G}$ is a universe $\tau : \tilde{W} \to W$ in $\mathcal{G}$ where

$$W := \langle V \in \mathrm{Pr}\,\mathcal{S}, \quad \mathrm{Obj}_U(\mathcal{G}) \in \mathrm{Set}, \quad (\lambda \langle A, P, p\rangle.\ \ulcorner A \urcorner)\rangle$$

and $\ulcorner A \urcorner : 1 \to V$ is the code of the U-small presheaf A. The object $\tilde{W}$ is slightly more complex and we shall omit it here for simplicity.

5.3.2∗15. The open and closed modalities $\bigcirc$ and $\bullet$ can be both internalised as endomorphisms $\widehat{\bigcirc}, \widehat{\bullet} : W \to W$ on the universe W in $\mathcal{G}$. Explicitly,

$$\widehat{\bigcirc} = \langle id : V \to V,\ (\lambda G.\ \bigcirc G)\rangle$$
$$\widehat{\bullet} = \langle V \xrightarrow{!} 1 \xrightarrow{\ulcorner 1 \urcorner} V,\ (\lambda G.\ \bullet G)\rangle,$$

where $\ulcorner 1 \urcorner : 1 \to V$ is the code of the terminal object $1 \in \mathrm{Pr}\,\mathcal{S}$ in the universe V.

5.3.3 The Type Theory of STC

5.3.3∗1. Let S be a signature in LccLF. The idea of synthetic Tait computability is to use a type theory to do constructions in the glued topos $\mathscr{G}$ of $\textsc{Pr}\,(\textsc{Jdg}\,S)$ and $\textsc{Set}$ along the global section functor; we will refer to this type theory by TTstc.

The type theory TTstc is a dependent type theory with the following type formers that can be interpreted in $\mathscr{G}$:

1. the structure of an elementary topos Ω (5.3.3∗3),
2. a tower of predicative universes (5.3.3∗14),
3. a proposition $\mathfrak{ob} : \Omega$ and an $\mathfrak{ob}$-partial model of S (5.3.3∗16), and
4. strict glue types (5.3.3∗21).

We will present these type formers one-by-one in this subsection.

5.3.3∗2 Notation. Although we can formally define TTstc as a signature in LccLF, our purpose is not to *study* TTstc but to *use* it, so we will present TTstc in the style of inference rules, which some readers may find more familiar.

All the judgements of TTstc are stable under substitution, so the inference rules of TTstc all have an ambient context '$\Gamma \vdash$'. We shall omit the ambient context Γ in the inference rules below, only tracking the *change* of context. For example, the formation rule for Π-types would be written as

$$\frac{A \text{ type} \qquad (a : A) \vdash B \text{ type}}{\Pi\, A\, B \text{ type}}$$

5.3.3∗3 Axiom (TTstc-Topos). TTstc has the following type formers:

1. unit type 1, empty type 0, coproduct types $A + B$, Π-types $(a : A) \to B$, Σ-types $\Sigma(a : A).\ B$, extensional identity types $a = b$;
2. a universe Ω such that (1) it *classifies all propositions*, in the sense that if a type A satisfies $(a, b : A) \to (a = b)$, then there is $\ulcorner A \urcorner : \Omega$ with an isomorphism $\ulcorner A \urcorner \cong A$; (2) it is *univalent*: if $A, B : \Omega$ and $A \cong B$, then $A = B$; and (3) it is *proof irrelevant*: if $A : \Omega$ and $p, q : A$, then $p = q$.

5.3.3∗4. Semantically, the type formers in Axiom 5.3.3∗3 correspond to the structure of *elementary toposes*: the first item above corresponds to the structure of a locally cartesian closed category with pullback-stable finite coproducts. The second item corresponds to a subobject classifier Ω with the true proposition being $\ulcorner 1 \urcorner : \Omega$. Incidentally, it is well known that coproducts can be constructed from other axioms of elementary toposes, but the construction is somewhat complex so we just include coproducts in the definition of TTstc.

5.3.3∗5. A handful of very useful type formers can be derived from the structure of elementary toposes. Let us have a look at these derived type formers first before the next part of the definition of TTstc.

5.3.3∗6 (Subtypes). For a type A and a function $P : A \to \Omega$, we define

$$\{x : A \mid P(x)\} \;:=\; \Sigma(a : A)\; P(a).$$

Since Ω is a proof-irrelevant universe of propositions, we will informally treat $\{x : A \mid P(x)\}$ as a subtype of A, eliding the unique proof of the proposition $P(x)$ and the pairing/projections:

$$\frac{a : A \qquad \_ : P(a)}{a : \{x : A \mid P(x)\}} \qquad \frac{a : \{x : A \mid P(x)\}}{a : A} \qquad \frac{a : \{x : A \mid P(x)\}}{\_ : P(a)}$$

Of course, when using this notation, it is *our* job to ensure that we only use $a : A$ as an element of $\{x : A \mid P(x)\}$ when an element of $P(a)$ is available.

5.3.3∗7 (Extension types). For a type A, a proposition $\phi : \Omega$, and let a be a *partial element* of A that is defined only when ϕ holds, i.e. $a : \phi \to A$, we will write

$$\{A \mid \phi \hookrightarrow a\} \;:=\; \{x : A \mid (p : \phi) \to x = a(p)\}$$

for the type of A-elements that are strictly equal to a when ϕ holds. If we have a partial element of A given as an implicit function $a : \{\phi\} \to A$, we also write $\{A \mid \phi \hookrightarrow a\}$ for $\{x : A \mid \{\phi\} \to (x = a)\}$.

5.3.3∗8 (Universal quantification). For an arbitrary type A and type family $B : A \to \Omega$ in the universe Ω, the dependent function type $(x : A) \to B\ x$ can also be shown to be a proposition, i.e. $(f, g : (x : A) \to B\ x) \to f = g$. Therefore there is a type $\lceil (x : A) \to B\ x \rceil : \Omega$ isomorphic to $(x : A) \to B\ x$.

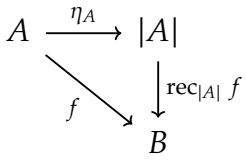
We will write $\forall(x : A).\ B\ x$ as a synonym for $\lceil (x : A) \to B\ x \rceil$, and we will elide the isomorphism between $\forall(x : A).\ B\ x$ and $(x : A) \to B\ x$. In fact, it is always possible to set up the interpretation of TTstc in any glued topos in a way that these two types are strictly equal.

5.3.3∗9 (Propositional truncation). For an arbitrary type A, its *propositional truncation* $|A| : \Omega$ is defined to be $\forall(B : \Omega).\ (A \to B) \to B$. Let $\eta_A : A \to |A|$ be

$$\eta_A\ (a : A)\ (B : \Omega)\ (k : A \to B) = k\ a.$$

We will also write $|a|$ for $\eta_A\ a$. The universal property for $\eta_A : A \to |A|$ is that

for all $B : \Omega$, every $f : A \to B$ uniquely factors via η_A:



where the function $\mathrm{rec}_{|A|}\ f := \lambda p.\ p\ B\ f$. The statement of this universal property can be slightly generalised by allowing B to be any type that satisfies $(x, y : B) \to x = y$, as there is an isomorphism $B \cong \lceil B \rceil$ for some $\lceil B \rceil : \Omega$.

5.3.3∗10 (Existential quantification). For any type A and $B : A \to \Omega$, we will write $\exists(x : A).\ B\ x$ for the propositional truncation $|\Sigma(x : A).\ (B\ x)|$.

As an example, for any two types A and B, the invertibility of a function $f : A \to B$ can be expressed by the following:

$$\begin{aligned} &\mathit{is\text{-}iso} : (f : A \to B) \to \Omega \\ &\mathit{is\text{-}iso}\ f = \exists(g : B \to A).\ \mathit{inv}\ f\ g \end{aligned}$$

$$\begin{aligned} &\mathit{inv} : (f : A \to B) \to (g : B \to A) \to \Omega \\ &\mathit{inv}\ f\ g = (f \cdot g = \mathit{id}_B) \wedge (g \cdot f = \mathit{id}_A) \end{aligned}$$

where $f \cdot g$ is function composition and $\mathit{id}_A : A \to A$ is the identity function. A nice, and somewhat surprising, property of elementary toposes is that we can always construct the inverse from just invertibility:

$$\begin{aligned} &\mathit{inv\text{-}of} : (f : A \to B) \to \mathit{is\text{-}iso}\ f \to \Sigma(g : B \to A).\ \mathit{inv}\ f\ g \\ &\mathit{inv\text{-}of}\ f\ i = \mathrm{rec}_{|\mathit{is\text{-}iso}\ f|}\ \ \mathit{id}\ i \end{aligned}$$

This is well typed because $\Sigma(g : B \to A).\ \mathit{inv}\ f\ g$ is in fact a proposition, in the sense that for every $x, y : \Sigma(g : B \to A).\ \mathit{inv}\ f\ g$, we have $x = y$ since the inverse of f is unique. Therefore $\Sigma(g : B \to A).\ \mathit{inv}\ f\ g$ is classified by the universe Ω, so we can recover it from its propositional truncation. In contrast, we cannot directly eliminate $\mathit{is\text{-}iso}\ f$ to $B \to A$, because the latter is not a proposition, although it seemingly has less data than the proposition $\Sigma(g : B \to A).\ \mathit{inv}\ f\ g$.

5.3.3∗11 (Conjunction and disjunction). Similar to the situation of universal quantification, for $A : \Omega$ and $B : \Omega$, their conjunction $A \wedge B : \Omega$ is defined to be the code $\lceil A \times B \rceil$ of their product in Ω. On the other hand, their disjunction $A \vee B$ is the propositional truncation $|A + B|$ of their coproduct.

5.3.3∗12 (Finite colimits). With the above logical apparatus, the concrete construction of finite colimits $\mathrm{colim}_i F_i$ in Set as the quotient set of $\coprod_i F_i$ by the minimal equivalence relation containing the relation R

$$R = \{((i, x), (j, y)) \mid \exists f : i \to j.\ Ff(x) = y\}$$

can be directly carried out in TTstc. In particular, the *quotient type* A/R in TTstc for a type A and an equivalence relation $R : A \to A \to \Omega$ is defined as

$$A/R = \{P : A \to \Omega \mid \exists(a : A).\ \forall(a' : A).\ P(a') = R(a, a')\}$$

with the 'equivalence class' function $[\_] : A \to A/R$ given by $[a] = \lambda a'.\ R(a, a')$.

5.3.3∗13. Coming back to the axioms of TTstc, the next part is universes.

5.3.3∗14 Axiom (TTstc-Universe). TTstc has a cumulative tower of predicative universes $U_0 : U_1 : \cdots : U_i : \cdots$, for every natural number i. Each of the universe is closed under the unit type, the empty types, Π types, Σ types, extensional equality types, and inductive types (W-types). Moreover, the universe of propositions is classified by U_0, i.e. $\Omega : U_0$.

5.3.3∗15. According to Chapter 4, a (diagrammatic) model of the signature S is an element of the record type $[\![S]\!]$ that has the same fields as S with all $\mathbb{J}$ replaced by U. For example, U-small models of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ are elements of the type

$$\begin{array}{l}
\textsc{record}\ [\![\mathrm{F}^{\mathrm{ha}}_{\omega}]\!]_U\ \textsc{where} \\
\quad ki : U \\
\quad el : ki \to U \\
\quad ty : ki \\
\quad _\Rightarrow_k_ : ki \to ki \to ki \\
\quad tm : el\ ty \to mu \\
\quad \ldots \quad \text{-- all other declarations of System } \mathrm{F}^{\mathrm{ha}}_{\omega}
\end{array}$$

On the other hand, the Yoneda embedding $\mathrm{Y} : \textsc{Jdg}\, S \to \textsc{Pr}\,(\textsc{Jdg}\, S)$ is a (functorial) model of S in $\textsc{Pr}\,(\textsc{Jdg}\, S) \cong \mathcal{G}/\mathfrak{ob}$, where $\mathfrak{ob}$ is the subterminal object in 5.3.2∗6 that renders $\textsc{Pr}\,(\textsc{Jdg}\, S)$ as the open subtopos of $\mathcal{G}$. This situation is axiomatised in TTstc as follows.

5.3.3∗16 Axiom (TTstc-Obj). TTstc has constants $\mathfrak{ob} : \Omega$ and $M : \{\mathfrak{ob}\} \to [\![S]\!]_{U_0}$.

5.3.3∗17. The modalities $\bigcirc, \bullet : \mathcal{G} \to \mathcal{G}$ that we saw in Section 5.3.2 can be internally defined in TTstc. For every universe U in TTstc, we define

$$\begin{array}{l}
\bigcirc : U \to U \\
\bigcirc\ A = (\{\mathfrak{ob}\} \to A)
\end{array}$$

A type A is called $\bigcirc$-*modal* if the function $\eta^{\circ}_A := (\lambda a.\ \lambda\{z : \mathfrak{ob}\}.\ a) : A \to \bigcirc A$ is an isomorphism, i.e. $\bigcirc$-*modal* A = *is-iso* η°_A for *is-iso* defined in 5.3.3∗10.

One useful observation is there is a function $\delta : \bigcirc U \to U$ for the universe U, sending every $\mathfrak{ob}$-partial type $X : \{\mathfrak{ob}\} \to U$ to $\{\mathfrak{ob}\} \to X$; here the application of X to the proof of $\mathfrak{ob}$ is implicit, so the type is $\{z : \mathfrak{ob}\} \to X\ \{z\}$ more explicitly.

This is not an algebra for $\bigcirc$ as a *monad* though, since when X is not $\bigcirc$-modal, $\delta(\eta^{\circ}_{U} X) = \{\mathfrak{ob}\} \to X$ is not isomorphic, let alone equal, to X.

5.3.3∗18. The closed modality $\bullet A$ is defined as a pushout *internal* to TTstc:

$$\begin{array}{ccc} \mathfrak{ob} \times A & \xrightarrow{\pi_2} & A \\ {\scriptstyle \pi_1}\downarrow & \ulcorner & \downarrow{\scriptstyle \eta^{\bullet}_{A}} \\ \mathfrak{ob} & \xrightarrow[pt]{} & \bullet A \end{array} \tag{5.11}$$

or in terms of a *quotient inductive type* [Altenkirch et al. 2018; Fiore et al. 2022]:

$$\begin{array}{l} \textsc{record}\ \bullet A : U\ \textsc{where} \\ \quad \eta^{\bullet}_{A} : A \to \bullet A \\ \quad pt\ : \{\mathfrak{ob}\} \to \bullet A \\ \quad eq\ : \{\mathfrak{ob}\} \to (a : A) \to \eta^{\bullet}_{A}\ a = pt \end{array}$$

The type $\bullet A$ can be explicitly constructed using quotient and coproduct types, in the same way as constructing pushouts in Set, which we shall not belabour here. To eliminate from the type $\bullet A$, we use the following syntax:

$$\frac{\begin{array}{cc} B : \bullet A \to U_i & a : A \vdash b : B\ (\eta^{\bullet}_{A}\ a) \\ \_ : \mathfrak{ob} \vdash b' : \eta^{\bullet}_{A}\ pt & \_ : \mathfrak{ob}, a : A \vdash b = b' \end{array} \qquad c : \bullet A}{\textsc{case}\ c\ \textsc{of}\ \{\eta^{\bullet}_{A}\ a \mapsto b;\ pt \mapsto b'\} : B\ c} \tag{5.12}$$

satisfying the usual β and η-rules.

A type A is called $\bullet$-*modal* if the function $\eta^{\bullet}_{A} : A \to \bullet A$ is an isomorphism

$$\bullet\text{-}modal\ A\ :=\ is\text{-}iso\ \eta^{\bullet}_{A}.$$

In this case, we will write $\epsilon^{\bullet}_{A} : \bullet A \to A$ for the inverse of $\eta^{\bullet}_{A} : A \to \bullet A$.

5.3.3∗19 Lemma. A type A is $\bullet$-modal iff $\bigcirc A$ is isomorphic to the unit type 1.

Proof. Assuming $\bullet$-*modal* A, we can define $f : 1 \to \bigcirc A$ by

$$f\ * = \lambda\{z : \mathfrak{ob}\}.\ \epsilon^{\bullet}_{A}\ (pt\ z),$$

where $pt : \mathfrak{ob} \to \bullet A$ is the bottom arrow in the pushout (5.11). We need to show that f is the inverse of $(\lambda a.\ *) : \bigcirc A \to 1$. For all $a : \bigcirc A$, we need to show that

$$a\ =\ (\lambda\{z : \mathfrak{ob}\}.\ \epsilon^{\bullet}_{A}\ (pt\ z)).$$

Given $z : \mathfrak{ob}$, $pt\ z = \eta^{\bullet}_{A}\ (a\ \{z\})$ by the property of the pushout. Hence we have $\epsilon^{\bullet}_{A}\ (pt\ z) = \epsilon^{\bullet}_{A}\ (\eta^{\bullet}_{A}\ (a\ \{z\}))$, which is also equal to $a\ \{z\}$ because $\epsilon^{\bullet}_{A}$ is the inverse of $\eta^{\bullet}_{A}$. The other direction is trivial since $*$ is the unique element of 1.

Now assuming $\bigcirc A \cong 1$, let a be the unique element of $\bigcirc A$. We define

$$\begin{aligned}&\epsilon^\bullet_A : \bullet A \to A\\ &\epsilon^\bullet_A\ c = \textsc{case}\ c\ \textsc{of}\ \{\eta^\bullet_A\ a' \mapsto a'; pt \mapsto a\}\end{aligned}$$

The side condition $(\_ : \mathfrak{ob}, a' : A \vdash a = a')$ for making this case split (5.12) is satisfied because $A \cong 1$ under $\mathfrak{ob}$. The functions $\epsilon^\bullet_A$ and $\eta^\bullet_A$ are inverses following from the universal property of the pushout $\bullet A$. □

5.3.3∗20. Recall that objects in the glued topos $\mathscr{G}$ are tuples $\langle A, B, p\rangle$. Presheaves $A \in \textsc{Pr}\,\textsc{Jdg}\,S$ can be internally expressed in TTstc as $\bigcirc$-modal types, or equivalently, $\mathfrak{ob}$-partial types $A : \{\mathfrak{ob}\} \to U$ in a universe U. The set B and the function $p : B \to \Gamma A$ can be internally expressed in TTstc as $\bullet$-modal type families $B : (\{\mathfrak{ob}\} \to A) \to U$. The next axiom of TTstc says that given such A and B, we can glue them together to form a new type that is strictly equal to A under $\mathfrak{ob}$.

5.3.3∗21 Axiom (TTstc-Glue). The type theory TTstc has *strict glue types*:

$$\frac{A : \bigcirc U \qquad B : (\{\mathfrak{ob}\} \to A) \to \{X : U \mid \bullet\text{-}\mathit{modal}\ X\}}{(a : A) \ltimes B\ a : \{U \mid \mathfrak{ob} \hookrightarrow A\}}$$

The elements of the glue type $(a : A) \ltimes B\ a$ are specified by an isomorphism to those of the Σ-type $\Sigma\ (\{\phi\} \to A)\ B$:

$$\mathit{glue} : \{\ \Sigma(a : \{\mathfrak{ob}\} \to A).\ B\ a \cong (a : A) \ltimes B\ a\ \mid \mathfrak{ob} \hookrightarrow \pi_1\}.$$

which restricts to $\pi_1 : \Sigma(\{\mathfrak{ob}\} \to A)\ B \to A$ under the proposition $\mathfrak{ob}$.

5.3.3∗22. We define the following notation for $a : \{\mathfrak{ob}\} \to A$ and $b : B\ a$ for constructing elements of the strict glue type:

$$[\mathfrak{ob} \hookrightarrow a \mid b] :=\ \mathit{glue}\ (a, b) : (a : A) \ltimes B\ a.$$

Given an element $g : (a : A) \ltimes B\ a$, when $\mathfrak{ob}$ holds, $(a : A) \ltimes B\ a$ is equal to the type A, so we can directly use g as an element of A when $\mathfrak{ob}$ holds. To access the second component of a glued element conveniently, we define

$$\begin{aligned}&\mathit{unglue}\ : (g : (a : A) \ltimes B\ a) \to B\ (\lambda\{\_ : \mathfrak{ob}\}.\ g)\\ &\mathit{unglue}\ g = \pi_2\ (\mathit{glue}^{-1}\ g)\end{aligned}$$

We will also use a pattern-matching syntax to define functions out of glue types. For example, the following definition

$$\begin{aligned}&f : (g : (a : A) \ltimes B\ a) \to C\ g\\ &f\ [\mathfrak{ob} \hookrightarrow a \mid b] = e\end{aligned} \tag{5.13}$$

is understood as the definition $f\ g = e[(\lambda\{\_ : \mathfrak{ob}\}.\ g)/a, (\mathit{unglue}\ g)/b]$.

5.3.3∗23. We can not only glue but also tear types apart. Given any type $A : U$, we can tear it to an object-space fragment A° and a meta-space fragment $A^\bullet$:

$$\begin{aligned} A^\circ &: \bigcirc U & A^\bullet &: (\{\mathfrak{ob}\} \to A) \to U^\bullet \\ A^\circ &= \eta^\circ_U\, A & A^\bullet &= \lambda o.\ \{A \mid \mathfrak{ob} \hookrightarrow o\} \end{aligned}$$

where $U^\bullet := \{A : U \mid \bullet\text{-}\mathit{modal}\ A\}$ is the subuniverse of $\bullet$-modal types. The type $\{A \mid \mathfrak{ob} \hookrightarrow a\}$ is $\bullet$-modal because it is a singleton under $\mathfrak{ob}$ (Lemma 5.3.3∗19).

If we glue these two fragments together, we get a type isomorphic to A:

$$\begin{aligned} \mathit{fwd} &: A \to (o : A^\circ) \ltimes A^\bullet\ o & \mathit{bwd} &: ((o : A^\circ) \ltimes A^\bullet\ o) \to A \\ \mathit{fwd}\ a &= [\mathfrak{ob} \hookrightarrow \lambda\{\_ : \mathfrak{ob}\}.\ a \mid a] & \mathit{bwd}\ [\mathfrak{ob} \hookrightarrow o \mid c] &= c \end{aligned}$$

These two functions are indeed mutual inverses: for all $a : A$,

$$\mathit{bwd}\ (\mathit{fwd}\ a) = \mathit{bwd}\ [\mathfrak{ob} \hookrightarrow \lambda\{\_ : \mathfrak{ob}\}.\ a \mid a] = a;$$

for all $[\mathfrak{ob} \hookrightarrow o \mid c]$, by definition $\mathit{fwd}\ (\mathit{bwd}\ [\mathfrak{ob} \hookrightarrow o \mid c]) = [\mathfrak{ob} \hookrightarrow c \mid c]$, but c has type $A^\bullet := \{c \mid \mathfrak{ob} \hookrightarrow o\}$, so $[\mathfrak{ob} \hookrightarrow c \mid c] = [\mathfrak{ob} \hookrightarrow o \mid c]$.

5.3.3∗24. Since every type of TTstc is isomorphic to a glue type, we can characterise function types of TTstc more extrinsically, which explicitises the idea that a map between logical predicates sends (proofs for) related input to (proofs for) related output. For all universe U, there is an isomorphism $\ltimes\text{-}\mathit{fun\text{-}iso}$:

$$\begin{gathered} \big((a : A) \ltimes P\ a\big) \to \big((b : B) \ltimes Q\ b\big) \\ \cong \\ (f : A \to B) \ltimes \big((a : \{\mathfrak{ob}\} \to A) \to P\ a \to Q\ (f\ a)\big) \end{gathered}$$

for all $A, B : \bigcirc U$, $P : (\{\mathfrak{ob}\} \to A) \to U^\bullet$, and $Q : (\{\mathfrak{ob}\} \to B) \to U^\bullet$, where $U^\bullet$ is the subuniverse $\{A : U \mid \bullet\text{-}\mathit{modal}\ A\}$ of $\bullet$-modal types. The two directions are

$$\mathit{fwd}\ g = [\mathfrak{ob} \hookrightarrow \lambda a.\ g\ a \mid \lambda a\ p.\ \mathit{unglue}\ (g\ [\mathfrak{ob} \hookrightarrow a \mid p])]$$

$$\mathit{bwd}\ [\mathfrak{ob} \hookrightarrow f \mid h]\ [\mathfrak{ob} \hookrightarrow a \mid p] = [\mathfrak{ob} \hookrightarrow f\ a \mid h\ a\ p]$$

It is routine calculation to check that these two directions are mutual inverses.

5.3.3∗25. We have finished the definition of the type theory TTstc, parameterised by an LF-signature S. The type theory TTstc can be interpreted in the glued presheaf topos $\mathscr{G}$, i.e. the Artin gluing of Pr (Jdg S) and Set along the global section functor Pr (Jdg S) $\to$ Set. We will not belabour the details of the interpretation here; we refer the reader to classic materials [Hofmann 1997; Jacobs 1999] on interpreting dependent type theories in presheaf categories and also to Gratzer [2023] for the semantics of the strict glue types.

In fact, the interpretation of TTstc can be done more generally. Let $\mathscr{C}$ be

a small LCC category, $M : \textsc{Jdg}\, S \to \mathscr{C}$ be a model of S in $\mathscr{C}$, $\mathscr{A}$ be a small category, and $\rho : \mathscr{A} \to \textsc{Pr}\,\mathscr{C}$ be a functor. The Artin gluing of $\textsc{Pr}\,\mathscr{C}$ and $\textsc{Pr}\,\mathscr{A}$ along the functor $X \mapsto \textsc{Hom}_{\textsc{Pr}\,\mathscr{C}}(\rho -, X) : \textsc{Pr}\,\mathscr{C} \to \textsc{Pr}\,\mathscr{A}$ is also a presheaf topos (Remark 5.3.2∗5) and models TTstc. This more general setup allows us to use TTstc to prove meta-theoretic properties pertaining to *open* terms of S, such as normalisation of S and definability of $\mathscr{C}$-morphisms by S-deductions.

5.4 Canonicity of System $\mathrm{F}^{\mathrm{ha}}_{\omega}$

5.4∗1. With the language of STC in our hand, we come back to the proof of canonicity of $\mathrm{F}^{\mathrm{ha}}_{\omega}$: every closed term of type *bool* is equal to either *tt* or *ff* (but not both). The plan of the proof is to define in TTstc a model of $\mathrm{F}^{\mathrm{ha}}_{\omega}$

$$M^* : \{[\![\mathrm{F}^{\mathrm{ha}}_{\omega}]\!]_{U_2} \mid \mathfrak{ob} \hookrightarrow M\} \tag{5.14}$$

such that (1) it restricts to the syntactic model $M : \{\mathfrak{ob}\} \to [\![\mathrm{F}^{\mathrm{ha}}_{\omega}]\!]_{U_0}$ from Axiom 5.3.3∗16, and that (2) canonicity is encoded in $M^*.\mathit{bool}$.

Since TTstc can be interpreted in the glued topos $\mathscr{G}$ of $\textsc{Pr}\,(\textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega})$ and Set, the definition of M^* gives a diagrammatic model of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ in $\mathscr{G}$. Then by Theorem 4.4∗10, we have an LCC-functor $\overline{M^*} : \textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega} \to \mathscr{G}$. Since M^* restricts to M under $\mathfrak{ob}$, we have a commutative triangle (up to some natural isomorphism):

$$\begin{array}{ccc} \textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega} & \xrightarrow{\overline{M^*}} & \mathscr{G} \\ & \overset{\cong}{\searrow^{\mathrm{Y}}} & \downarrow j^* \\ & & \textsc{Pr}\,(\textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}) \end{array} \tag{5.15}$$

Canonicity then follows from this diagram and the definition of $M^*.\mathit{bool}$.

5.4∗2 Remark. As a minor point, the reason why we have only an isomorphism (5.15) rather than a strict equality – even though M^* is strictly equal to M under $\mathfrak{ob}$ – is that the interpretation of M may *not* be exactly the Yoneda embedding. Although it *is* possible to carefully set up the interpretation of TTstc to make it true, there is no need to do so. Conversely, it is also possible to weaken our goal (5.14) to be a model that is just isomorphic to M under $\mathfrak{ob}$, but keeping track of this isomorphism in the proof is a complication rather than a simplification.

5.4∗3. We define the model M^* of System $\mathrm{F}^{\mathrm{ha}}_{\omega}$ piece by piece in the following sections. The reader may refer to Appendix A.1 for the full signature of $\mathrm{F}^{\mathrm{ha}}_{\omega}$.

5.4∗4 Notation. In the rest of this section, for every declaration *dec* in the signature of $\mathrm{F}^{\mathrm{ha}}_{\omega}$, we will write dec^* for $M^*.\mathit{dec}$ and just *dec* for $M.\mathit{dec}$. For example, $\mathit{ki}^* : \{U_2 \mid \mathfrak{ob} \hookrightarrow \mathit{ki}\}$ means $M^*.\mathit{ki} : \{U_2 \mid \mathfrak{ob} \hookrightarrow \mathit{ki}\}$.

5.4.1 Kinding

5.4.1∗1. The logical predicate model of the judgement of kinds (5.1) is

$$\begin{aligned} &ki^* : \{U_2 \mid \mathfrak{ob} \hookrightarrow ki\} \\ &ki^* = (\alpha : ki) \ltimes \{U_1 \mid \mathfrak{ob} \hookrightarrow el\ \alpha\} \end{aligned} \tag{5.16}$$

This uses the glue type (5.3.3∗21) correctly because the generic model M (5.3.3∗16) has type $\{\mathfrak{ob}\} \to [\![\mathrm{F}^{\mathrm{ha}}_{\omega}]\!]_{U_0}$, so the type of ki, or more explicitly $\lambda\{z : \mathfrak{ob}\}.\ (M\ \{z\}).ki$, is $\{\mathfrak{ob}\} \to U_0$, i.e. $\bigcirc U_0$. The type $\{U_1 \mid \mathfrak{ob} \hookrightarrow el\ \alpha\}$ is $\bullet$-modal because when $\mathfrak{ob}$ holds, all elements of $\{U_1 \mid \mathfrak{ob} \hookrightarrow el\ \alpha\}$ are equal to $el\ \alpha$, so the type $\{U_1 \mid \mathfrak{ob} \hookrightarrow el\ \alpha\}$ has exactly one element so isomorphic to the unit 1. By Lemma 5.3.3∗19, the type $\{U_1 \mid \mathfrak{ob} \hookrightarrow el\ \alpha\}$ is $\bullet$-modal.

More intuitively, the definition (5.16) is the *proof-relevant* logical predicate for kinds. A proof for a kind $\alpha : ki$ satisfying the predicate is a type $A : U_1$ that restrict to $el\ \alpha$ in the object space. Such a type A is a 'candidate' for the logical predicate for the kind α. This is the same idea as *reducibility candidates* in Girard's proof of strong normalisation of System F [Girard 1989].

5.4.1∗2. In accordance, the corresponding $M^*.el$ is as follows:

$$\begin{aligned} &el^* : \{ki^* \to U_1 \mid \mathfrak{ob} \hookrightarrow el\} \\ &el^*\ g = unglue\ g \end{aligned} \tag{5.17}$$

Let us more carefully examine how this definition type checks: the argument g has type $ki^* = (\alpha : ki) \ltimes \{U_1 \mid \mathfrak{ob} \hookrightarrow el\ \alpha\}$. Therefore $unglue\ g$ has type $\{U_1 \mid \mathfrak{ob} \hookrightarrow el\ g\}$ (note that under $\mathfrak{ob}$, the type of g is strictly equal to the type ki, thus it makes sense to write $el\ g$ in a context where $\mathfrak{ob}$ holds). Thus el^* is indeed a function $ki^* \to U_1$ that strictly restricts to el under $\mathfrak{ob}$.

5.4.1∗3. For kind-level functions, we need to define

$$\_\Rightarrow_k\_^* : \{ki^* \to ki^* \to ki^* \mid \mathfrak{ob} \hookrightarrow \_\Rightarrow_k\_\}$$

Let us derive the definition step-by-step. Our goal is to fill the hole ?0 in

$$[\mathfrak{ob} \hookrightarrow \alpha \mid A] \Rightarrow^*_k [\mathfrak{ob} \hookrightarrow \beta \mid B] \ =\ ?0 : \{ki^* \mid \mathfrak{ob} \hookrightarrow \alpha \Rightarrow_k \beta\},$$

where the variables in context have the following types

$$\alpha, \beta : \{\mathfrak{ob}\} \to ki \qquad A : \{U_1 \mid \mathfrak{ob} \hookrightarrow el\ \alpha\} \qquad B : \{U_1 \mid \mathfrak{ob} \hookrightarrow el\ \beta\}. \tag{5.18}$$

Since ki^* is a glue type (5.16), we can use the term former of glue types:

$$[\mathfrak{ob} \hookrightarrow \alpha \mid A] \Rightarrow^*_k [\mathfrak{ob} \hookrightarrow \beta \mid B] \ =\ [\mathfrak{ob} \hookrightarrow ?1 \mid ?2]$$

Since ?0 must restrict to $\alpha \Rightarrow_k \beta$ under $\mathfrak{ob}$, ?1 has to be $\alpha \Rightarrow_k \beta$:

$$[\mathfrak{ob} \hookrightarrow \alpha \mid A] \Rightarrow_k^* [\mathfrak{ob} \hookrightarrow \beta \mid B] \;=\; [\mathfrak{ob} \hookrightarrow \alpha \Rightarrow_k \beta \mid ?2\,]$$

The hole ?2 now has type $\{U_1 \mid \mathfrak{ob} \hookrightarrow el\ (\alpha \Rightarrow_k \beta)\}$; in other words, ?2 is a type in U_1 such that it restricts to $el\ (\alpha \Rightarrow_k \beta)$ when $\mathfrak{ob}$ holds. We again use the glue type to satisfy the restriction:

$$?2 := (f : el\ (\alpha \Rightarrow_k \beta)) \ltimes ?3\,.$$

Conceptually, ?2 is the logical predicate for the function kind $\alpha \Rightarrow_k \beta$. Readers experienced with traditional logical relations might expect ?3 to be the proposition asserting that f sends input $a : el\ \alpha$ satisfying the logical predicate A to output $f\ a : el\ \beta$ satisfying logical predicate B. However, here the predicates A and B are proof-relevant, so the correct definition of ?3 should be the *type* of functions sending proofs for $a : el\ \alpha$ satisfying A to proofs for $f\ a : el\ \beta$ satisfying B. This can be concisely expressed in TTstc as

$$?3 := \{A \to B \mid \mathfrak{ob} \hookrightarrow \Rightarrow_k\text{-}iso.fwd\ f\}$$

where $\Rightarrow_k$-*iso* is the isomorphism in $\mathrm{F}_\omega^{\mathrm{ha}}$ specifying function kinds:

$$\Rightarrow_k\text{-}iso : el\ (\alpha \Rightarrow_k \beta) \cong (el\ \alpha \to el\ \beta).$$

The function type $A \to B$ in TTstc is translated to exponentials in the glued topos $\mathscr{G}$, which takes care of 'sending related input to related output' by construction.

For the record, we have completed our initial goal $\_\Rightarrow_k\_^*$:

$$\begin{aligned} &\_\Rightarrow_k\_^* : \{ki^* \to ki^* \to ki^* \mid \mathfrak{ob} \hookrightarrow \_\Rightarrow_k\_\} \\ &[\mathfrak{ob} \hookrightarrow \alpha \mid A] \Rightarrow_k^* [\mathfrak{ob} \hookrightarrow \beta \mid B] = [\mathfrak{ob} \hookrightarrow \alpha \Rightarrow_k \beta \mid F] \end{aligned} \tag{5.19}$$

where F is the logical predicate for the function kind $\alpha \Rightarrow_k \beta$:

$$F := (f : el\ (\alpha \Rightarrow_k \beta)) \ltimes \{A \to B \mid \mathfrak{ob} \hookrightarrow \Rightarrow_k\text{-}iso.fwd\ f\}. \tag{5.20}$$

5.4.1∗4. We also need to exhibit the isomorphism $\Rightarrow_k$-*iso* (5.2) for M^*:

$$\Rightarrow_k\text{-}iso^* : \{\alpha^*, \beta^* : ki^*\} \to \{el^*\ (\alpha^* \Rightarrow_k^* \beta^*) \cong (el^*\ \alpha^* \to el^*\ \beta^*) \mid \mathfrak{ob} \hookrightarrow \Rightarrow_k\text{-}iso\}.$$

Again by pattern matching the input α^* and β^* as $[\mathfrak{ob} \hookrightarrow \alpha \mid A]$ and $[\mathfrak{ob} \hookrightarrow \beta \mid B]$ as in (5.18), after expanding out the definition of el^*, what we need to construct is an isomorphism $F \cong A \to B$ that restricts to $\Rightarrow_k$-*iso* under $\mathfrak{ob}$, where F is defined as in (5.20). We let the two directions of this isomorphism be

$$fwd\ [\mathfrak{ob} \hookrightarrow f \mid g] = g \qquad\qquad bwd\ h = [\mathfrak{ob} \hookrightarrow \Rightarrow_k\text{-}iso.bwd\ h \mid h]$$

where $h : A \to B$, $f : \{\mathfrak{ob}\} \to el\ (\alpha \Rightarrow_k \beta)$, and

$$g : \{A \to B \mid \mathfrak{ob} \hookrightarrow \Rightarrow_k\text{-}iso.fwd\ f\}.$$

These two functions are mutual inverses because

$$fwd\ (bwd\ h) = fwd\ ([\mathfrak{ob} \hookrightarrow \Rightarrow_k\text{-}iso.bwd\ h \mid h]) = h$$

and from the other direction,

$$bwd\ (fwd\ [\mathfrak{ob} \hookrightarrow f \mid g]) = bwd\ g = [\mathfrak{ob} \hookrightarrow \Rightarrow_k\text{-}iso.bwd\ g \mid g];$$

now by the type of g, $g = (\Rightarrow_k\text{-}iso.fwd\ f)$ under $\mathfrak{ob}$, so the above further equals

$$[\mathfrak{ob} \hookrightarrow \Rightarrow_k\text{-}iso.bwd\ (\Rightarrow_k\text{-}iso.fwd\ f) \mid g] = [\mathfrak{ob} \hookrightarrow f \mid g].$$

5.4.1∗5 (Realignment). The definition (5.20) of the logical predicate F for function kinds may look complicated at first, but it has a very intuitive explanation: F is basically the same as the type $A \to B$, except that its component in the object space, which is equal to $el\ \alpha \to el\ \beta$, is swapped for $el\ (\alpha \Rightarrow_k \beta)$ along the isomorphism $\Rightarrow_k\text{-}iso$, just like in the old days when a component of a personal computer can be replaced by a compatible part. This will be a recurring construction in the future, so for every universe U we define

$$\begin{aligned}
&realign : (A : U) \to (B : \{\mathfrak{ob}\} \to U) \to (\{\mathfrak{ob}\} \to B \cong A) \to \{U \mid \mathfrak{ob} \hookrightarrow B\}\\
&realign\ A\ B\ \phi = (b : B) \ltimes \{A \mid \mathfrak{ob} \hookrightarrow \phi.fwd\ b\}
\end{aligned}$$

$$\begin{aligned}
&realign\text{-}iso : (A : U) \to (B : \{\mathfrak{ob}\} \to U) \to (\phi : \{\mathfrak{ob}\} \to B \cong A)\\
&\qquad \to \{realign\ A\ B\ \phi \cong A \mid \mathfrak{ob} \hookrightarrow \phi\}\\
&(realign\text{-}iso\ A\ B\ \phi).fwd\ [\mathfrak{ob} \hookrightarrow b \mid a] = a\\
&(realign\text{-}iso\ A\ B\ \phi).bwd\ a = [\mathfrak{ob} \hookrightarrow \phi.bwd\ a \mid a]
\end{aligned}$$

This construction is called *realignment* [Sterling 2021, §3.3] on the universe U. In fact, realignment and strict glue types (Axiom 5.3.3∗21) are inter-definable: if we take *realign* and *realign-iso* as axioms, we can define strict glue types $(a : A) \ltimes B$ by realigning the dependent pair type $\Sigma(a : A).\ B$.

Using realignment, the definition (5.19) can be succinctly expressed as

$$[\mathfrak{ob} \hookrightarrow \alpha \mid A] \Rightarrow_k^* [\mathfrak{ob} \hookrightarrow \beta \mid B] = [\mathfrak{ob} \hookrightarrow \alpha \Rightarrow_k \beta \mid realign\ (A \to B) \Rightarrow_k\text{-}iso]$$

and $\Rightarrow_k\text{-}iso^*$ is simply $realign\text{-}iso\ (A \to B) \Rightarrow_k\text{-}iso$.

5.4.2 Typing

5.4.2∗1. We move on to the logical predicates for types and terms. Similar to function kinds, ty^* is ty glued together with some additional data:

$$\begin{aligned}
&ty^* : \{ki^* \mid \mathfrak{ob} \hookrightarrow ty\}\\
&ty^* = [\mathfrak{ob} \hookrightarrow ty \mid \ ?0 : \{U_1 \mid \mathfrak{ob} \hookrightarrow el\ ty\}\ ]
\end{aligned}$$

Since ?0 is a type in U_1 that is equal to *el ty* under $\mathfrak{ob}$, it can be a glue type:

$$ty^* = [\mathfrak{ob} \hookrightarrow ty \mid (A : el\ ty) \ltimes \text{?1}\ ] \tag{5.21}$$

which means that an element of the kind *ty* in the logical predicate model M^* is a syntactic type A together with the data ?1. It is natural to expect that the data ?1 associated to a type A is a (candidate of) logical predicate for the type A, which is just any type that restricts to $tm\ A$ under $\mathfrak{ob}$:

$$ty^* = [\mathfrak{ob} \hookrightarrow ty \mid (A : el\ ty) \ltimes \{U_0 \mid \mathfrak{ob} \hookrightarrow tm\ A\}\ ] \tag{$*$}$$

mimicking the kind structure (5.16) that we have seen earlier. However, this definition will not work when we come to *impredicative* polymorphic types $\forall\alpha.A$ later, because U_0 is not impredicative in the sense of being closed under Π-types $\Pi\ A\ B$ for *arbitrary* types A that are not necessarily in U_0.

5.4.2∗2. In every topos, we do have an impredicative universe – the universe Ω of *propositions*. Unfortunately, this universe is 'too small' for interpreting $\mathrm{F}^{\mathrm{ha}}_{\omega}$-types. If we have an element $A^* : \{\Omega \mid \mathfrak{ob} \hookrightarrow tm\ A\}$ for some object-space type $A : el\ ty$, when $\mathfrak{ob}$ holds, A^* is equal to $tm\ A$, but A^* is in the universe Ω, so we have $\{\mathfrak{ob}\} \to (a, b : tm\ A) \to a = b$, which means that the object-space type A has at most one element, and this is clearly not true in general.

5.4.2∗3. To find a way out, let us recall how traditional logical predicates/relations of System F work in, for example, Girard's [1989] normalisation proof. For every type A of System F, its logical predicate is a proof-irrelevant predicate on the set of terms of A, or equivalently, a function from terms of A to the set of classical propositions. Moreover, the logical predicate $P(t)$ of the impredicative polymorphic type $\forall\alpha.\ A$ is defined by 'for all types X and all candidate logical predicates Q over terms of X, the term $t\ [X]$ is related by the logical predicate of A with α replaced by (X, Q)'. This works because classical propositions are impredicative, so we can quantify over all X and Q.

5.4.2∗4. Mimicking the traditional approach, we first define a universe of *meta-space propositions* (which are just classical propositions $\{\top, \bot\}$ when TTstc is interpreted in the Artin gluing of the syntactic category and the category of sets):

$$\Omega^\bullet := \{p : \Omega \mid \bullet\text{-}modal\ p\}.$$

The universe $\Omega^\bullet$ inherits all the connectives that Ω has, including impredicative quantification. For example, if A is an arbitrary type and $B : A \to \Omega^\bullet$, the type $\forall(x : A).B\ x$ is in Ω, and when $\mathfrak{ob}$ holds, $B\ x \cong 1$ because a type is $\bullet$-modal iff it is isomorphic to 1 under $\mathfrak{ob}$ (Lemma 5.3.3∗19), so $\forall(x : A).B\ x = \forall(x : A).1 \cong 1$.

5.4.2∗5. Using $\Omega^\bullet$, we fill out the hole ?1 in ty^* (5.21) by

$$\begin{aligned} &ty^* : \{ki^* \mid \mathfrak{ob} \hookrightarrow ty\} \\ &ty^* = [\mathfrak{ob} \hookrightarrow ty \mid (A : el\ ty) \ltimes (\{\mathfrak{ob}\} \to tm\ A) \to \Omega^\bullet\ ] \end{aligned} \tag{5.22}$$

That is to say, the candidate of a logical predicate for a type A is given as a meta-space predicate $P : (\{\mathfrak{ob}\} \to tm\ A) \to \Omega^\bullet$.

Then tm^* glues terms $tm\ A$ of an object-space type A with the predicate P:

$$\begin{aligned} &tm^* : \{el^*\ ty^* \to U_0 \mid \mathfrak{ob} \hookrightarrow tm\} \\ &tm^*\ [\mathfrak{ob} \hookrightarrow A \mid P] = (t : tm\ A) \ltimes P\ t \end{aligned} \tag{5.23}$$

That is to say, in the model M^*, a term of the semantic type $[\mathfrak{ob} \hookrightarrow A \mid P] : ty^*$ is a term t of the syntactic type A that satisfies the meta-space predicate P.

5.4.2∗6 Notation. For every $A^* : el^*\ ty^*$, we define

$$\begin{aligned} &pre\ A^* : (\{\mathfrak{ob}\} \to tm\ A^*) \to \Omega^\bullet \\ &pre\ A^* = unglue\ A^* \end{aligned}$$

to remind us that ungluing a semantic type gives its underlying logical predicate. Similarly, for every $a^* : tm^*\ A^*$, we define

$$\begin{aligned} &prf\ a^* : pre\ A^*\ (\lambda\{\_ : \mathfrak{ob}\}.\ a^*) \\ &prf\ a^* = unglue\ a^* \end{aligned}$$

to remind us that ungluing a semantic term is the proof that the underlying syntactic term satisfies the corresponding logical predicate.

5.4.2∗7 Remark. For every $A^* : el^*\ ty^*$, the type $tm^*\ A^*$ satisfies the property that for every $a : \{\mathfrak{ob}\} \to A^*$, there is at most one element $a^* : tm^*\ A^*$ that restricts to a under $\mathfrak{ob}$, because the meta-space component of $tm^*\ A^*$ is a (fiberwise) meta-space proposition. Based on this observation, there is a more intrinsic alternative definition of ty^*: for every universe U of TTstc, we can define its *proof-irrelevant* subuniverse U^{ir} to be

$$U^{\mathrm{ir}} := \{A : U \mid \forall(a : \{\mathfrak{ob}\} \to A).\ (x, y : \{A \mid \mathfrak{ob} \hookrightarrow a\}) \to (x = y)\}.$$

Then we can define ty^* and tm^* as simply

$$\begin{aligned} &ty^* = [\mathfrak{ob} \hookrightarrow ty \mid (A : el\ ty) \ltimes \{U_0^{\mathrm{ir}} \mid \mathfrak{ob} \hookrightarrow tm\ A\}\ ] \\ &tm^*\ A^* = unglue\ A^* \end{aligned}$$

which directly mirrors the definition of ki^* (5.16) and el^* (5.17).

This alternative definition is in a suitable sense equivalent to the one above

(5.22, 5.23) because for every $A : \{\mathfrak{ob}\} \to U$, we have an equivalence

$$\{U_0^{\mathrm{ir}} \mid \mathfrak{ob} \hookrightarrow A\} \;\cong\; ((\{\mathfrak{ob}\} \to A) \to \Omega^\bullet).$$

when treating them as categories (in fact, preorders) suitably. We choose to work with ty^* (5.22) in terms of $\Omega^\bullet$-valued predicates because it is slightly more convenient for logical predicates on computation judgements later.

5.4.3 Base Types

5.4.3∗1. Since in the theory of $\mathrm{F}_\omega^{\mathrm{ha}}$, the unit type is specified to be isomorphic to meta-level unit type (5.3), we have no choice for the logical predicate for the logical predicate of the unit type (of $\mathrm{F}_\omega^{\mathrm{ha}}$) other than the always true predicate:

$$\begin{aligned} &\mathit{unit}^* : \{\mathit{el}^*\ \mathit{ty}^* \mid \mathfrak{ob} \hookrightarrow \mathit{unit}\} \\ &\mathit{unit}^* = [\mathfrak{ob} \hookrightarrow \mathit{unit} \mid \lambda(\_ : \{\mathfrak{ob}\} \to \mathit{tm}\ \mathit{unit}).\ 1] \end{aligned}$$

Recall that $\mathit{tm}^*\ \mathit{unit}^*$ computes to $(t : \mathit{tm}\ \mathit{unit}) \ltimes 1$, we define

$$\begin{aligned} &\mathit{unit\text{-}iso}^* : \mathit{tm}^*\ \mathit{unit}^* \cong 1 \\ &\mathit{unit\text{-}iso}^*.\mathit{fwd}\ \_ = * \\ &\mathit{unit\text{-}iso}^*.\mathit{bwd}\ \_ = [\mathfrak{ob} \hookrightarrow \mathit{unit\text{-}iso}.\mathit{bwd} \mid *] \end{aligned}$$

This is an isomorphism because $\mathit{tm}\ \mathit{unit} \cong 1$ by *unit-iso*.

5.4.3∗2. The other base type is the weak Boolean type *bool*. It is also the type that canonicity is about, so its logical predicate is specific to canonicity:

$$\begin{aligned} &\mathit{bool}^* : \{\mathit{el}^*\ \mathit{ty}^* \mid \mathfrak{ob} \hookrightarrow \mathit{bool}\} \\ &\mathit{bool}^* = [\mathfrak{ob} \hookrightarrow \mathit{bool} \mid P_{\mathit{can}}] \\ &P_{\mathit{can}} : (\{\mathfrak{ob}\} \to \mathit{tm}\ \mathit{bool}) \to \Omega^\bullet \\ &P_{\mathit{can}}\ b = \bullet(\{\mathfrak{ob}\} \to (b = \mathit{tt} \vee b = \mathit{ff})) \end{aligned} \tag{5.24}$$

The closed modality $\bullet$ is needed here to erase the object-space component of the proposition $\{\mathfrak{ob}\} \to (b = \mathit{tt} \vee b = \mathit{ff})$, turning it $\bullet$-modal. We also need to define the two terms of the weak Boolean types, i.e. showing that the two terms ff and tt satisfy the logical predicate of *bool*:

$$\begin{aligned} &\mathit{tt}^* : \{\mathit{tm}^*\ \mathit{bool}^* \mid \mathfrak{ob} \hookrightarrow \mathit{tt}\} \\ &\mathit{tt}^* = [\mathfrak{ob} \hookrightarrow \mathit{tt} \mid \eta^\bullet\ (\mathrm{inl}\ \mathit{refl})] \\ \\ &\mathit{ff}^* : \{\mathit{tm}^*\ \mathit{bool}^* \mid \mathfrak{ob} \hookrightarrow \mathit{ff}\} \\ &\mathit{ff}^* = [\mathfrak{ob} \hookrightarrow \mathit{ff} \mid \eta^\bullet\ (\mathrm{inr}\ \mathit{refl})] \end{aligned}$$

In the construction of M^*, the only things that are specific to canonicity are P_{can}, tt^* and ff^*. They can be changed to anything else without affecting other parts of M^* (although there seemingly are not many interesting choices of P_{can}).

5.4.4 Function Types

5.4.4∗1. A function $t : tm\ (A \Rightarrow_t B)$ is related by the logical predicate for the function type $A \Rightarrow_t B$ if it maps all input a satisfying the logical predicate for A to output $t\ a$ satisfying the logical predicate for B:

$$\begin{aligned}
&_\Rightarrow_t_^* : \{el^*\ ty^* \to el^*\ ty^* \to el^*\ ty^* \mid \mathfrak{ob} \hookrightarrow _\Rightarrow_t_\}\\
&[\mathfrak{ob} \hookrightarrow A \mid P] \Rightarrow_t^* [\mathfrak{ob} \hookrightarrow B \mid Q] = [\mathfrak{ob} \hookrightarrow A \Rightarrow_t B \mid P_{\Rightarrow_t}]
\end{aligned}$$

$$P_{\Rightarrow_t} := \lambda t.\ \forall(a : \{\mathfrak{ob}\} \to A).\ P\ a \to Q\ (\lambda\{\_ : \mathfrak{ob}\}.\ t\ a)$$

Note that in the expression $t\ a$, we have elided the isomorphism $\Rightarrow_t$-*iso* (5.3) between $tm\ (A \Rightarrow_t B)$ and $tm\ A \to tm\ B$.

We also need to define an isomorphism, for all $A^*, B^* : el^*\ ty^*$,

$$\Rightarrow_t\text{-}iso^* : tm^*\ (A^* \Rightarrow_t^* B^*) \cong (tm^*\ A^*) \to (tm^*\ B^*).$$

Letting $A^* = [\mathfrak{ob} \hookrightarrow A \mid P]$ and $B^* = [\mathfrak{ob} \hookrightarrow B \mid Q]$, we compute as follows:

$$\begin{aligned}
&tm^*\ ([\mathfrak{ob} \hookrightarrow A \mid P] \Rightarrow_t^* [\mathfrak{ob} \hookrightarrow B \mid Q])\\
&= (t : tm\ (A \Rightarrow_t B)) \ltimes (a : \{\mathfrak{ob}\} \to tm\ A) \to P\ a \to Q\ (t\ a)\\
&\cong \quad \{\text{by } \Rightarrow_t\text{-}iso\ (5.3)\}\\
&\quad (t : tm\ A \to tm\ B) \ltimes (a : \{\mathfrak{ob}\} \to tm\ A) \to P\ a \to Q\ (t\ a)\\
&\cong \quad \{\text{by } \ltimes\text{-}fun\text{-}iso \text{ from } 5.3.3{*}24\}\\
&\quad \big((a : tm\ A) \ltimes P\ a\big) \to \big((b : tm\ B) \ltimes Q\ b\big)\\
&= (tm^*\ [\mathfrak{ob} \hookrightarrow A \mid P]) \to (tm^*\ [\mathfrak{ob} \hookrightarrow B \mid Q])
\end{aligned}$$

5.4.4∗2. The logical predicate for polymorphic functions is

$$\begin{aligned}
&All^* : \{(k^* : ki^*) \to (el^*\ k^* \to el^*\ ty^*) \to el^*\ ty^* \mid \mathfrak{ob} \hookrightarrow All\}\\
&All^*\ k^*\ F = [\mathfrak{ob} \hookrightarrow All\ k^*\ F \mid \lambda t.\ \forall(\alpha^* : el^*\ k^*).\ pre\ (F\ \alpha^*)\ (\lambda\{_ : \mathfrak{ob}\}.\ (t\ \alpha^*))]
\end{aligned}$$

Let us check the type of this definition step-by-step. The object-space component $All\ k^*\ F$ is well typed because under $\mathfrak{ob}$, ki^* equals ki, so $k^* : ki$ under $\mathfrak{ob}$, and similarly $F : el\ k \to el\ ty$ under $\mathfrak{ob}$, thus $All\ k^*\ F : el\ ty$ as expected.

The meta-space component $\lambda t. \cdots$ should be an $\Omega^\bullet$-valued predicate on $t : \{\mathfrak{ob}\} \to tm\ (All\ k^*\ F)$. We have $F\ \alpha^* : el^*\ ty^*$; this type computes to

$$(A : el\ ty) \ltimes (\{\mathfrak{ob}\} \to\ tm\ A) \to \Omega^\bullet$$

by definitions (5.17, 5.22). Thus $pre\ (F\ \alpha^*)$ has type $(\{\mathfrak{ob}\} \to\ tm\ (F\ \alpha^*)) \to \Omega^\bullet$. On the other hand, t has type $\{\mathfrak{ob}\} \to tm\ (All\ k^*\ F)$, which is isomorphic to $\{\mathfrak{ob}\} \to (\alpha^* : el\ k^*) \to tm\ (F\ \alpha^*)$ via *All-iso* (5.3), which we elided in the definition above. The implicit function $\lambda\{\_ : \mathfrak{ob}\}.\ (t\ \alpha^*)$ then has type $\{\mathfrak{ob}\} \to tm\ (F\ \alpha^*)$, and therefore it can be supplied as an argument to $pre\ (F\ \alpha^*)$, yielding a proposition

in $\Omega^\bullet$. The quantification $\forall(\alpha^* : el^*\ k^*)$ is allowed because $\Omega^\bullet$ is closed under impredicative universal quantification.

We also need to define an isomorphism for $k^* : ki^*$ and $F : el^*\ k^* \to el^*\ ty^*$

$$All\text{-}iso^* : tm^*\ (All^*\ k^*\ F) \cong ((\alpha^* : el^*\ k^*) \to tm^*\ (F\ \alpha^*)).$$

This is very similar to $\Rightarrow_t\text{-}iso^*$ in 5.4.4∗1, and we give a direct definition here:

$$fwd\ [\mathfrak{ob} \hookrightarrow t \mid p] = \lambda\alpha^*.\ [\mathfrak{ob} \hookrightarrow p\ \alpha^* \mid All\text{-}iso.fwd\ t\ \alpha^*]$$

$$bwd\ h = [\mathfrak{ob} \hookrightarrow All\text{-}iso.bwd\ h \mid \lambda\alpha^*.\ prf\ (h\ \alpha^*)]$$

5.4.4∗3. By this point we have completed the definition of the logical predicates for the F_ω-fragment of F^{ha}_ω, so the derived concepts in Section 5.1.2 such as raw functors can be interpreted in M^* as well. For example, we have

$$\begin{aligned}
&tyco^* : ki^*\\
&tyco^* = ty^* \Rightarrow^*_k ty^*\\
&fmap\text{-}ty^* : (F : el^*\ tyco^*) \to el^*\ ty^*\\
&fmap\text{-}ty^*\ F = All^*\ ty^*\ (\lambda\alpha.\ All^*\ ty^*\ (\lambda\beta.\ (\alpha \Rightarrow^*_t \beta) \Rightarrow^*_t (F\ \alpha \Rightarrow^*_t F\ \beta)))\\
&\textsc{record}\ RawFunctor^* : \mathcal{U}_1\ \textsc{where}\\
&\quad F_0 : el^*\ tyco^*\\
&\quad F_1 : tm^*\ fmap\text{-}ty^*\ F_0
\end{aligned}$$

which is simply the same as the definition of *RawFunctor* in Section 5.1.2 except that all the judgements of F_ω such as *ki* and *ty* are replaced by their corresponding interpretation in M^*. Interpretations of other derived concepts in Section 5.1.2 such as $RawMonad^*$ and $RawHFunctor^*$ can be obtained in this way as well.

5.4.5 Computation Judgements

5.4.5∗1. What remains is the logical predicates for computation judgements of F^{ha}_ω. Recall that given $H : RawHFunctor$ and $A : el\ ty$, the judgements $co\ H\ A$ in F^{ha}_ω roughly axiomatise a monad equipped with H-operations – more precisely, $co\ H\ A$ axiomatises the Kleisli category of this monad since $co\ H\ A$ is not an F^{ha}_ω-type but a separate judgement.

5.4.5∗2 (First attempt). Since our logical predicates live in an impredicative universe, a natural idea is to define the logical predicate for *co* by an impredicative encoding of the initial monad equipped with H-operations:

$$\begin{aligned}
&co^* : \{HFunctor^* \to el^*\ ty^* \to \mathcal{U}_0 \mid \mathfrak{ob} \hookrightarrow co\}\\
&co^*\ H^*\ A^* = (c : co\ H^*\ A^*) \ltimes P_{co}\ c\\
&P_{co} : \{H^*, A^*\} \to (\{\mathfrak{ob}\} \to co\ H^*\ A^*) \to \Omega^\bullet\\
&P_{co}\ c = \forall(m : MonadAlg^*\ H^*).\ pre\ (m.M_0\ A^*)\ (eval\ m\ A^*\ c)
\end{aligned} \tag{5.25}$$

The function P_{co} type-checks as follows: the type of $m.M_0$ is $el^*\ tyco^*$, and

$$\begin{aligned}
& el^*\ tyco^* \\
= \quad & \{\text{by definition}\} \\
& el^*\ (ty^* \Rightarrow^*_k ty^*) \\
\cong \quad & \{\text{by the axiom (5.2)}\} \\
& el^*\ ty^* \to el^*\ ty^*
\end{aligned}$$

Therefore the type of $m.M_0\ A^*$ is $el^*\ ty^*$, which is the glue type

$$(A : el\ ty) \ltimes (\{\mathfrak{ob}\} \to\ tm\ A) \to \Omega^\bullet$$

by (5.17, 5.22). Then $pre\ (m.M_0\ A^*)$ has type $(\{\mathfrak{ob}\} \to\ tm\ (m.M_0\ A^*)) \to \Omega^\bullet$, i.e. it is a meta-space predicate on terms of type $\{\mathfrak{ob}\} \to m.M_0\ A^*$. The term $eval\ m\ A^*\ c$, which in fact is the implicit function $\lambda\{\_ : \mathfrak{ob}\}.\ eval\ m\ A^*\ c$, has precisely the type $\{\mathfrak{ob}\} \to\ tm\ (m.M_0\ A^*)$. Finally, since $\Omega^\bullet$ is closed under universal quantification $\forall(m : MonadAlg^*\ H^*)$, $P_{co}\ c$ has type $\Omega^\bullet$, i.e. $\{\Omega \mid \mathfrak{ob} \hookrightarrow 1\}$.

5.4.5∗3 Remark. Usually, the impredicative encoding of a datatype needs to be 'refined' by some additional equalities to have the correct universal property Awodey et al. [2018]. For example, the impredicative encoding of the coproduct type $A + B$ in an impredicative universe U is

$$\begin{aligned}
A + B &= \textstyle\sum(\alpha : (X : U) \to (A \to X) \to (B \to X) \to X)\ N\ \alpha \\
N\ \alpha &= (X, Y : U) \to (f : X \to Y) \to (h : A \to X) \to (k : B \to X) \\
&\qquad \to f\ (\alpha\ X\ h\ k) = \alpha\ Y\ (f \circ h)(f \circ k)
\end{aligned}$$

Without imposing N on α, the impredicative encoding would not satisfy the η-rule of the coproduct type. However, our logical predicates land in a universe *propositions*, where two elements of the same type are automatically equal, so this refinement is unnecessary.

5.4.5∗4. However, as we commented in 5.1.4∗2, evaluating the sequential composition $let\text{-}in\ c\ f$ is *not* compositional because of the discrepancy between computations *co* and raw monads: *co* satisfies the monadic laws while raw monads do not. For this reason, with the above definition of P_{co}, we will have problems with showing the term constructor *let-in* satisfies its logical predicate:

$$\begin{aligned}
let\text{-}in^* : \{\{H^*, A^*, B^*\} &\to co^*\ H^*\ A^* \to (tm^*\ A^* \to co^*\ H^*\ B^*) \\
&\to co^*\ H^*\ B^* \mid \mathfrak{ob} \hookrightarrow let\text{-}in\} \\
let\text{-}in^*\ c\ f = [\mathfrak{ob} \hookrightarrow let\text{-}in\ c\ f &\mid \lambda m.\ \texttt{?1}\ ]
\end{aligned}$$

where the hole ?1 has type $P_{co}\ (let\text{-}in\ c\ f)$, that is, by the definition of P_{co} above,

$$\forall(m : MonadAlg^*\ H^*).\ pre\ (m.M_0\ B^*)\ (eval\ m\ B^*\ (let\text{-}in\ c\ f)).$$

Since we do not have the equation *eval-let* in 5.1.4∗2 to simplify the computation $\mathit{eval}\ m\ B^*\ (\mathit{let\text{-}in}\ c\ f)$, we have no way to fill in the hole ?1 using c and f.

5.4.5∗5. To fix this problem, we strengthen $P_{co}\ c$ above to take into account all possible *continuations* after the computation c, which is essentially the idea of ⊤⊤-*lifting* [Katsumata 2005; Katsumata et al. 2018; Lindley and Stark 2005]. We first define a type of continuations accepting A^*-values:

$$\begin{aligned}
&\textsc{record}\ \mathit{Con}\ (H^* : \mathit{RawHFunctor}^*)\ (A^* : \mathit{el}^*\ \mathit{ty}^*) : U_1\ \textsc{where}\\
&\quad m^* : \mathit{MonadAlg}^*\ H^*\\
&\quad R^* : \mathit{el}^*\ \mathit{ty}^*\\
&\quad k : \{\mathfrak{ob}\} \to A^* \to \mathit{co}\ H^*\ R^*\\
&\quad k^* : \{\mathit{tm}^*\ A^* \to \mathit{tm}^*\ (m^*.M_0\ R^*) \mid \mathfrak{ob} \hookrightarrow \lambda a.\ \mathit{eval}\ m^*\ R^*\ (k\ a)\}
\end{aligned}$$

and the strengthened definition of P_{co} is

$$\begin{aligned}
&P_{co} : \{H^*, A^*\} \to (\{\mathfrak{ob}\} \to \mathit{co}\ H^*\ A^*) \to \Omega^\bullet\\
&P_{co}\ c = \forall(K : \mathit{Con}\ H^*\ A^*).\ \mathit{pre}\ (K.m^*.M_0\ \ K.R^*)\ (\lambda\{_ : \mathfrak{ob}\}. \qquad (5.26)\\
&\qquad\qquad \mathit{eval}\ K.m^*\ K.R^*\ (\mathit{let\text{-}in}\ c\ K.k))
\end{aligned}$$

Compared to the earlier version of P_{co} (5.25), the new version asserts that the computation c extended with an arbitrary 'good' continuation k and evaluated into a raw monad results in a value satisfying its logical predicate. Here a continuation k is 'good' if k followed by *eval* sends input satisfying its logical predicate to output satisfying its logical predicate, which is succinctly expressed by a function $k^* : \mathit{tm}^*\ A^* \to \mathit{tm}^*\ (m^*.M_0\ R^*)$, c.f. 5.3.3∗24.

5.4.5∗6 Remark. The new definition of P_{co} is similar to the model of computations in the realizability model (5.2∗9), except that here we only consider Kleisli morphisms $k^* : A^* \to m^*.M_0\ R^*$ whose object-space component factors through some $k : \{\mathfrak{ob}\} \to A^* \to \mathit{co}\ H^*\ R^*$. The author currently does not know if there could be a conceptual explanation of such a modified codensity transformation.

5.4.5∗7. The logical predicate for thunks is the same as that for computations, modulo the isomorphism $\mathit{th\text{-}iso} : \{H, A\} \to \mathit{tm}\ (\mathit{th}\ H\ A) \cong \mathit{co}\ H\ A$ from 5.1.3∗2:

$$\begin{aligned}
&\mathit{th}^* : \{\mathit{RawHFunctor}^* \to \mathit{el}^*\ \mathit{ty}^* \to \mathit{el}^*\ \mathit{ty}^* \mid \mathfrak{ob} \hookrightarrow \mathbb{T}\}\\
&\mathit{th}^*\ H^*\ A^* = [\mathfrak{ob} \hookrightarrow \mathbb{T}\ H^*\ A \mid \lambda t.\ P_{co}\ (\lambda\{_ : \mathfrak{ob}\}.\ \Uparrow t)]
\end{aligned}$$

where $\Uparrow$ is the forward direction of the isomorphism *th-iso*. The isomorphism $\mathit{th\text{-}iso}^* : \mathit{tm}^*\ (\mathbb{T}\ H^*\ A^*) \cong \mathit{co}^*\ H^*\ A^*$ is also straightforward:

$$\mathit{fwd}\ [\mathfrak{ob} \hookrightarrow t \mid p] = [\mathfrak{ob} \hookrightarrow \Uparrow t \mid p], \qquad \mathit{bwd}\ [\mathfrak{ob} \hookrightarrow c \mid p] = [\mathfrak{ob} \hookrightarrow \Downarrow c \mid p].$$

5.4.6 Computation Terms

5.4.6∗1. Finally, we need to prove that the constructors *val*, *let-in*, *op* and the eliminator *eval* of computations satisfy the logical predicates. We start with *val*:

$$\begin{aligned}
&val^* : \{\{H^*, A^*\} \to tm^*\ A^* \to co^*\ H^*\ A^* \mid \mathfrak{ob} \hookrightarrow val\} \\
&val^*\ \{H^*\}\ \{A^*\}\ a = [\mathfrak{ob} \hookrightarrow val\ a \mid \texttt{?1}\ ]
\end{aligned}$$

where the hole ?1 has type $P_{co}\ (val\ a)$, that is, by definition (5.26),

$$\begin{aligned}
&\forall(K : Con\ H^*\ A^*).\ pre\ (K.m^*.M_0\ \ K.R^*) \\
&\qquad (\lambda\{_ : \mathfrak{ob}\}.\ \ eval\ K.m^*\ K.R^*\ (let\text{-}in\ (val\ a)\ K.k)) \\
=\quad &\{\text{by axiom } let\text{-}val\ (5.4)\} \\
&\forall(K : Con\ H^*\ A^*).\ pre\ (K.m^*.M_0\ \ K.R^*) \\
&\qquad (\lambda\{_ : \mathfrak{ob}\}.\ \ eval\ K.m^*\ K.R^*\ (K.k\ a))
\end{aligned}$$

We put ?1 $= \lambda K.\ prf\ (K.k^*\ a)$, which is well typed because $K.k^*$ has type

$$k^* : \{tm^*\ A^* \to tm^*\ (m^*.M_0\ R^*) \mid \mathfrak{ob} \hookrightarrow \lambda a.\ eval\ m^*\ R^*\ (k\ a)\} \tag{5.27}$$

so $K.k^*\ a$ has type $tm^*\ (K.m^*.M_0\ K.R^*)$, and $prf\ (K.k^*\ a)$ has type

$$\begin{aligned}
&pre\ (K.m^*.M_0\ K.R^*)\ (\lambda\{_ : \mathfrak{ob}\}.\ (K.k^*\ a)) \\
=\quad &\{\text{by the restriction of } k^* \text{ under } \mathfrak{ob} \text{ in } (5.27)\} \\
&pre\ (K.m^*.M_0\ K.R^*)\ (\lambda\{_ : \mathfrak{ob}\}.\ \lambda a.\ eval\ K.m^*\ K.R^*\ (K.k\ a))
\end{aligned}$$

which is the desired type of ?1 .

5.4.6∗2. The case for *let-in* is similar:

$$\begin{aligned}
&let\text{-}in^* : \{\{H^*, A^*, B^*\} \to co^*\ H^*\ A^* \to (co^*\ H^*\ A^* \to co^*\ H^*\ B^*) \\
&\qquad\qquad \to co^*\ H^*\ B \mid \mathfrak{ob} \hookrightarrow let\text{-}in\} \\
&let\text{-}in^*\ c\ f = [\mathfrak{ob} \hookrightarrow let\text{-}in\ c\ f \mid \lambda(K : Con\ H^*\ A^*).\ unglue\ c\ K']
\end{aligned}$$

where each field of $K' : Con\ H^*\ B^*$ is defined as follows:

$$\begin{aligned}
K'.m^* &= K.m^* \\
K'.R^* &= K.R^* \\
K'.k &= \lambda\{_ : \mathfrak{ob}\}\ a.\ let\text{-}in\ (f\ a)\ K.k \\
K'.k^* &= \lambda a.\ [\mathfrak{ob} \hookrightarrow eval\ K'.m^*\ K'.R^*\ (let\text{-}in\ (f\ a)\ K.k) \mid unglue\ (f\ a)\ K]
\end{aligned}$$

The last line type checks because $f\ a : co^*\ H^*\ B^*$, so $unglue\ (f\ a) : P_{co}\ (f\ a)$, so by definition (5.26), the type of $unglue\ (f\ a)\ K$ is

$$pre\ (K.m^*.M_0\ \ K.R^*)\ (\lambda\{\_ : \mathfrak{ob}\}.\ eval\ K.m^*\ K.R^*\ (let\text{-}in\ (f\ a)\ K.k))$$

which is indeed the type of proofs that the syntactic component of $K'.k^*$ satisfies

the logical predicate of the type $k.m^*.M_0\ K.R^*$.

5.4.6∗3. The case for *op* is slightly more involved, so let us first show that *eval* satisfies the corresponding logical predicate:

$$\begin{aligned}&eval^* : \{\{H^*\} \to (m^* : MonadAlg^*\ H^*) \to (A^* : \_)\\ &\qquad\qquad \to co^*\ H^*\ A^* \to tm^*\ (m^*.M_0\ A^*) \mid \mathfrak{ob} \hookrightarrow eval\}\\ &eval^*\ m^*\ A^*\ c^* = [\mathfrak{ob} \hookrightarrow eval\ m^*\ A^*\ c \mid unglue\ c^*\ K]\end{aligned}$$

where the continuation $K : Con\ H^*\ A^*$ is defined by

$$\begin{aligned}K.m^* &= m^* & K.R^* &= A^*\\ K.k &= \lambda\{\_ : \mathfrak{ob}\}.\ val & K.k^* &= m^*.ret\end{aligned}$$

The definition of $K.k^*$ is well typed because the expected type of $K.k^*$ is

$$\begin{aligned}&\{tm^*\ A^* \to tm^*\ (m^*.M_0\ R^*) \mid \mathfrak{ob} \hookrightarrow \lambda a.\ eval\ m^*\ R^*\ (k\ a)\}\\ =\quad &\{\text{by the definition of } K.k \text{ above}\}\\ &\{tm^*\ A^* \to tm^*\ (m^*.M_0\ A^*) \mid \mathfrak{ob} \hookrightarrow \lambda a.\ eval\ m^*\ R^*\ (val\ a)\}\\ =\quad &\{\text{by axiom } \textit{eval-val}\ (5.6)\}\\ &\{tm^*\ A^* \to tm^*\ (m^*.M_0\ A^*) \mid \mathfrak{ob} \hookrightarrow \lambda a.\ m^*.ret\ R^*\ a\}\end{aligned}$$

5.4.6∗4. Coming back to *op*, we start with some obvious steps and a hole:

$$\begin{aligned}&op^* : \{\{H^*, A^*, B^*\} \to tm^*\ (H^*.H_0\ (\mathbb{T}^*\ H^*)\ A^*)\\ &\qquad\qquad \to (tm^*\ A^* \to co^*\ H^*\ B^*) \to co^*\ H^*\ B^* \mid \mathfrak{ob} \hookrightarrow op\}\\ &op^*\ o\ k = [\mathfrak{ob} \hookrightarrow op\ o\ k \mid \lambda(K : Con\ H^*\ B^*).\ \texttt{?1}\ ]\end{aligned}$$

where the hole ?1 has type

$$\begin{aligned}&pre\ (K.m^*.M_0\ \ K.R^*)\ (\lambda\{\_ : \mathfrak{ob}\}.\ eval\ K.m^*\ K.R^*\ (let\text{-}in\ (op\ o\ k)\ K.k))\\ =\quad &\{\text{by axiom } \textit{let-op}\ (5.5)\}\\ &pre\ (K.m^*.M_0\ \ K.R^*)\ (\lambda\{\_ : \mathfrak{ob}\}.\ eval\ K.m^*\ K.R^*\ (op\ o\ (\lambda a.\ let\text{-}in\ (k\ a)\ K.k)))\\ =\quad &\{\text{by axiom } \textit{eval-op}\ (5.7)\}\\ &pre\ (K.m^*.M_0\ \ K.R^*)\ (\lambda\{\_ : \mathfrak{ob}\}.\ K.m^*.bind\ o'\ (\lambda a.\ eval\ \_\ \_\ (let\text{-}in\ (k\ a)\ K.k)))\end{aligned}$$

where $o' : tm^*\ (K.m^*.M_0\ A^*)$ is the result of evaluating the operand o inside the higher-order functor H and then applying the operation on the monad $K.m^*$:

$$o' := K.m^*.malg\ \_\ (H^*.hmap\ \_\ \_\ e\ \_\ o),$$

and $e : tm^*\ (trans^*\ (M^*.\mathbb{T}\ H^*)\ K.m^*.M_0)$ is $eval^*$ specialised to $K.m^*$:

$$e\ A^*\ c = eval^*\ K.m^*\ A^*\ (\Uparrow c).$$

Now coming back to the hole ?1 , using k we can define

$$f : (a : tm^*\ A^*) \to tm^*\ (K.m^*.M_0\ \ K.R^*)$$
$$f = [\mathfrak{ob} \hookrightarrow eval\ \_\ \_\ (let\text{-}in\ (k\ a)\ K.k) \mid unglue\ (k\ a)\ K]$$

and finally we can put ?1 $= prf\ (K.m^*.bind\ o'\ f)$.

5.4.6∗5. The last bit of our construction of the glued model M^* is showing that it satisfies the equational axioms of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ pertaining to computations, but this is easy because our interpretation of computations and terms in M^* is proof *irrelevant*. For every universe U of TTstc, there is a subuniverse

$$U^{\mathrm{ir}} = \{A : U \mid \forall(a : \{\mathfrak{ob}\} \to A).\ (x, y : \{A \mid \mathfrak{ob} \hookrightarrow a\}) \to x = y\}$$

which classifies *proof-irrelevant* logical predicates in U, in the sense that partial elements $a : \{\mathfrak{ob}\} \to A$ of a type $A : U^{\mathrm{ir}}$ have unique total extensions (if exist).

5.4.6∗6 Lemma. For all $A^* : el^*\ ty^*$ and $H^* : RawHFunctor^*$, the types

$$tm^*\ A^* : U_0 \quad \text{and} \quad co^*\ H^*\ A^* : U_0$$

are classified by the subuniverse U_0^{ir}.

Proof. Let A^* be $[\mathfrak{ob} \hookrightarrow A \mid P]$ where $A : \{\mathfrak{ob}\} \to el\ ty$ is an object-space type and $P : (\{\mathfrak{ob}\} \to tm\ A) \to \Omega^\bullet$ is a meta-space predicate. By definition (5.23), $tm^*\ A^*$ is the glue type $(a : tm\ A) \ltimes P\ a$. Thus a partial element a of $tm^*\ A^*$ is exactly an element $a : \{\mathfrak{ob}\} \to tm\ A$. Given two elements $x, y : \{tm^*\ A^* \mid \mathfrak{ob} \hookrightarrow a\}$, $unglue\ x = unglue\ y$ since they are elements of the propositional type $P\ a$, so $x = [\mathfrak{ob} \hookrightarrow a \mid unglue\ x] = [\mathfrak{ob} \hookrightarrow a \mid unglue\ y] = y$. The case for co^* is similar. □

5.4.6∗7 Corollary. The glued model M^* satisfies the equational axioms *val-let*, *let-val*, *let-assoc* (5.4), *let-op* (5.5), *eval-val* (5.6), *eval-op* (5.7) of $\mathrm{F}^{\mathrm{ha}}_{\omega}$.

Proof. Taking *val-let* for example, we need to show

$$val\text{-}let^* : \{H^*, A^*, B^*\} \to (a : tm^*\ A^*) \to (k : tm^*\ A \to co^*\ H^*\ B^*)$$
$$\to let\text{-}in^*\ (val^*\ H^*\ a)\ k = k\ a$$

Since the type $co^*\ H^*\ B^*$ is in the universe U_0^{ir}, it is sufficient to show that

let-in* (*val** H^* a) k and k a are equal under $\mathfrak{ob}$,

$$\begin{aligned}
&\textit{let-in}^* \ (\textit{val}^* \ H^* \ a) \ k \\
=\ &\{\text{under } \mathfrak{ob},\ \textit{let-in}^* = \textit{let-in} \text{ and } \textit{val}^* = \textit{val}\} \\
&\textit{let-in} \ (\textit{val} \ H^* \ a) \ k \\
=\ &\{M \text{ satisfies } \textit{val-let}\} \\
&k \ a
\end{aligned}$$

The case for other equational axioms are similar. □

5.4.6∗8. We have completed the construction of M^*, which may be called the *synthetic fundamental lemma* of logical predicates for System $\mathrm{F}^{\mathrm{ha}}_{\omega}$.

5.4.6∗9 Lemma (Fundamental). In the language TTstc for System $\mathrm{F}^{\mathrm{ha}}_{\omega}$, given any $P : (\{\mathfrak{ob}\} \to M.tm \ M.bool) \to \Omega^{\bullet}$ with $t : P \ M.tt$ and $f : P \ M.ff$, there is an

$$M^* : \{[\![\mathrm{F}^{\mathrm{ha}}_{\omega}]\!]_{U_2} \mid \mathfrak{ob} \hookrightarrow M\}$$

such that $M^*.tm \ M^*.bool = (b : M.tm \ M.bool) \ltimes P \ b$.

5.4.7 The External Canonicity Result

5.4.7∗1. We have defined a glued model M^* internal to the language TTstc, the last part of our canonicity proof is to externalise the model M^* in the ambient meta theory, obtaining what we want to prove.

5.4.7∗2. As we remarked in 5.3.3∗25, the type theory TTstc can be interpreted in the glued topos $\mathcal{G}$ of $\mathrm{Pr}\,(\mathrm{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega})$ and Set along the global section functor Γ. Thus M^* gives rise to a diagrammatic model of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ in $\mathcal{G}$, which further induces an LCC-functor $\overline{M^*} : \mathrm{Jdg} \to \mathcal{G}$ by Theorem 4.4∗10. On the other hand, the model M corresponds to the Yoneda embedding $Y : \mathrm{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega} \to \mathrm{Pr}\,\mathrm{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega} \cong G/\mathfrak{ob}$. The fact that M^* restricts to M under $\mathfrak{ob}$ in the language TTstc means externally that the following diagram commutes up to some unique natural isomorphism:

$$\begin{array}{ccc}
\mathrm{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega} & \xrightarrow{\overline{M^*}} & \mathcal{G} \\
& \overset{\cong}{\searrow^{Y}} & \downarrow{\scriptstyle j^*} \\
& & \mathrm{Pr}\,(\mathrm{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}) \ \cong \ \mathcal{G}/\mathfrak{ob}
\end{array} \tag{5.28}$$

where j^* is the inverse image of the geometric morphism $j : \mathrm{Pr}\,(\mathrm{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}) \to \mathcal{G}$:

$$j^*\langle A, S, f : S \to \Gamma A\rangle = A \quad \text{and} \quad j_*A = \langle A, \Gamma A, id : \Gamma A \to \Gamma A\rangle.$$

5.4.7∗3 Theorem (Canonicity). *For every closed Boolean term b of System $\mathrm{F}^{\mathrm{ha}}_{\omega}$, i.e. a morphism $b : 1 \to tm \ bool$ in the category $\mathrm{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}$, either $b = tt$ or $b = ff$ holds* (*but*

not both) *in the equational theory of* $\mathrm{F}_\omega^{\mathrm{ha}}$.

Proof. Instantiating Lemma 5.4.6∗9 with P_{can} as in (5.24), let $B := j^*(\overline{M^*}(tm\ bool))$ and let $\phi_{bool} : B \cong Y(tm\ bool)$ be the component of the natural isomorphism (5.28) at $tm\ bool \in \textsc{Jdg}\,\mathrm{F}_\omega^{\mathrm{ha}}$. The functor $\overline{M^*}$ maps the object $tm\ bool$ to an object $B^* \in \mathscr{G}$:

$$B^* := \langle B,\ \{t : 1 \to B \mid (\phi_{bool} \cdot t = \mathrm{Y}\ tt) \vee (\phi_{bool} \cdot t = \mathrm{Y}\ ff)\},\ j \rangle$$

where j is the inclusion function into $\Gamma B := \{t : 1 \to B\}$. The morphism $b : 1 \to tm\ bool$ is then sent by $\overline{M^*}$ to a morphism $1 \to B^*$ in $\mathscr{G}$:

$$\begin{array}{ccc} \{*\} & \xrightarrow{\ p\ } & \{t : 1 \to B \mid (\phi_{bool} \cdot t = \mathrm{Y}\ tt) \vee (\phi_{bool} \cdot t = \mathrm{Y}\ ff)\} \\ \downarrow{\scriptstyle !} & & \downarrow{\scriptstyle j} \\ \{*\} & \xrightarrow[\lambda *.\ \phi_{bool}^{-1} \cdot \mathrm{Y}b]{} & \{t : 1 \to B\} \end{array}$$

The commutativity of this diagram entails $\mathrm{Y}b = \mathrm{Y}tt$ or $\mathrm{Y}b = \mathrm{Y}ff$, so $b = tt$ or $b = ff$ since Yoneda embedding is faithful. Moreover, $b = tt$ and $b = ff$ cannot be true at the same time because tt and ff have different interpretations in the realizability model in Section 5.2. □

5.4.7∗4 Corollary. The realizability in Section 5.2 is *adequate* in the sense that for for every closed Boolean term b, if its realizability interpretation is true (resp. false), then b equals tt (resp. ff) in $\mathrm{F}_\omega^{\mathrm{ha}}$. This is because b is either equal to tt or ff in $\mathrm{F}_\omega^{\mathrm{ha}}$ by canonicity, and if its realizability interpretation is true, then b must be equal to tt (otherwise its realizability interpretation would be false).

5.5 Parametricity and Free Theorems

5.5∗1. An appealing aspect of the synthetic fundamental lemma (5.4.6∗9) is that it is proved solely in the language TTstc, thus applicable to any category $\mathscr{G}$ that models TTstc. As an instance, we can deduce the *abstraction theorem* [Reynolds 1983], also known as *parametricity* [Wadler 1989], for System $\mathrm{F}_\omega^{\mathrm{ha}}$.

5.5∗2. Let $M : \textsc{Jdg}\,\mathrm{F}_\omega^{\mathrm{ha}} \to \mathscr{C}$ be any model of $\mathrm{F}_\omega^{\mathrm{ha}}$ in a small LCCC $\mathscr{C}$. As commented in 5.3.3∗25, we can interpret TTstc in the Artin gluing $\mathscr{G}_{\mathscr{C}}$ of $\textsc{Pr}\,\mathscr{C}$ and $\textsc{Set}$ along the global section functor $\textsc{Hom}_{\textsc{Pr}\,\mathscr{C}}(1, -) : \textsc{Pr}\,\mathscr{C} \to \textsc{Set}$, with the object-space model M of TTstc interpreted as the given functor $M : \textsc{Jdg}\,\mathrm{F}_\omega^{\mathrm{ha}} \to \mathscr{C}$ composed with Yoneda embedding $\mathrm{Y} : \mathscr{C} \to \textsc{Pr}\,\mathscr{C}$.

We have a functor $\overline{M^*} : \textsc{Jdg}\,\mathrm{F}_\omega^{\mathrm{ha}} \to \mathscr{G}_{\mathscr{C}}$ by instantiating the fundamental lemma (5.4.6∗9) with P_{can} as in (5.24). For every $A : 1 \to el\ ty \in \textsc{Jdg}\,\mathrm{F}_\omega^{\mathrm{ha}}$, we let P_A be $\overline{M^*}\ (tm\ A) \in \mathscr{G}$ (5.28) viewed as a predicate (in the ambient meta-theory) on

the set $\mathscr{C}(1, M\ (tm\ A))$. Similarly, for every $K : 1 \to ki \in \textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}$, we let P_K be $\overline{M^*}\ (el\ K) \in \mathscr{G}$ viewed as a family of sets indexed by the set $\mathscr{C}(1, M\ (el\ K))$.

5.5∗3 Theorem (Unary Parametricity). *Let M and P be as in* 5.5∗2. *For every $A : 1 \to el\ ty$ and $t : 1 \to tm\ A$ in $\textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}$, $P_A(M\ t)$ holds. Moreover, for every $K : 1 \to ki$ and $t : 1 \to el\ K$, there is an element $t^* \in P_K(M\ t)$.*

Proof. Given $t : 1 \to tm\ A \in \textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}$, it is mapped by the logical predicate model $\overline{M^*}$ to a morphism $1 \to M^*(tm\ A)$ in $\mathscr{G}_{\mathscr{C}}$, which amounts to a commutative square:

$$\begin{array}{ccc} \{*\} & \xrightarrow{t^*} & \{t : 1 \to M(tm\ A) \mid P_A(t)\} \\ \downarrow{\scriptstyle !} & & \downarrow{\scriptstyle \subseteq} \\ \{*\} & \xrightarrow[\lambda *.\ Mt]{} & \textsc{Hom}_{\textsc{Pr}\,\mathscr{C}}(1, M(tm\ A)) \end{array}$$

The commutativity of the square means that Mt satisfies P_A.

The statement for $t : 1 \to el\ K$ is essentially the same, with the element $t^* \in P_K(t)$ given by the top arrow of the diagram. □

5.5∗4 Example. Parametricity are useful for deriving 'free theorems' of programming languages [Wadler 1989]. As a 'hello world'-application, we can use parametricity to deduce that for every closed $\mathrm{F}^{\mathrm{ha}}_{\omega}$ term $t : tm\ (All\ ty\ (\lambda\alpha.\ \alpha \Rightarrow_t \alpha))$, t applied to every closed type A and closed term $a : tm\ A$ is equal a.

First of all, internal to TTstc, we prove the following statement:

$$\begin{aligned} &lem : (t^* : tm^*\ (All^*\ ty^*\ (\lambda\alpha.\ \ \alpha \Rightarrow^*_t \alpha))) \\ &\quad \to (A : \{\mathfrak{ob}\} \to el\ ty) \to (a : \{\mathfrak{ob}\} \to tm\ A) \\ &\quad \to \bullet(\{\mathfrak{ob}\} \to t^*\ A\ a = a) \\ &lem\ t^*\ A\ a\ \ = \ ?0 \end{aligned}$$

Recall that $prf\ t^*$ is the proof that the object-space component of t^* satisfies its logical predicate. Expanding definitions (5.23, 5.4.2∗6, 5.4.4∗2), we have

$$prf\ t^* : \forall(\alpha^* : el^*\ ty^*).\ pre\ (\alpha^* \Rightarrow^*_t \alpha^*)\ (\lambda\{\_ : \mathfrak{ob}\}.\ (t^*\ \alpha^*)).$$

To use $prf\ t^*$, we define a predicate $A^* : \{el^*\ ty^* \mid \mathfrak{ob} \hookrightarrow A\}$ by

$$A^* := [\mathfrak{ob} \hookrightarrow A \mid \lambda x.\ \bullet(\{\mathfrak{ob}\} \to x = a)]$$

for which only the element $a : \mathfrak{ob} \to tm\ A$ is satisfied. Now we have

$$prf\ t^*\ A^* : pre\ (A^* \Rightarrow^*_t A^*)\ (\lambda\{\_ : \mathfrak{ob}\}.\ (t^*\ A)).$$

Expanding the definition of $\Rightarrow^*_t$ from 5.4.4∗1, we have

$$prf\ t^*\ A^* : \forall(x : \{\mathfrak{ob}\} \to A).\ \bullet(\{\mathfrak{ob}\} \to x = a) \to \bullet(\{\mathfrak{ob}\} \to t^*\ A\ x = a).$$

The element a is always equal to itself, so we can complete the hole:

$$?0 = prf\ t^*\ A^*\ a\ (\eta^\bullet\ refl_a).$$

Now we interpret *lem* in the glued topos $\mathscr{G}$ as in 5.5∗2 with M being the identity functor. Evaluating the interpretation of *lem* at t, A, and a, we get a global section of the interpretation of $\bullet(t\ A\ a = a)$, which implies $t\ A\ a$ and a are equal morphisms $1 \to tm\ A$ in $\textsc{Jdg}\,\mathrm{F}_\omega^{\mathrm{ha}}$.

5.5∗5. It is also possible to extend the unary parametricity result above to the binary (or n-ary) case. Following Sterling and Harper [2021], given two models $M_L : \textsc{Jdg}\,\mathrm{F}_\omega^{\mathrm{ha}} \to \mathscr{C}$ and $M_R : \textsc{Jdg}\,\mathrm{F}_\omega^{\mathrm{ha}} \to \mathscr{D}$, we consider the Artin gluing $\mathscr{G}_{\mathscr{C}\mathscr{D}}$ of the product category $\textsc{Pr}\,\mathscr{C} \times \textsc{Pr}\,\mathscr{D}$ and the category of sets along the functor

$$\langle A, B\rangle \mapsto \mathscr{C}(1, A) \times \mathscr{D}(1, B).$$

The category $\mathscr{G}_{\mathscr{C}\mathscr{D}}$ is equivalent to the presheaf topos over $(\mathscr{C} + \mathscr{D})_\top$, and every object in the category $\mathscr{G}_{\mathscr{C}\mathscr{D}}$ is a tuple

$$\langle A \in \textsc{Pr}\,\mathscr{C},\ B \in \textsc{Pr}\,\mathscr{D},\ P \in \textsc{Set},\ l : P \to \textsc{Hom}(1, A),\ r : P \to \textsc{Hom}(1, B)\rangle,$$

i.e. a proof-relevant binary relation (also known as a span) over global elements of the presheaves A and B. The category $\mathscr{G}_{\mathscr{C}\mathscr{D}}$ has two subterminal objects

$$\mathfrak{ob}_L := \langle 1_{\textsc{Pr}\,\mathscr{C}}, 0, \emptyset, !, !\rangle \qquad \text{and} \qquad \mathfrak{ob}_R := \langle 0, 1_{\textsc{Pr}\,\mathscr{D}}, \emptyset, !, !\rangle,$$

which determine two open subtoposes that are equivalent to $\textsc{Pr}\,\mathscr{C}$ and $\textsc{Pr}\,\mathscr{D}$ respectively. The disjunction of $\mathfrak{ob}_L$ and $\mathfrak{ob}_R$ is another subterminal object

$$\mathfrak{ob} := \langle 1_{\textsc{Pr}\,\mathscr{C}}, 1_{\textsc{Pr}\,\mathscr{D}}, \emptyset, !, !\rangle,$$

whose corresponding open subtopos is equivalent to $\textsc{Pr}\,(\mathscr{C} + \mathscr{D})$.

5.5∗6. The type theory TTstc can be interpreted in $\mathscr{G}_{\mathscr{C}\mathscr{D}}$ as usual, with $\mathfrak{ob} : \Omega$ interpreted as the subterminal object $\mathfrak{ob}$ above. Moreover, we can extend TTstc with the following new constants with the evident interpretation in $\mathscr{G}_{\mathscr{C}\mathscr{D}}$:

$$\mathfrak{ob}_L : \Omega \qquad \mathfrak{ob}_R : \Omega \qquad \_ : \mathfrak{ob}_L \vee \mathfrak{ob}_R = \mathfrak{ob} \qquad \_ : \mathfrak{ob}_L \wedge \mathfrak{ob}_R = 0$$
$$M_L : \{\mathfrak{ob}_L\} \to [\![\mathrm{F}_\omega^{\mathrm{ha}}]\!]_{U_0} \qquad M_R : \{\mathfrak{ob}_R\} \to [\![\mathrm{F}_\omega^{\mathrm{ha}}]\!]_{U_0}$$
$$\_ : M = \lambda\{z : \mathfrak{ob}\}.\ \textsc{case}\ z\ \textsc{of}\ \{\mathrm{inl}\ (\_ : \mathfrak{ob}_L) \mapsto M_L;\ \mathrm{inr}\ (\_ : \mathfrak{ob}_R) \mapsto M_R\}$$

We refer to the extended language by 2-TTstc.

5.5∗7. The synthetic fundamental lemma (5.4.6∗9) holds in 2-TTstc without needing any modification, since 2-TTstc only adds new axioms to TTstc. However, in 2-TTstc an $\mathfrak{ob}$-partial element $\{\mathfrak{ob}\} \to A$ of some type A is now equal to an element of type $\{\mathfrak{ob}_L \vee \mathfrak{ob}_R\} \to A$, which are equivalently two partial elements

$\{\mathfrak{ob}_L\} \to A$ and $\{\mathfrak{ob}_R\} \to A$. Therefore the unary logical predicates in the proof of Lemma 5.4.6∗9 can be now read as binary logical relations.

Specially, we can set both M_L and M_R to be $\mathrm{Id} : \textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega} \to \textsc{Jdg}\,\mathrm{F}^{\mathrm{ha}}_{\omega}$, and we obtain the binary version of parametricity of closed $\mathrm{F}^{\mathrm{ha}}_{\omega}$-terms (Theorem 5.5∗3) by instantiating the fundamental lemma with the logical relation P for *bool* to be equality (this relation cannot be internally defined in 2-TTstc though, since this relation only makes sense when $M_L = M_R$).

5.6 General Recursion

5.6∗1. Handling (higher-order) algebraic effects is a form of *structural recursion*, where recursive calls are always made on structurally smaller input, so the recursion is guaranteed to terminate. This is in contrast with *general recursion*, which allows unrestricted recursive calls and thus results in possibly non-terminating programs. General recursion is allowed in most 'practical' programming languages nowadays. Without getting into the discussion of whether we *should or not* introduce general recursion into our language [McBride 2015; Turner 2004], in this section we show how it can be done if we want to.

5.6.1 The Signature of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ with Recursion

5.6.1∗1. We will refer to the extension of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ with general recursion as $\mathrm{rF}^{\mathrm{ha}}_{\omega}$. The signature of $\mathrm{rF}^{\mathrm{ha}}_{\omega}$ extends that of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ with a new family of judgements *pco* for *partial computations* that has the same signature as *co* (5.1.3∗1):

$$pco : (H : RawHFunctor) \to (A : el\ ty) \to \mathbb{J}.$$

The original computation judgement *co* is still kept in the language and is used for total computations as usual. Most accompanying rules for *co* in Section 5.1.3 are inherited by *pco*: *val*, *let-in*, *th*, *op*, and all their associated equations. We shall refer to the copy of them for *pco* by same names as before, except for thunks of partial computations, which we call $pth : RawHFunctor \to el\ ty \to el\ ty$.

5.6.1∗2. The first new rule for *pco* is as expected a fixed-point combinator:

$$Y : \{H, A\} \to (pth\ H\ A \to pco\ H\ A) \to pco\ H\ A,$$

together with the equation:

$$Y\text{-}eq : \{H, A, f\} \to Y\ f = f\ (\Downarrow Y\ f).$$

5.6.1∗3. Another difference between *pco* and *co* is the their elimination rule: *eval* from 5.1.3∗4 allows computations *co H A* to be evaluated into any raw monad T equipped with an H-operation, but naturally, *pco* shall only be evaluated into

monads T that 'support recursion'. In the current call-by-value setting, the only thing that supports recursion is *pco*, so we will require that the raw monad T send every type $A : \mathit{el\ ty}$ to thunks of partial computations $\mathit{pth}\ H\ (F\ A)$ for some $H : \mathit{RawHFunctor}$ and type constructor $F : \mathit{el\ ty} \to \mathit{el\ ty}$:

$$
\begin{array}{l}
\textsc{record}\ \mathit{MonadAlgRec}\ (H : \mathit{RawHFunctor}) : \mathbb{J}\ \textsc{where} \\
\quad \textsc{include}\ \mathit{MonadAlg}\ H\ \textsc{as}\ T \\
\quad H : \mathit{RawHFunctor} \\
\quad F\ \ : \mathit{el\ ty} \to \mathit{el\ ty} \\
\quad \mathit{eq} : (A : \mathit{el\ ty}) \to T\ A = \mathit{pth}\ H\ (F\ A)
\end{array}
$$

Here we have formulated the requirement *eq* using the equality type of LccLF, and in an implementation of the type checker for $\mathrm{rF}_\omega^{\mathrm{ha}}$, the equation *eq* may be mechanically checked since the kind language of $\mathrm{rF}_\omega^{\mathrm{ha}}$ is normalising.

The language $\mathrm{rF}_\omega^{\mathrm{ha}}$ then has the following declaration:

$$\mathit{eval} : \{H\} \to (T : \mathit{MonadAlgRec}\ H) \to (A : \mathit{el\ ty}) \to \mathit{pco}\ H\ A \to \mathit{tm}\ (T\ A)$$

together with equations *eval-val* and *eval-op* similar to the those of *co* (5.6, 5.7) with *co* replaced by *pco*, *th* replaced by *pth*, and *MonadAlg* replaced by *MonadAlgRec*.

5.6.1∗4. We will further include in $\mathrm{rF}_\omega^{\mathrm{ha}}$ the *empty type*:

$$
\begin{array}{ll}
\mathit{empty} : \mathit{el\ ty} & \\
\mathit{absurd} & : (A : \mathit{el\ ty}) \to \mathit{tm\ empty} \to \mathit{tm}\ A \\
\mathit{absurd\text{-}uniq} & : \{A : \mathit{el\ ty}\} \to (f : \mathit{tm\ empty} \to \mathit{tm}\ A) \to f = \mathit{absurd}\ A
\end{array}
$$

This gives us a judgement *pco VoidH* for *effect-free partial computations*, where *VoidH* is the constant raw higher-order functor: $\mathit{VoidH}\ \_\ \_ = \mathit{empty}$.

5.6.2 Synthetic Domain Theory in Assemblies

5.6.2∗1. Simply typed λ-calculi with general recursion famously can be modelled by variations of *complete partial orders* from classical domain theory [Plotkin 1977; Scott 1993; Streicher 2006]. However, the language $\mathrm{rF}_\omega^{\mathrm{ha}}$ has impredicative polymorphism, which is very tricky to model using classical domain theory, although not impossible [Coquand et al. 1989; Crole 1994].

5.6.2∗2. Alongside a few other reasons, the difficulty of modelling polymorphism in classical domain theory motivated the development of *synthetic domain theory* (SDT) [Hyland 1991; Phoa 1991; Rosolini 1986]. The idea of SDT is to axiomatise 'domains', in the general sense of objects that provide meaning to (recursive) programs, as special 'sets' satisfying certain properties in the logic of toposes or constructive set theory, so that every function between those special sets is

automatically a 'continuous map' between domains. In this way, one can give denotational semantics to recursive programs in a naive set-theoretic way.

5.6.2∗3. The exact axiomatisation of SDT varies across authors, but there are mainly two kinds of models: realizability toposes [Longley and Simpson 1997; Phoa 1991] and Grothendieck toposes [Fiore and Plotkin 1997; Fiore and Rosolini 1997]. Since we are already modelling $\mathrm{F}_\omega^{\mathrm{ha}}$ using a realizability model in Section 5.2, we will stick with the realizability model, following the ideas of SDT concretely in this model (as opposed to using SDT as an axiomatic language).

5.6.2∗4. The rest of this section is a short introduction to SDT based on Longley and Simpson's [1997] approach using *well complete objects*, adapted to a type-theoretic language. See also Longley's [1995] thesis, the more general treatment by Simpson [2004, 1999], and the type-theoretic formalisation of SDT using *well complete* Σ*-spaces* by Reus [1996, 1999] and Reus and Streicher [1999]. With the machinery of SDT in this section, the interpretation of $\mathrm{rF}_\omega^{\mathrm{ha}}$ will be almost trivial and will be presented in the next section.

5.6.2∗5. Before going into SDT, let us quickly recall a typical setup of interpreting recursion in classical domain theory, which we are going to mirror in the SDT.

A *predomain* (or precisely, an ω-cpo, in this setup) is a partially ordered set $\langle A, \sqsubseteq\rangle$ that has suprema $\sqcup_i a_i$ for all ω-chains $a_0 \sqsubseteq a_1 \sqsubseteq a_2 \sqsubseteq a_3 \sqsubseteq \cdots$ in A; a predomain need not have a bottom element. Morphisms between predomains are monotone functions preserving those suprema of ω-chains.

A *(Scott-) open set* of a predomain A is a subset $O \subseteq A$ that is (1) upward closed: for all $x, y \in A$, if $x \in O$ and $x \sqsubseteq y$ then $y \in O$, and (2) continuous: for all ω-chains a_i in A, $\sqcup_i a_i \in O$ iff there exists some n such that $a_n \in O$. Open sets of a predomain A are in bijection with morphisms $A \to \mathfrak{S}$, where $\mathfrak{S}$ is the two-element predomain $\{\bot \sqsubseteq \top\}$, sometimes called the *Sierpiński space* ($\mathfrak{S}$ is the Fraktur letter for S). Namely, every open set $O \subseteq A$ corresponds to the morphism $\chi : A \to \mathfrak{S}$ where $\chi(a) = \top$ if $a \in O$ and $\chi(a) = \bot$ if $a \notin O$.

The *lifting monad* LA on predomains adjoins a new *bottom element* $\bot$ to A, with a monad structure similar to that of the monad $1 + -$ on sets. Kleisli morphisms of predomains $f : \Gamma \to LA$ are in bijection with *partial* morphisms $\langle O, \bar{f}\rangle : \Gamma \rightharpoonup A$, each consisting of an open set $O \subseteq \Gamma$ and a (total) morphism $\bar{f} : O \to A$.

A *domain* D is a predomain with bottom element $\bot_D$, which is the same as an Eilenberg-Moore algebra of the lifting monad L. Every endo-morphism $f : D \to D$ on domains then has a least fixed point by taking the supremum of the chain $\bot_D \sqsubseteq f(\bot_D) \sqsubseteq f(f(\bot_D)) \sqsubseteq \cdots$ in D.

Contexts Γ and types σ of a call-by-value programming language with recursion are then interpreted as predomains $[\![\Gamma]\!]$, $[\![\sigma]\!]$. Terms $\Gamma \vdash t : \sigma$ are interpreted as morphisms $[\![\Gamma]\!] \to L[\![\sigma]\!]$, i.e. partial morphisms between predomains.

5.6.2∗6. Recall that the internal language of assemblies Asm($\mathbb{A}$) over a partial combinator algebra $\mathbb{A}$, which we used to construct a model of $\mathrm{F}_\omega^{\mathrm{ha}}$ in Section 5.2, is an extensional MLTT with three cumulative universes $P : V_1 : V_2$ such that

- each closed under the unit type, Σ, Π, and inductive types (W-types);
- for all types A and $a, b : A$, the equality type $a = b$ is in the universe P;
- for all types A and P-valued type families $B : A \to P$, $\Pi\ A\ B$ is in P.

The interpretation of P is the assembly of modest sets (i.e. PERs), and V_i is the assembly of U_i-small assemblies, for universe of sets U_i in the meta-theory. Details of the interpretation can be found in Reus's thesis [Reus 1996, §8].

5.6.2∗7. In the following, we will further fix the PCA $\mathbb{A}$ to be *Kleene's first algebra* $\mathbb{K}$ [Oosten 2008], whose elements are natural numbers (which intuitively play dual roles as both *data* and *computation* via Gödel codes of Turing machines), and partial application $n\ m$ is defined to be $\phi_n(m)$, the possibly divergent result of running the Turing machine coded by n with input m. We will write $n\ m \uparrow$ and $n\ m \downarrow$ to mean that the partial application diverges and converges respectively.

Specialising $\mathbb{A}$ to $\mathbb{K}$ is only for providing more intuition, and interested readers can consult Longley and Simpson [1997] to see how it can be done more generally with an arbitrary PCA equipped with a notion of *divergence*.

5.6.2∗8. A type A is said to be a *proposition* if the type *is-prop* $A := (a, b : A) \to x = y$ is inhabited [Univalent Foundations Program 2013]. The subuniverse $P_{-1} \subseteq P$ of *propositional modest sets* is then defined by

$$\begin{aligned} &P_{-1} : V_1 \\ &P_{-1} = \Sigma(A : P).\ \textit{is-prop}\ A \end{aligned}$$

whose elements are decoded as types by first projection π_1, which we will left as implicit, as if P_{-1} is a Russell-style universe.

It might be useful to see an external description for the universe P_{-1}, in the sense of universes in categories (Section 4.3.1). The semantics of the universe (P_{-1}, π_1) is (isomorphic to) an assembly morphism $i : \tilde{P}_{-1} \to P_{-1}$, where P_{-1} has an underlying set containing all *sub-singleton modest sets* A, and $r \Vdash_{P_{-1}} A$ holds for all r and A. The assembly $\tilde{P}_{-1}$ has an underlying set containing all modest sets A with exactly one element $a \in |A|$, and $r \Vdash_{\tilde{P}_{-1}} A$ if and only if $r \Vdash_A a$. The morphism $i : \tilde{P}_{-1} \to P_{-1}$ is the inclusion morphism.

From the above explicit description, we can see that the universe P_{-1} satisfies Voevodsky's *propositional resizing axiom*: in the internal language of Asm($\mathbb{K}$), for every propositional type A, there is some $\lceil A \rceil : P_{-1}$ isomorphic to A, since a sub-singleton assembly A is necessarily a modest set.

5.6.2∗9. The universe P_{-1} is similar to the universe Ω of propositions in elementary toposes as axiomatised in 5.3.3∗3, and we can define logical connectives $\top$, $\bot$, $\wedge$, $\vee$, $\forall$, and $\exists$ on P_{-1} in exactly the same way as we did in Section 5.3.3.

The crucial difference between P_{-1} and the universe Ω in elementary toposes is that P_{-1} is *not* univalent: given $p, q : P_{-1}$ with $p \cong q$, it is not always the case that $p = q$. Indeed, two singleton modest sets $\langle \{*\}, \models_p \rangle$ and $\langle \{*\}, \models_q \rangle$ can have different realizing relations even when their underlying sets are exactly the same.

This seemingly insignificant flaw of P_{-1} has an impact bigger than one may expect on doing mathematics internal to $\textsc{Asm}(\mathbb{K})$. For one thing, we cannot construct *quotient types* using P_{-1} as we did in 5.3.3∗12: $[x]$ and $[y]$ will not be equal for $R\ x\ y$ if we follow the formula in 5.3.3∗12.

We can switch to the realizability topos to use the better-behaved universe Ω, but we will stay in category of assemblies, as it turns out to be good enough for carrying out our development, and more importantly, the simplicity of $\textsc{Asm}(\mathbb{K})$ allows us to give simple external descriptions of many constructions of SDT, which I found essential when learning SDT for the first time.

5.6.2∗10. The universe P_{-1} has a subuniverse of *semi-decidable* propositions:

$$P^s_{-1} := \{p : P_{-1} \mid \exists f : \mathbb{N} \to 2.\ p \cong (\exists n : \mathbb{N}.\ f\ n = 0)\}. \tag{5.29}$$

Roughly speaking, a proposition in P^s_{-1} is determined by the (semi-decidable) property of the existence of zero points for a computable function $f : \mathbb{N} \to 2$.

A simple external description of P^s_{-1} is available: the assembly P^s_{-1} is isomorphic, *up to bi-implication* in P_{-1}, to the assembly $\mathfrak{S} := \langle \{\bot, \top\}, \models_{\mathfrak{S}} \rangle$ where

$$r \models_{\mathfrak{S}} \bot \quad \text{iff} \quad r\ 0 \uparrow \qquad\qquad \text{and} \qquad\qquad r \models_{\mathfrak{S}} \top \quad \text{iff} \quad r\ 0 \downarrow .$$

In sketch, the direction $\mathfrak{S} \to P^s_{-1}$ sends $\bot$ and $\top$ to the empty and terminal assemblies respectively, and is realized by the Turing machine accepting r and returning the computable function $f : \mathbb{N} \to 2$ that accepts n and returns 0 if and only if running the Turing machine r halts in n steps. The other direction $P^s_{-1} \to \mathfrak{S}$ sends sends an assembly to $\top$ iff it is non-empty, and this is realized by the Turing machine accepting the code for $f : \mathbb{N} \to 2$ (and $p \cong \exists n.\ f\ n = 0$) and returns the machine r that searches for a zero point of f iteratively.

5.6.2∗11. The type $\mathfrak{S}$ can be viewed as a universe directly: every element $p : \mathfrak{S}$ is decoded as the equality type $p = \top$. Although P^s_{-1} and $\mathfrak{S}$ are equivalent universes, we will prefer using the universe $\mathfrak{S}$ over P^s_{-1} because $\mathfrak{S}$ is univalent:

$$(p, q : \mathfrak{S}) \to (p \cong q) \to (p = q).$$

The universe $\mathfrak{S}$ is closed under truth $\top : \mathfrak{S}$ and dependent conjunction $\Sigma(p : \mathfrak{S}).\ q(p) : \mathfrak{S}$ for all $p : \mathfrak{S}$ and $q : p \to \mathfrak{S}$. Therefore, it is a *dominance*

[Rosolini 1986], which is the fundamental notion in general SDT [Hyland 1991]. In the present situation, the dominance $\mathfrak{S}$ is moreover closed under falsity $\bot$ and countable disjunction $\exists n : \mathbb{N}.\ p(n)$ for $p : \mathbb{N} \to \mathfrak{S}$.

As suggested by the notation, the universe $\mathfrak{S}$ of semi-decidable propositions will play the role of the Sierpiński space $\{\bot \sqsubseteq \top\}$ in classical domain theory (5.6.2∗5). In the internal language, a (Scott-) open of a type A is defined as a function $O : A \to \mathfrak{S}$, giving rise to a subtype $\{a : A \mid O\ a = \top\}$, which we shall usually just write as O when no confusion. An ($\mathfrak{S}$-) partial function $\Gamma \rightharpoonup A$ is again an open set O of Γ with a function $O \to A$. Externally, an open set of an assembly $\langle |A|, \Vdash_A \rangle$ is a subset $O \subseteq A$ such that there is some Turing machine r satisfying that whenever $n \Vdash_A a$, then $r\ n \downarrow$ iff $a \in O$.

5.6.2∗12. Analogous to the lifting monad in 5.6.2∗5, we have a lifting monad

$$\begin{aligned} &L : P \to P \\ &L\ A = \Sigma(p : \mathfrak{S}).\ (\{p\} \to A) \end{aligned}$$

on modest sets in the internal language of Asm($\mathbb{K}$). We can actually define L on all types but we shall only need it on P. The monad structure for L is

$$\begin{aligned} &\eta : A \to L\ A && \mu : (A \to L\ B) \to L\ B \\ &\eta\ a = (\top, a) && \mu\ (p, a)\ k = (\Sigma(\_ : p).\ \pi_1\ (k\ a),\ \pi_2\ (k\ a)) \end{aligned}$$

An isomorphic external description of the monad L is that it sends every modest set $\langle |A|,\ \Vdash_A \rangle$ to the modest set $\langle 1 + |A|,\ \Vdash_{LA} \rangle$ where

$$\begin{aligned} r \Vdash_{LA} \mathrm{inl}\ * \quad &\text{iff} \quad r\ 0 \uparrow, \\ r \Vdash_{LA} \mathrm{inr}\ a \quad &\text{iff} \quad r\ 0 \downarrow\ \wedge\ r\ 0 \Vdash_A a. \end{aligned}$$

That is to say, if a Turing machine r diverges on the input 0 then it realizes the 'bottom element' inl $*$, and otherwise r realizes inr a, for elements $a \in |A|$ that are realized by $r\ 0$. The input 0 here is completely arbitrary, and the definition will be isomorphic if 0 is replaced by any other fixed number or r itself. The monad structure on L is the same as the one on $1 + - : \textsc{Set} \to \textsc{Set}$; see Longley and Simpson [1997, §4] for details.

Note that LA is not the same as the coproduct $1 + A$ in Asm($\mathbb{K}$). The latter has the same underlying set $1 + |A|$, but the existence predicate of $1 + A$ is

$$\begin{aligned} r \Vdash_{1+A} \mathrm{inl}\ * \quad &\text{iff} \quad \pi_1\ r = 0 \\ r \Vdash_{1+A} \mathrm{inr}\ a \quad &\text{iff} \quad \pi_1\ r \neq 0 \wedge \pi_2\ r \Vdash_A a, \end{aligned}$$

where π_1, π_2, and $\langle -, - \rangle$ are some Turing machines implementing projections and pairing of natural numbers $\mathbb{N} \times \mathbb{N} \cong \mathbb{N}$. The crucial difference between LA and $1 + A$ is that morphisms of assemblies $f : X \to 1 + A$ must be realised

by Turing machines that can *decide* whether $f(x)$ is inr a given a realizer of x, while morphisms $f : X \to LA$ need only be realised by Turing machines that *semi-decide* whether $f(x)$ is inr a. Thus LA is the right choice of the lifting monad capturing the idea of 'possibly divergent elements of A'.

5.6.2∗13. As an *endofunctor* on the universe P, L has both a final coalgebra $\langle \bar{\omega} : P, \sigma : \bar{\omega} \to L\bar{\omega} \rangle$ and an initial algebra $\langle \omega : P, \tau : L\omega \to \omega \rangle$. The following formulae of $\bar{\omega}$ and ω are due to Jibladze [1997]:

$$\bar{\omega} = \{f : \mathbb{N} \to \mathfrak{S} \mid \forall n.\ f\ (n+1) \to f\ n\}$$
$$\omega = \{f : \bar{\omega} \mid \forall p : P_{-1}.\ \big(\forall (n : \mathbb{N}).\ (f\ n \to p) \to p\big) \to p\}$$

which in fact works for any dominance in any elementary topos with a natural number object (with P_{-1} in the formula of ω replaced by Ω).

Again, we have simple external descriptions for $\bar{\omega}$ and ω in the case of $\textsc{Asm}(\mathbb{K})$. The carrier $\bar{\omega}$ is isomorphic to the assembly $\langle \mathbb{N} \cup \{\infty\}, \models_{\bar{\omega}} \rangle$ where

$$r \models_{\bar{\omega}} n \quad \text{iff} \quad \forall k \in \mathbb{N}.\ (k < n) \leftrightarrow (r\ k\downarrow)$$
$$r \models_{\bar{\omega}} \infty \quad \text{iff} \quad \forall k \in \mathbb{N}.\ r\ k\downarrow$$

with $\sigma : \bar{\omega} \to L\bar{\omega}$ given by $\sigma(0) = \text{inl}\ *$, $\sigma(n+1) = \text{inr}\ n$, and $\sigma(\infty) = \text{inr}\ \infty$. The type ω is given by the assembly $\langle \mathbb{N}, \models_{\omega} \rangle$ that restricts $\bar{\omega}$ to the sub-underlying set $\mathbb{N}$. The algebra $\tau : L\omega \to \omega$ is simply $\tau(\text{inl}\ *) = 0$ and $\tau(\text{inr}\ n) = n + 1$.

From the explicit description we see that the assembly ω is a non-standard representation of natural numbers as an assembly, different from the standard representation $\langle \mathbb{N}, \{(n, n) \mid n \in \mathbb{N}\} \rangle$ that satisfies the universal property of a natural number object in $\textsc{Asm}(\mathbb{K})$. In ω, every natural number $n \in \mathbb{N}$ is represented as a Turing machine that halts exactly for inputs k smaller than n. Since Turing machines are unable to tell whether other machine halts, assembly maps $\omega \to A$ are constrained to be 'continuous' in a sense.

5.6.2∗14. Let $\kappa : \omega \to \bar{\omega}$ be the canonical inclusion morphism (given as the unique algebra homomorphism from the initial algebra $\tau : L\omega \to \omega$ to the L-algebra $\sigma^{-1} : L\bar{\omega} \to \bar{\omega}$). The morphism κ plays an important role in synthetic domain theory: a morphism $c : \omega \to X$ of assemblies will play the role of an ω-chain of elements $c_0 \sqsubseteq c_1 \sqsubseteq \cdots$ in a partial order X. Similarly, a morphism $c^* : \bar{\omega} \to X$ is analogous to a chain c_i together with its supremum $\sqcup_i c_i$.

5.6.2∗15 Definition. A modest set $X : P$ is called *complete* if the function

$$(- \cdot \kappa) : (\bar{\omega} \to X) \to (\omega \to X)$$

is an isomorphism, i.e. the following proposition holds

$$\textit{complete}\ X := \exists \overline{(-)} : (\omega \to X) \to (\bar{\omega} \to X).\ (\forall c.\ \bar{c} \cdot \kappa = c) \wedge (\forall d.\ \overline{d \cdot \kappa} = d)$$

internally in Asm(𝕂). A modest set $X : P$ is called *well complete* if LX is complete.

5.6.2∗16. Well complete types will be our synthetic version of predomains:

$$PDom : V_1$$
$$PDom = \Sigma(A : P).\ complete\ (L\ A)$$

They are intuitively modest sets in which an ω-chain of partial elements has a unique (partial) supremum. Their crucial difference from predomains in classical domain theory is that they are just 'sets' satisfying a property, rather than sets carrying additional data (the partial order).

5.6.2∗17. There are some nuances in the external meaning of (well) completeness. Firstly, we notice that $complete : P \to P_{-1}$ in Definition 5.6.2∗15 is a proper *realizability predicate* in the sense that *complete* X for a modest set $X : P$ has non-trivial realizers. Namely, *complete* X is realized by machines sending realizers of $\omega \to X$ to realizers of $\bar{\omega} \to X$, in a way that is an inverse to $- \cdot \kappa$. Secondly, *complete* $X : P_{-1}$ makes sense in an arbitrary context Γ in the internal language of Asm(𝕂). Therefore, externally X is not one modest set but a family of modest sets $\Gamma \to P$ indexed by an assembly Γ of the context.

If we forget about realizers of completeness and consider only *global* elements $X : P$, then *complete* X has a realizer if and only if the morphism $X^\kappa : X^{\bar{\omega}} \to X^\omega$ in Asm(𝕂) is an isomorphism. The latter condition is precisely the definition of completeness for an object in Asm(𝔸) by Longley and Simpson [1997]. This further means that for all assemblies Γ and $c : \Gamma \times \omega \to X$, there is a unique $\bar{c} : \Gamma \times \bar{\omega} \to X$ making the following diagram commute:

$$\begin{array}{ccc} \Gamma \times \omega & \xrightarrow{c} & X \\ {\scriptstyle \kappa}\downarrow & \nearrow_{\bar{c}} & \\ \Gamma \times \bar{\omega} & & \end{array}$$

To see this, by Yoneda embedding, X is complete if and only if

$$(X^\kappa \cdot -) : \mathrm{Hom}(\Gamma, X^{\bar{\omega}}) \to \mathrm{Hom}(\Gamma, X^\omega)$$

is an isomorphism, natural in Γ. By adjointness, this is equivalent to asking

$$(- \cdot \Gamma \times \kappa) : \mathrm{Hom}(\Gamma \times \bar{\omega}, X) \to \mathrm{Hom}(\Gamma \times \omega, X)$$

to be a natural isomorphism. A natural transformation is a natural isomorphism iff every component of it is an isomorphism, so X is complete if and only if for every $c : \Gamma \times \omega \to X$, there is a unique $\bar{c} : \Gamma \times \bar{\omega} \to X$ such that $\bar{c} \cdot \Gamma \times \kappa = c$.

5.6.2∗18 Theorem. *The subuniverse $PDom \subseteq P$ is closed under liftings L, the unit type, Σ-types, Π-types, equality types, coproducts, the natural number type $\mathbb{N}$ in P.*

Moreover, predomains are also complete (*i.e. well completeness implies completeness*).

Proof. This is essentially shown by Longley and Simpson [1997, §7] aside from the difference between Longley and Simpson's external definition of completeness and our internal definition (5.6.2∗17). Longley and Simpson defined completeness of an assembly X as a proposition in the ambient logic ($X^\kappa : X^{\bar{\omega}} \to X^\omega$ being an isomorphism), while our definition is in internal in the language of Asm(𝕀K), which has non-trivial realizers (Turing machines accepting code of $c : \omega \to X$ and outputting code of $\bar{\omega} \to X$). Therefore, we have to check that the proofs of the closure properties by Longley and Simpson are realizable. For example, to show $L : P \to P$ restricts to $L : PDom \to PDom$, we need to check that there is a Turing machine sending realizers of X being well complete to realizers of LX being well complete. This is indeed the case by observing that the proofs by Longley and Simpson can be carried internally in the language of Asm(𝕀K). □

5.6.2∗19 Definition. Mirroring the setup of 5.6.2∗5, the universe of *domains* is defined as the type of Eilenberg-Moore algebras of the monad $L : PDom \to PDom$:

$$\begin{aligned}&\textsc{record}\ Dom : V_1\ \textsc{where}\\ &\quad A : PDom\\ &\quad \alpha : L\ A \to A\\ &\quad \_ : (x : A) \to \alpha\ (\eta^L\ x) = x\\ &\quad \_ : (x : L\ (L\ A)) \to \alpha\ (\mu^L\ x) = \alpha\ (L\ \alpha\ x)\end{aligned}$$

As usual, given $D : Dom$, we usually write the type $\pi_1\ (D.A)$ as just D.

5.6.2∗20. The crucial property of domains $D : Dom$ is that they admit fixed points for all endofunctions $f : D \to D$:

$$fix : \{D : Dom\} \to (f : D \to D) \to D$$

which is defined as follows: first we define a function $\alpha_f : L\ D \to D$ by $\alpha_f = \alpha \cdot Lf$. By the initiality of $\tau : L\omega \to \omega$, we have a homomorphism $c : \omega \to D$. Then using the completeness of D, we have $\bar{c} : \bar{\omega} \to D$, and we let $fix\ f := \bar{c}\ \infty$. It is then the case that $f\ (fix\ f) = fix\ f$ [Reus and Streicher 1999, Theorem 7.3].

5.6.2∗21. We note in passing that by Longley and Simpson [1997, Theorem 5.6], Eilenberg-Moore L-algebras on a predomain are unique if exist, so it makes sense to say 'a predomain is a domain' as a proposition.

5.6.3 The Interpretation of $\mathrm{rF}_\omega^{\mathrm{ha}}$

5.6.3∗1. Now we are ready to construct a (V_2-small) model of the signature $\mathrm{rF}_\omega^{\mathrm{ha}}$ (Section 5.6.1) in Asm(𝕀K). Our goal is to define an element M of the record type $[\![\mathrm{rF}_\omega^{\mathrm{ha}}]\!]_{V_2}$ containing all the declarations of $\mathrm{rF}_\omega^{\mathrm{ha}}$ with $\mathbb{J}$ replaced by the universe V_2.

5.6.3∗2. The non-recursive fragment of $\mathrm{rF}_\omega^{\mathrm{ha}}$ will be interpreted in almost the same way as $\mathrm{F}_\omega^{\mathrm{ha}}$ in Section 5.2, except that all the occurrences of the universe $P : V_1$ in the model will be replaced by the subuniverse $PDom : V_1$. For example, $M.ty$ is now $PDom$ instead of P, and the computation judgement $M.co$ is now

$$M.co : M.RawHFunctor \to M.el\ M.ty \to PDom$$
$$M.co\ H\ A = (T : M.MonadAlg\ H) \to (B : PDom) \to (A \to T\ B) \to T\ B$$

The constructions in Section 5.2 still work because by Theorem 5.6.2∗18, the universe $PDom$ is closed under the type formers that we used to interpret $\mathrm{F}_\omega^{\mathrm{ha}}$, in particular, impredicative Π-types.

5.6.3∗3. The empty type from 5.6.1∗4 is as expected interpreted as the empty modest set 0, which is trivially well complete.

5.6.3∗4. The interesting thing is modelling partial computations *pco* (5.6.1∗1). In 5.6.1∗3, we had a type *MonadAlgRec* of monads supporting recursion (and some effectful operations). However, we cannot simply define $M.pco$ by replacing *MonadAlg* in the definition of $M.co$ above with *MonadAlgRec*, because the definition of *MonadAlgRec* depends on $M.pth$ and thus $M.pco$.

The type *MonadAlgRec* ensures that a monad T in $\mathrm{rF}_\omega^{\mathrm{ha}}$ supports recursion by requiring the monad T to be partial thunks *syntactically*. What we need here is a semantic counterpart of monad supporting recursion:

$$\begin{aligned}&\textsc{record}\ MonadAlgL\ (H : RawHFunctor) : V_2\ \textsc{where}\\ &\quad \textsc{include}\ MonadAlg\ H\ \textsc{as}\ T\\ &\quad dom : (A : PDom) \to \{D : Dom\ \mid D.A = T\ A\}\end{aligned}$$

which requires that $T\ A : PDom$ is a domain for all $A : PDom$.

The model of partial computations is then

$$M.pco : M.RawHFunctor \to M.el\ M.ty \to PDom$$
$$M.pco\ H\ A = (T : MonadAlgL\ H) \to (B : PDom) \to (A \to T\ B) \to T\ B$$

The models of the declarations *val*, *let-in*, *pth* and *op* are the same as those for *co* in Section 5.2, which we shall not repeat here.

5.6.3∗5. The model for the fixed-point combinator has type

$$M.Y : \{H, A\} \to (M.pth\ H\ A \to M.pco\ H\ A) \to M.pco\ H\ A$$

In the present model, $pth\ H\ A$ is simply equal to $pco\ H\ A$, so by 5.6.2∗20, it is sufficient to show that $M.pco\ H\ A$ is a domain. We define the algebra by

$$\alpha : \{H, A\} \to L\ (M.pco\ H\ A) \to M.pco\ H\ A$$
$$\alpha\ (p, c) = \lambda T\ B\ k.\ \beta_{TB}\ (p, c\ T\ B\ k)$$

where $\beta_{TB} : L\ (T\ B) \to T\ B$ is $\beta_{TB} := (T.dom\ B).\alpha$. The L-algebra α is a product of a family of L-algebras, so it is easy to check that α satisfies the laws:

$$\begin{aligned}
&\alpha\ (\top,\ c)\\
&= \lambda T\ B\ k.\ \beta_{TB}\ (\top,\ c\ T\ b\ k)\\
&= \quad \{\beta_{TB} \text{ is an Eilenberg-Moore algebra}\}\\
&\quad \lambda T\ B\ k.\ c\ T\ b\ k\\
&= c
\end{aligned}$$

and similarly for all $(p, (q, c)) : L\ (L\ (M.pco\ H\ A))$,

$$\begin{aligned}
&\alpha\ (L\ \alpha\ (p, (q, c)))\\
&= \alpha\ (p, \lambda T\ B\ k.\ \beta_{TB}\ (q, c\ T\ B\ k))\\
&= \lambda T\ B\ k.\ \beta_{TB}\ (p, \beta_{TB}\ (q, c\ T\ B\ k))\\
&= \quad \{\beta_{TB} \text{ is an Eilenberg-Moore algebra}\}\\
&\quad \lambda T\ B\ k.\ \beta_{TB}\ (\mu^L\ (p, (q, c\ T\ B\ k)))\\
&= \lambda T\ B\ k.\ \beta_{TB}\ (\Sigma(_ : p).\ q,\ c\ T\ B\ k)\\
&= \alpha\ (\mu^L\ (p, (q, c)))
\end{aligned}$$

We have shown that $pco\ H\ A$ is a domain, so we can use *fix* (5.6.2∗20) to define

$$M.Y\ f = fix\ f.$$

5.6.3∗6. Finally, we need to give an interpretation of *eval* from 5.6.1∗3:

$$\begin{aligned}
M.eval : \{H\} \to (T : M.MonadAlgRec\ H) \to (A : M.el\ ty)\\
\to M.pco\ H\ A \to M.tm\ (T\ A)
\end{aligned}$$

Note that the type of T is *MonadAlgRec* rather than *MonadAlgL*. By the definition of *MonadAlgRec* in 5.6.1∗3, there exists some $F : M.el\ M.ty \to M.el\ M.ty$ such that the monad T maps every $A : PDom$ to $M.pth\ H\ (F\ A)$. By the discussion above in 5.6.3∗5, $M.pth\ H\ (F\ A)$, which is just $M.pco\ H\ (F\ A)$, is always a domain. Therefore we have a conversion function

$$\sigma : (T : M.MonadAlgRec\ H) \to \{T' : MonadAlgL\ H \mid T.T = T'.T\},$$

and we define the model of *eval* to be

$$M.eval\ T\ A\ c = c\ (\sigma\ T)\ A\ T.ret$$

This completes the definition of the model $M : [\![\mathrm{rF}_\omega^{\mathrm{ha}}]\!]_{V_2}$.

5.6.3∗7. The realizability model M of $\mathrm{rF}_\omega^{\mathrm{ha}}$ gives a way to compute recursive programs written in $\mathrm{rF}_\omega^{\mathrm{ha}}$ by program extraction. Since $L\ A$ is a domain, the

monad $L : \mathit{PDom} \to \mathit{PDom}$ can be extended to

$$L' : \{L' : M.\mathit{MonadAlgL}\ \mathit{VoidH} \mid L'.T = L\}$$

Therefore we have a function

$$\begin{aligned}&\mathit{toL} : \{A : \mathit{PDom}\} \to M.\mathit{pco}\ \mathit{VoidH}\ A \to L\ A\\&\mathit{toL}\ c = c\ L'\ A\ \eta^L\end{aligned}$$

Every closed program $p : \mathit{pco}\ \mathit{VoidH}\ \mathit{bool}$ is interpreted as a global element of $M.\mathit{pco}\ \mathit{VoidH}\ 2$ in $\textsc{Asm}(\mathbb{K})$. Composing it with *toL*, we then have a global element of the modest set $L\ 2$. The realizer of this element is then a possibly divergent Turing machine r that yields a Boolean value if it halts.

5.6.3∗8. We conjecture that the realizability model of $\mathrm{rF}_\omega^{\mathrm{ha}}$ in this section is *adequate* in the following sense:

> For all closed program $c : \mathit{pco}\ \mathit{VoidH}\ \mathit{bool}$ in $\mathrm{rF}_\omega^{\mathrm{ha}}$, if the morphism $\mathit{toL} \cdot [\![c]\!] : 1 \to L\ 2$ in $\textsc{Asm}(\mathbb{K})$ is inr tt (or inr ff), then $c = \mathit{val}\ \mathit{tt}$ (or $c = \mathit{val}\ \mathit{ff}$) in the theory of $\mathrm{rF}_\omega^{\mathrm{ha}}$.

This implies that if $\mathit{toL} \cdot [\![c]\!] = \mathrm{inl}\ *$, then c does not equal to $\mathit{val}\ \mathit{tt}$ or $\mathit{val}\ \mathit{ff}$, otherwise $\mathit{toL} \cdot [\![c]\!]$ would not be inl $*$.

We expect adequacy can be proved using synthetic Tait computability (STC) *internally* in the effective topos Eff, in which we glue the (internal) category P with the category of P-valued presheaves over the category of judgements of $\mathrm{rF}_\omega^{\mathrm{ha}}$ (constructed internally in Eff). Such an internal STC argument has been attempted by Sterling and Harper [2022] to prove adequacy for a language with *security levels* and *general recursion* but without impredicative polymorphism, whose denotational semantics is a sheaf-model of SDT, although there is a currently unfixed problem in their proof [Sterling 2023].

Chapter 6

Epilogue

6∗1. Let us conclude this thesis with a summary of what we have done in each chapter and some potential future directions.

6∗2. In Chapter 2, we explored some basic properties of the category of equational systems, which we used as a convenient way for presenting algebraic theories. Future work about equational systems includes the following.

1. Whenever an equational system has the free-forgetful adjunction it is monadic, and we have paid our attention solely to such equational systems in this thesis. It is not clear to me whether equational systems that do not have the free-forgetful adjunction is useful. Particularly, do they make the category of equational systems as a kind of completion of the category of monads?

2. Related to the last question, what is an intrinsic characterisation (similar to the monadicity theorem or the definition of algebraic categories) of functors $U_{\dot{\Sigma}} : \dot{\Sigma}\text{-}\textsc{Alg} \to \mathscr{C}$ arising from equational systems $\dot{\Sigma}$?

3. The framework of equational systems is 1-categorical. Although in 2.2.1∗9 we noted that inequations or higher-dimensional cells can be encoded in equational systems, they have to be preserved *strictly* by algebra homomorphisms. Therefore it is worthwhile in the long term to develop a 2-dimensional theory of equational systems, mimicking the 2-dimensional theory of monads [Blackwell et al. 1989].

6∗3. In Section 2.3, we introduced *monoidal algebraic theories* for describing constructions over monoidal categories conveniently. Besides the questions in 2.3.1∗10, proving normalisation of the terms of a monoidal algebraic theory without equational axioms is also an interesting question. Ideally, we could extend synthetic Tait computability to handle linear variable contexts.

Another interesting direction about monoidal algebraic theories is *functional data structures* in monoidal categories: the inductive datatype $\mu X.\ I + A \,\square\, X$ is one representation of $\square$-lists over A, but there are many other sophisticated representations of lists in functional programming with better asymptotic complexities [Hinze and Paterson 2005; Okasaki 1998]. These data structures were

invented in languages such as Haskell or ML, but some of them are still valid in (symmetric) monoidal algebraic theories, therefore can be interpreted in arbitrary (symmetric) monoidal categories. An initial example is Okasaki's *catenable list*, which supports amortised constant time list concatenation and pattern matching. After a re-inspection of Okasaki's algorithm, we can see that it is valid in any closed symmetric monoidal category, so we can derive from it an efficient representation of free applicative functors with amortised constant time multiplication and pattern matching [Yang 2022]. It is interesting to revisit those functional data structures and find out which of them can be generalised in this way. Particularly, the author hopes to find an efficient version of free monads to optimise substitution based normalisation algorithms. If we managed to find a representation of free monads whose substitution operation needs only $O(1)$-time for each variable in the term, with suitable lazy evaluation, a naive substitution based evaluator for untyped λ-calculus (with all bound variables renamed to be distinct a priori) should be asymptotically as efficient as the normalisation-by-evaluation algorithm for evaluating a term.

6∗4. In Sections 2.4 and 2.5, we looked into theories of operations on a monoid and some subcategories of such theories. The main results here are that the category of algebraic operations is equivalent to the category of monoids (Theorem 2.5∗10), and that the category of algebraic operations is a coreflective subcategory of all operations with free-forgetful adjunction (Theorem 2.5∗14). Currently these two sections lack concrete examples, so the most important future work is to study more concrete examples with the abstract framework here. Specifically, theories of operations on applicative functors seem interesting to explore.

6∗5. In Chapter 3, we proposed a formal theory of *modularity* in algebraic structures using the framework of lifting functors along fibrations, or *model transformers* as how we called them in this context. As the first step of this theory, a number of general and concrete constructions were presented (3.3∗1). Although this framework is only applied to models of computational effects in this thesis, the author hopes that in the future this framework can be applied to theories and models of more interesting programming languages to mitigate one aspect of programming language theory nowadays that the author finds very unsatisfactory – the fact that theorems about programming language are so rarely reusable. However, the framework in this thesis is still quite crude, and more subtle forms of modularity, such as the modularity in normalisation proofs of programming languages, need to be uncovered with more careful analysis.

6∗6. In Chapter 4, we gave an introduction to Sterling's [2021] logical framework and proved in detail the existence of classifying locally cartesian closed categories for signatures defined in this logical framework (Theorem 4.4∗10). The most

useful future work is probably mechanising the logical framework in a proof assistant. It is not difficult to implement a type checker and an evaluator for the logical framework by for example embedding it in Agda using postulates and rewriting rules, but this is not enough for formalising models of signatures, which do not live in the logical framework. Instead, a full formalisation will need to carry out everything we did in Chapter 4 in a proof assistant (e.g. Cubical Agda), which will be a much more ambitious formalisation project.

6∗7. In Chapter 5, we defined $\mathrm{F}_\omega^{\mathrm{ha}}$, an extension of F_ω with (a restricted form of) higher-order algebraic effects. We gave a denotational model of it using realizability and proved the canonicity of closed terms using synthetic Tait computability. A further extension with general recursion was introduced and was modelled using synthetic domain theory. Future work abound:

1. We should be able to prove normalisation of open $\mathrm{F}_\omega^{\mathrm{ha}}$-terms following the lines of Sterling [2021]. A subtlety is that we will need in TTstc an *impredicative* universe U_0 that contains the normalisation model and the syntactic model $M : \{\mathfrak{ob}\} \to [\![\mathrm{F}_\omega^{\mathrm{ha}}]\!]_{U_0}$, which means that in the first place the category of judgements of $\mathrm{F}_\omega^{\mathrm{ha}}$ has to be constructed in some impredicative universe of the ambient meta-theory, and as we commented in 4.5∗2, this should be workable since we did not rely on anything classical in Chapter 4.

2. The language $\mathrm{F}_\omega^{\mathrm{ha}}$ is a core calculus. Although we commented in Section 5.1.4 that important features such as effect systems and modular models may be implemented as libraries, for practical use they should be supported in a more seamless way, such as by elaboration or by directly baking into the language.

3. Although our categorical foundation allows arbitrary monoidal structures, only monadic computations are considered in $\mathrm{F}_\omega^{\mathrm{ha}}$ for simplicity. Generalising $\mathrm{F}_\omega^{\mathrm{ha}}$ from monads to arbitrary user-defined (left closed) monoidal structures should be very useful. There does not seem to be any theoretical obstacle, but designing a user-friendly syntax, or allowing user-designed syntax, may be challenging.

4. Efficiency of implementations is also an interesting aspect. Note that a continuation-passing style translation for $\mathrm{F}_\omega^{\mathrm{ha}}$-computations can be readily extracted from the realizability model of $\mathrm{F}_\omega^{\mathrm{ha}}$, which will be (asymptotically) faster than a naive implementation based on structural operational semantics, but it should be possible to further optimise out computations at all for statically known computations and handlers.

5. We based $\mathrm{F}_\omega^{\mathrm{ha}}$ on fine-grain call-by-value (FGCBV) rather than call-by-push-value (CBPV) since our theory of higher-order algebraic effects was based on monads/monoids rather than adjunctions, but CBPV is also possible and we sketch the judgements for a CBPV variant of $\mathrm{F}_\omega^{\mathrm{ha}}$ (without stack judgements) here. Again, starting with F_ω, instead of $co : \mathit{RawHFunctor} \to \mathit{el}\ \mathit{ty} \to \mathbb{J}$, we add to F_ω a

new kind for *computation types* and a judgement for *computation terms*:

$$cty : RawHFunctor \to ki \qquad ctm : \{H\} \to el\ (cty\ H) \to \mathbb{J}$$

We then add two new type formers for value-returning computations (called *returners* by Levy [2003]) and thunk values:

$$F : (H : RawHFunctor) \to el\ ty \to el\ (cty\ H) \quad U : \{H\} \to el\ (cty\ H) \to el\ ty$$

We have value returning and sequential composition as usual:

$$\begin{aligned} &val \quad : \{H, A\} \to tm\ A \to ctm\ (F\ H\ A) \\ &let\text{-}in : \{H, A, X\} \to ctm\ (F\ H\ A) \to (tm\ A \to ctm\ X) \to ctm\ X \end{aligned}$$

Note that the second argument of *let-in* can be an arbitrary computation type $X : el\ (cty\ H)$ rather than just value-returning computations $F\ H\ A : el\ (cty\ H)$. Terms of a thunk type are in bijection with the terms of the computation type:

$$U\text{-}iso : \{H\}\ \{X : el\ (cty\ H)\} \to tm\ (U\ X) \cong ctm\ X$$

The computation judgement $co\ H\ A$ in $\mathrm{F}^{\mathrm{ha}}_{\omega}$ then corresponds to $ctm\ (F\ H\ A)$ in the CBPV language. What we have in CBPV but not in FGCBV are *function computation types* from a value type A to a computation type X:

$$\begin{aligned} &\_\Rightarrow_c\_ \ : \{H\} \to (A : el\ ty) \to (X : el\ (cty\ H)) \to el\ (cty\ H) \\ &\Rightarrow_c\text{-}iso : \{H, A, X\} \to ctm\ (A \Rightarrow_c X) \cong (tm\ A \to ctm\ X) \end{aligned}$$

Finally, we have H-operations and evaluation by raw monads with H-operations:

$$\begin{aligned} &th : RawHFunctor \to el\ ty \to el\ ty \\ &th\ H\ A = U\ (F\ H\ A) \\ &op : \{H, A, X\} \to tm\ (H\ (th\ H)\ A) \to (tm\ A \to ctm\ X) \to ctm\ X \\ &eval : \{H, A\} \to (m : MonadAlg\ H) \to ctm\ (F\ H\ A) \to tm\ (m\ .M_0\ A) \end{aligned}$$

The author sees no obvious difficulties in adapting the rest of the development for $\mathrm{F}^{\mathrm{ha}}_{\omega}$ in Chapter 5 to this CBPV variant by interpreting F and U using the Eilenberg-Moore adjunction of the monads that we used to model $\mathrm{F}^{\mathrm{ha}}_{\omega}$.

6. The biggest limitation of $\mathrm{F}^{\mathrm{ha}}_{\omega}$ is probably that equations have no place in the language, since we did not have dependent types, in particular equality/identity types, in $\mathrm{F}^{\mathrm{ha}}_{\omega}$. Adding (intensional) identity types to $\mathrm{F}^{\mathrm{ha}}_{\omega}$ is straightforward

$$\begin{aligned} &Id \ : \{A : el\ ty\} \to (a, b : tm\ A) \to el\ ty \\ &refl : \{A : el\ ty\} \to (a : tm\ A) \to el\ (Id\ a\ a) \\ &J \quad : \{A : el\ ty\}\ \{C : (a, b : tm\ A) \to Id\ a\ b \to el\ ty\} \\ &\qquad \to ((x : tm\ A) \to C\ x\ x\ (refl\ x)) \\ &\qquad \to \{a, b : tm\ A\} \to (p : Id\ a\ b) \to C\ a\ b\ p \end{aligned}$$

and Σ and Π types are no more difficult. With *Id* we can then define in $\mathrm{F}^{\mathrm{ha}}_{\omega}$ *law-abiding* functors, monads, higher-order functors, equational systems, etc.

These allow us to ensure that a user-defined monad to be used with *eval* must satisfy the equations associated to the operations. However, what is challenging is adding equalities from the algebraic theory of effectful operations to the computations without breaking the canonicity of the type theory. This difficulty is the same as that of adding *quotient types* to types theories without breaking canonicity, which is possible in *observational type theory* [Pujet and Tabareau 2022] and *cubical type theory* [Coquand et al. 2018]. Apart from the difficulty with quotients, having general recursion, impredicative polymorphism, and dependent types all together is not straightforward either because the category of well complete objects that we used in Section 5.6 as predomains is unlikely locally cartesian closed, so we need to find another category of predomains.

* * *

6∗8. Thank you for getting this far into the thesis! I hope you enjoyed (at least some part of) it even though the thesis was written in a rather personal way, exploring the author's point of view on higher-order algebraic effects, using categorical and logical tools liberally, while not trying very hard (if at all) to explain the background knowledge or convince the reader that the theory developed here is immediately useful. By the standard of PLT top conferences nowadays, the material of this thesis is almost unpublishable, yet I decided to write the thesis in this manner for the following reasons.

Firstly, the writing of thesis has convinced me that profound scientific work is more likely done if immediate applications (i.e. immediate publications) are not constantly the top priority. There were no technical obstacles for me to write everything in this thesis two years ago, but it is only when I was liberated from the 25-page `acmart` format, all the ideas naturally flowed into my mind. I don't mean that these ideas are necessarily significant, but I do believe that the way of writing it is more likely to lead us to bigger intellectual enrichment and scientific discoveries than the current practice of our community.

Secondly, faced with the reality I know that this thesis may become the tombstone of my academic life, so this thesis may be a now-or-never opportunity for me to write something that I'm actually interested in. Still, I hope the thesis will be of some value to future researchers interested in computational effects.

Bibliography

Samson Abramsky and Achim Jung. *Domain Theory*, page 1–168. Oxford University PressOxford, 1995. ISBN 9781383026108. doi:10.1093/oso/9780198537625.003.0001. URL https://www.cs.bham.ac.uk/~axj/pub/papers/handy1.pdf.

J. Adamek, J. Rosicky, E. M. Vitale, and F. W. Lawvere. *Algebraic Theories*. Cambridge University Press, Cambridge, 2010. doi:10.1017/CBO9780511760754. URL http://ebooks.cambridge.org/ref/id/CBO9780511760754.

Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 015(4):589–602, 1974. URL http://eudml.org/doc/16649.

Jiří Adámek and Jiří Rosicky. *Locally Presentable and Accessible Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1994. doi:10.1017/CBO9780511600579.

Alejandro Aguirre, Shin-ya Katsumata, and Satoshi Kura. Weakest preconditions in fibrations. *Mathematical Structures in Computer Science*, 32(4):472–510, 2022. doi:10.1017/S0960129522000330.

Amal Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In Peter Sestoft, editor, *Programming Languages and Systems*, pages 69–83. Springer Berlin Heidelberg, 2006.

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, page 18–29, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2837614.2837638.

Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In David Pitt, David E. Rydeheard, and Peter Johnstone, editors, *Category Theory and Computer Science*, pages 182–199. Springer Berlin Heidelberg, 1995.

Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification*, PLPV '07, page 57–68. Association for Computing Machinery, 2007. doi:10.1145/1292597.1292608.

Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. Quotient inductive-inductive types. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, pages 293–310. Springer International Publishing, 2018.

Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. *Setoid Type Theory – A Syntactic Translation*, page 155–196. Springer International Publishing, 2019. doi:10.1007/978-3-030-33636-3_7.

Carlo Angiuli and Daniel Gratzer. Principles of dependent type theory. Draft, 2024. URL `https://www.danielgratzer.com/papers/type-theory-book.pdf`.

M. A. Arbib and E. G. Manes. A categorist's view of automata and systems. In Ernest Gene Manes, editor, *Category Theory Applied to Computation and Control*, pages 51–64, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

Nathanael Arkor. *Monadic and Higher-Order Structure*. PhD thesis, University of Cambridge, 2022.

M. Artin, A. Grothendieck, and J. L. Verdier. *Théorie des Topos et Cohomologie Etale des Schémas (SGA4)*, volume 269 of *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1972. doi:10.1007/BFb0081551.

Andrea Asperti and Simone Martini. Categorical models of polymorphism. *Information and Computation*, 99(1):1–79, 1992. doi:10.1016/0890-5401(92)90024-A.

Robert Atkey. Parameterised notions of computation. *Journal of Functional Programming*, 19(3-4):335–376, 2009. doi:10.1017/S095679680900728X.

Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, page 503–515. Association for Computing Machinery, 2014. doi:10.1145/2535838.2535852.

Steve Awodey. Natural models of homotopy type theory. *Mathematical Structures in Computer Science*, 28(2):241–286, 2018. doi:10.1017/S0960129516000268.

Steve Awodey, Jonas Frey, and Sam Speight. Impredicative encodings of (higher) inductive types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on*

Logic in Computer Science, LICS '18, page 76–85. Association for Computing Machinery, 2018. doi:10.1145/3209108.3209130.

E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70(1):35–64, 1990. doi:10.1016/0304-3975(90)90151-7.

Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Behavioral metrics via functor lifting. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, volume 29 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 403–415, Dagstuhl, Germany, 2014. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2014.403.

Andrej Bauer and Matija Pretnar. An effect system for algebraic effects and handlers. *Logical Methods in Computer Science*, Volume 10, Issue 4, 2014. doi:10.2168/LMCS-10(4:9)2014.

Jean Bénabou. *Problèmes dans les topos: d'après le cours de Questions spéciales de mathématique: rapport no 34, mars 1973, Seminaires de mathématique pure*. Université catholique de Louvain, 1973.

Jean Bénabou. Fibrations petites et localement petites. *C. R. Acad. Sci. Paris*, 281: 897–900, 1975. URL `https://gallica.bnf.fr/ark:/12148/bpt6k6228235m/f171.image`.

R. Blackwell, G.M. Kelly, and John Power. Two-dimensional monad theory. *Journal of Pure and Applied Algebra*, 59(1):1–41, 1989. ISSN 00224049. doi:10.1016/0022-4049(89)90160-6.

William W. Boone. The word problem. *Proceedings of the National Academy of Sciences*, 44(10):1061–1065, 1958. doi:10.1073/pnas.44.10.1061.

Francis Borceux. *Handbook of Categorical Algebra: Volume 1, Basic Category Theory*, volume 1. Cambridge University Press, 1994a.

Francis Borceux. *Handbook of Categorical Algebra: Volume 2, Categories and Structures*, volume 2. Cambridge University Press, 1994b.

Francis Borceux. *Handbook of Categorical Algebra: Volume 3, Sheaf Theory*, volume 3. Cambridge University Press, 1994c.

Aurelio Carboni and Peter Johnstone. Connected limits, familial representability and Artin glueing. *Mathematical Structures in Computer Science*, 5(4):441–459, 1995. doi:10.1017/S0960129500001183.

John Cartmell. *Generalised Algebraic Theories and Contextual Categories*. PhD thesis, University of Oxford, 1978. URL `https://ncatlab.org/nlab/files/Cartmell-Thesis.pdf`.

John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986. doi:10.1016/0168-0072(86)90053-9.

Luca Castellano, Giorgio De Michelis, and Lucia Pomello. Concurrency vs interleaving: An instructive example. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 31, 1987.

Pietro Cenciarelli and Eugenio Moggi. A syntactic approach to modularity in denotational semantics. Technical report, In Proceedings of the Conference on Category Theory and Computer Science, 1993.

Paul M. Cohn. *Universal Algebra*. Mathematics and Its Applications. Springer Dordrecht, 1981. doi:10.1007/978-94-009-8399-1.

Thierry Coquand. An analysis of Girard's paradox. In *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS 1986)*, pages 227–236. IEEE Computer Society Press, 1986.

Thierry Coquand. Reduction free normalisation for a proof irrelevant type of propositions. *Logical Methods in Computer Science*, Volume 19, Issue 3, 2023. doi:10.46298/lmcs-19(3:5)2023.

Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2):95–120, 1988. doi:10.1016/0890-5401(88)90005-3.

Thierry Coquand, Carl Gunter, and Glynn Winskel. Domain theoretic models of polymorphism. *Information and Computation*, 81(2):123–167, 1989. doi:10.1016/0890-5401(89)90068-0.

Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 255–264, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355834. doi:10.1145/3209108.3209197.

Roy L. Crole. *Categories for Types*. Cambridge University Press, 1994.

Brian Day. On closed categories of functors. In S. Mac Lane, H. Applegate, M. Barr, B. Day, E. Dubuc, Phreilambud, A. Pultr, R. Street, M. Tierney, and S. Swierczkowski, editors, *Reports of the Midwest Category Seminar IV*, pages 1–38, Berlin, Heidelberg, 1970. Springer Berlin Heidelberg.

Menno de Boer. A proof and formalization of the initiality conjecture of dependent type theory, 2020. Licentiate defense over Zoom.

Marcelo Fiore. Discrete generalised polynomial functors, 2012. URL https://www.cl.cam.ac.uk/~mpf23/talks/ICALP2012.pdf. Talk given at ICALP 2012.

Marcelo Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus, 2022. doi:10.48550/arXiv.2207.08777.

Marcelo Fiore and Chung-Kil Hur. Equational systems and free constructions. In *Proceedings of the 34th International Conference on Automata, Languages and Programming*, ICALP'07, page 607–618, Berlin, Heidelberg, 2007. Springer-Verlag. doi:10.1007/978-3-540-73420-8_53.

Marcelo Fiore and Chung-Kil Hur. On the construction of free algebras for equational systems. *Theoretical Computer Science*, 410(18):1704–1729, 2009. doi:10.1016/j.tcs.2008.12.052.

Marcelo Fiore and Chung-Kil Hur. Second-order equational logic. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic*, pages 320–335, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

Marcelo Fiore and Ola Mahmoud. Second-order algebraic theories. In Petr Hliněný and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2010*, pages 368–380, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

Marcelo Fiore and Ola Mahmoud. Functorial semantics of second-order algebraic theories, 2014.

Marcelo Fiore and Gordon Plotkin. An extension of models of Axiomatic Domain Theory to models of Synthetic Domain Theory. In Gerhard Goos, Juris Hartmanis, Jan Leeuwen, Dirk Dalen, and Marc Bezem, editors, *Computer Science Logic*, volume 1258, pages 129–149. Springer Berlin Heidelberg, 1997. doi:10.1007/3-540-63172-0_36.

Marcelo Fiore and Giuseppe Rosolini. Two models of synthetic domain theory. *Journal of Pure and Applied Algebra*, 116(1–3):151–162, 1997. doi:10.1016/S0022-4049(96)00164-8.

Marcelo Fiore and Philip Saville. List objects with algebraic structure. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSCD.2017.16.

Marcelo Fiore and Sam Staton. Substitution, jumps, and algebraic effects. *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic, CSL 2014 and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2014*, 2014. doi:10.1145/2603088.2603163.

Marcelo Fiore and Dmitrij Szamozvancev. Formal metatheory of second-order abstract syntax. *Proc. ACM Program. Lang.*, 6(POPL), 2022. doi:10.1145/3498715.

Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proceedings. 14th Symposium on Logic in Computer Science*, pages 193–202, 1999. doi:10.1109/LICS.1999.782615.

Marcelo Fiore, Andrew M. Pitts, and S. C. Steenkamp. Quotients, inductive types, & quotient inductive types. *Logical Methods in Computer Science*, 18(2): 15:1–15:37, 2022. doi:10.46298/lmcs-18(2:15)2022.

Peter Freyd. On proving that 1 is an indecomposable projective in various free categories. *Manuscript*, 1978.

Neil Ghani, Tarmo Uustalu, and Makoto Hamana. Explicit substitutions and higher-order syntax. *Higher-Order and Symbolic Computation*, 2006. doi:10.1007/s10990-006-8748-4.

Jeremy Gibbons, Donnacha Oisín Kidney, Tom Schrijvers, and Nicolas Wu. Breadth-first traversal via staging. In Ekaterina Komendantskaya, editor, *Mathematics of Program Construction*, pages 1–33, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-16912-0_1.

Andrew Gill, John Launchbury, and Simon Peyton Jones. A short cut to deforestation. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, FPCA '93, page 223–232, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/165180.165214.

Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'État, Université Paris VII, 1972.

Jean-Yves Girard. The System F of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986. doi:10.1016/0304-3975(86)90044-7.

Jean-Yves Girard. *Proofs and types*. Cambridge tracts in theoretical computer science. Cambridge University Press, 1989.

Jean Giraud. Cohomologie non abélienne. *C. R. Acad. Sci. Paris*, 260:2666–2668, 1965.

J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial Algebra Semantics and Continuous Algebras. *Journal of the ACM*, 24(1):68–95, 1977.

Daniel Gratzer. *Syntax and semantics of modal type theory*. PhD thesis, Aarhus University, 2023. URL https://pure.au.dk/portal/en/publications/syntax-and-semantics-of-modal-type-theory.

Daniel Gratzer and Jonathan Sterling. Syntactic categories for dependent type theory: sketching and adequacy, 2021. URL https://arxiv.org/abs/2012.10783.

Daniel Gratzer, Michael Shulman, and Jonathan Sterling. Strict universes for Grothendieck topoi, 2022. URL https://arxiv.org/abs/2202.12012.

Philip Greenspun. The 10th rule of programming, 2003. URL https://philip.greenspun.com/bboard/q-and-a-fetch-msg?msg_id=000tgU&topic_id=22&topic=Ask+Philip. [Online; accessed 09-September-2024].

Harrison Grodin, Yue Niu, Jonathan Sterling, and Robert Harper. Decalf: A directed, effectful cost-aware logical framework. *Proceedings of the ACM on Programming Languages*, 8(POPL):273–301, 2024. doi:10.1145/3632852.

Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, Cambridge, 2nd edition, 2016.

Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993. doi:10.1145/138027.138060.

Ralf Hinze. Kan extensions for program optimisation or: Art and Dan explain an old trick. In Jeremy Gibbons and Pablo Nogueira, editors, *Mathematics of Program Construction*, pages 324–362, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-31113-0_16.

Ralf Hinze and Ross Paterson. Finger trees: a simple general-purpose data structure. *Journal of Functional Programming*, 16(02):197, 2005. ISSN 1469-7653. doi:10.1017/s0956796805005769.

Ralf Hinze, Thomas Harper, and Daniel W. H. James. Theory and practice of fusion. In Jurriaan Hage and Marco T. Morazán, editors, *Implementation and Application of Functional Languages*, pages 19–37, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-24276-2_2.

Martin Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, University of Edinburgh, 1995a. URL https://era.ed.ac.uk/handle/1842/399.

Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. *Lecture Notes in Computer Science*, 933:427–441, 1995b. doi:10.1007/bfb0022273.

Martin Hofmann. Syntax and semantics of dependent types. In Andrew M. Pitts and P.Editors Dybjer, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, page 79–130. Cambridge University Press, 1997. doi:10.1017/CBO9780511526619.004.

Martin Hofmann and Thomas Streicher. *The groupoid interpretation of type theory*. Oxford University Press, 1998. doi:10.1093/oso/9780198501275.003.0008.

Martin Hofmann and Thomas Streicher. Lifting Grothendieck universes. Unpublished note, 1999. URL `https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf`.

Kuen-Bang Hou (Favonia). *Higher-Dimensional Types in the Mechanization of Homotopy Theory*. PhD thesis, Carnegie Mellon University, 2017. URL `https://favonia.org/thesis`.

John Hughes. A novel representation of lists and its application to the function "reverse". *Inf. Process. Lett.*, 22:141–144, 1986.

John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1):67–111, 2000. doi:10.1016/S0167-6423(99)00023-4.

J.M.E. Hyland. *The Effective Topos*, volume 110, page 165–216. Elsevier, 1982. doi:10.1016/S0049-237X(09)70129-6.

J.M.E. Hyland. A small complete category. *Annals of Pure and Applied Logic*, 40(2): 135–165, 1988. doi:10.1016/0168-0072(88)90018-8.

J.M.E. Hyland. First steps in synthetic domain theory. In Aurelio Carboni, Maria Cristina Pedicchio, and Guiseppe Rosolini, editors, *Category Theory*, volume 1488, pages 131–156. Springer Berlin Heidelberg, 1991. doi:10.1007/BFb0084217.

J.M.E. Hyland, Gordon Plotkin, and John Power. Combining effects: Sum and tensor. *Theoretical Computer Science*, 357(1):70–99, 2006. doi:10.1016/j.tcs.2006.03.013.

Martin Hyland. Classical lambda calculus in modern dress. *Mathematical Structures in Computer Science*, 27(5):762–781, 2017. doi:10.1017/S0960129515000377.

Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.

Bart Jacobs, Chris Heunen, and Ichiro Hasuo. Categorical semantics for arrows. *Journal of Functional Programming*, 19(3-4):403–438, 2009. doi:10.1017/S0956796809007308.

Mauro Jaskelioff. Modular monad transformers. In Giuseppe Castagna, editor, *Programming Languages and Systems*, pages 64–79, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-00590-9_6.

Mauro Jaskelioff and Eugenio Moggi. Monad transformers as monoid transformers. *Theoretical Computer Science*, 411:4441–4466, 2010. doi:10.1016/j.tcs.2010.09.011.

Mamuka Jibladze. A presentation of the initial lift-algebra. *Journal of Pure and Applied Algebra*, 116(1–3):185–198, 1997. doi:10.1016/S0022-4049(96)00108-9.

Niles Johnson and Donald Yau. 2-dimensional categories, 2020. URL `https://arxiv.org/abs/2002.06055`.

Peter Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Oxford Logic Guides. Oxford University Press, 2002.

Achim Jung and Jerzy Tiuryn. A new characterization of lambda definability. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257. Springer-Verlag, 1993. doi:10.1007/BFb0037110.

Ohad Kammar and Gordon Plotkin. Algebraic foundations for effect-dependent optimisations. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 349–360. ACM, 2012. doi:10.1145/2103656.2103698.

Ohad Kammar, Paul Blain Levy, Sean K. Moss, and Sam Staton. A monad for full ground reference cells. *Proceedings - Symposium on Logic in Computer Science*, 2017. doi:10.1109/LICS.2017.8005109.

Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL), 2019. doi:10.1145/3290315.

Ambrus Kaposi, András Kovács, and Ambroise Lafont. For finitary induction-induction, induction is enough. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs (TYPES 2019)*, volume 175 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:30. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.TYPES.2019.6.

Chris Kapulkin and Yufeng Li. Extensional concepts in intensional type theory, revisited, 2023. doi:https://doi.org/10.48550/arXiv.2310.05706.

Krzysztof Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of Univalent Foundations (after Voevodsky). *Journal of the European Mathematical Society*, 23(6):2071–2126, 2021. doi:10.4171/jems/1050.

Shin-ya Katsumata. *A Semantic Formulation of* `TT`*-Lifting and Logical Predicates for Computational Metalanguage*, volume 3634 of *Lecture Notes in Computer Science*, page 87–102. Springer Berlin Heidelberg, 2005. doi:10.1007/11538363_8.

Shin-ya Katsumata. Parametric effect monads and semantics of effect systems. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, page 633–645, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2535838.2535846.

Shin-ya Katsumata, Tetsuya Sato, and Tarmo Uustalu. Codensity lifting of monads and its dual. *Logical Methods in Computer Science*, Volume 14, Issue 4, 2018. doi:10.23638/LMCS-14(4:6)2018. URL `https://lmcs.episciences.org/4924`.

Shin-ya Katsumata, Dylan McDermott, Tarmo Uustalu, and Nicolas Wu. Flexible presentations of graded monads. *Proc. ACM Program. Lang.*, 6(ICFP), 2022. doi:10.1145/3547654.

G. M. Kelly. Structures defined by finite limits in the enriched context, i. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 23(1):3–42, 1982.

G.M. Kelly and John Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *Journal of Pure and Applied Algebra*, 89(1):163–179, 1993. doi:10.1016/0022-4049(93)90092-8.

Donnacha Oisín Kidney and Nicolas Wu. Algebras for weighted search. *Proc. ACM Program. Lang.*, 5(ICFP), 2021. doi:10.1145/3473577.

Anders Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23: 113–120, 1972. doi:10.1007/BF01304852.

András Kovács. *Type-theoretic signatures for algebraic theories and inductive types*. PhD thesis, Eötvös Loránd University, 2023. URL `https://arxiv.org/abs/2302.08837`.

J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*. Cambridge University Press, 1986.

Joachim Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170, 1958. URL `http://www.jstor.org/stable/2310058`.

Joachim Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968. doi:10.1007/BF01110627.

Søren Bøgh Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Aarhus University, 1998. URL `https://www.brics.dk/DS/98/2/`. Series: BRICS Dissertation Series.

F. William Lawvere. *Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963. URL `http://www.tac.mta.ca/tac/reprints/articles/5/tr5abs.html`.

Daan Leijen. Koka: Programming with row polymorphic effect types. *Electronic Proceedings in Theoretical Computer Science*, 153:100–126, 2014. doi:10.4204/eptcs.153.8.

Tom Leinster. *Higher Operads, Higher Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2004. Available at `https://arxiv.org/abs/math/0305049`.

Paul Blain Levy. Possible world semantics for general storage in call-by-value. In Julian Bradfield, editor, *Computer Science Logic*, volume 2471 of *Lecture Notes in Computer Science*, page 232–246, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45793-3_16.

Paul Blain Levy. *Call-By-Push-Value*. Springer Netherlands, 2003. doi:10.1007/978-94-007-0954-6.

Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Information and Computation*, 185(2): 182–210, 2003. doi:10.1016/S0890-5401(03)00088-9.

Sheng Liang, Paul Hudak, and Mark Jones. Monad transformers and modular interpreters. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 333–343. ACM, 1995. doi:10.1145/199448.199528.

Sam Lindley and James Cheney. Row-based effect types for database integration. In *Proceedings of the 8th ACM SIGPLAN Workshop on Types in Language Design and Implementation*, TLDI '12, page 91–102. Association for Computing Machinery, 2012. doi:10.1145/2103786.2103798.

Sam Lindley and Ian Stark. Reducibility and `TT`-lifting for computation types. In Paweł Urzyczyn, editor, *Typed Lambda Calculi and Applications*, volume 3461 of *Lecture Notes in Computer Science*, page 262–277. Springer Berlin Heidelberg, 2005. doi:10.1007/11417170_20.

Sam Lindley, Philip Wadler, and Jeremy Yallop. Idioms are oblivious, arrows are meticulous, monads are promiscuous. *Electronic Notes in Theoretical Computer Science*, 229(5):97–117, 2011. ISSN 1571-0661. doi:10.1016/j.entcs.2011.02.018.

Sam Lindley, Cristina Matache, Sean Moss, Sam Staton, Nicolas Wu, and Zhixuan Yang. *Scoped Effects as Parameterized Algebraic Theories*, page 3–21. Springer Nature Switzerland, 2024. doi:10.1007/978-3-031-57262-3_1.

F. E. J. Linton. Some aspects of equational categories. In S. Eilenberg, D. K. Harrison, S. Mac Lane, and H. Röhrl, editors, *Proceedings of the Conference on Categorical Algebra*, pages 84–94. Springer Berlin Heidelberg, 1966. doi:10.1007/978-3-642-99902-4_3.

John R. Longley. *Realizability Toposes and Language Semantics*. PhD thesis, University of Edinburgh, 1995. URL `https://era.ed.ac.uk/handle/1842/402`.

John R. Longley and Alex K. Simpson. A uniform approach to domain theory in realizability models. *Mathematical Structures in Computer Science*, 7(5):469–505, 1997. doi:10.1017/S0960129597002387.

Fosco Loregian. *(Co)end Calculus*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021. doi:10.1017/9781108778657.

J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '88, page 47–57. Association for Computing Machinery, 1988. doi:10.1145/73560.73564.

Peter Lefanu Lumsdaine and Michael A. Warren. The local universes model: An overlooked coherence construction for dependent type theories. *ACM Trans. Comput. Logic*, 16(3), 2015. doi:10.1145/2754931.

Zhaohui Luo. *Computation and reasoning: a type theory for computer science*. Oxford University Press, 1994.

Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer, Berlin, 2nd edition, 1998.

Saunders MacLane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40 – 106, 1965. doi:bams/1183526392.

Maria Emilia Maietti. Modular correspondence between dependent type theories and categories including pretopoi and topoi. *Mathematical Structures in Computer Science*, 15(6):1089–1149, 2005. doi:10.1017/S0960129505004962.

Martin Markl. Operads and PROPs, 2006. URL https://arxiv.org/abs/math/0601129.

Jean-Pierre Marquis and Gonzalo Reyes. The history of categorical logic: 1963-1977. In Dov M. Gabbay, John Woods, and Akihiro Kanamori, editors, *Handbook of the history of logic*. Elsevier, 2004.

Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium 73 Proceedings of the Logic Colloquium*, pages 73–118. Elsevier, 1975a.

Per Martin-Löf. About models for intuitionistic type theories and the notion of definitional equality. In *Studies in Logic and the Foundations of Mathematics*, volume 82, pages 81–109. Elsevier, 1975b.

Per Martin-Löf. *Intuitionistic type theory*, volume 6. Bibliopolis Naples, 1984.

Christian Maurer. Universes in topoi. In F. William Lawvere, Christian Maurer, and Gavin C. Wraith, editors, *Model Theory and Topoi*, pages 284–296. Springer Berlin Heidelberg, 1975. doi:10.1007/BFb0061298.

Conor McBride. Turing-completeness totally free. In Ralf Hinze and Janis Voigtländer, editors, *Mathematics of Program Construction*, pages 257–275. Springer International Publishing, 2015. doi:10.1007/978-3-319-19797-5_13.

Conor McBride and Ross Paterson. Applicative programming with effects. *Journal of Functional Programming*, 18(1):1–13, 2008. doi:10.1017/S0956796807006326.

Dylan McDermott and Tarmo Uustalu. Flexibly graded monads and graded algebras. In Ekaterina Komendantskaya, editor, *Mathematics of Program Construction*, pages 102–128, Cham, 2022a. Springer International Publishing. doi:10.1007/978-3-031-16912-0_4.

Dylan McDermott and Tarmo Uustalu. What makes a strong monad? *Electronic Proceedings in Theoretical Computer Science*, 360:113–133, 2022b. doi:10.4204/EPTCS.360.6.

Eugenio Moggi. Interpretation of second order lambda-calculus in categories. Unpublished, 1987.

Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, 1989a. doi:10.1109/LICS.1989.39155.

Eugenio Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Edinburgh University, Department of Computer Science, 1989b.

Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

Andrey Mokhov, Neil Mitchell, and Simon Peyton Jones. Build systems à la carte. *Proc. ACM Program. Lang.*, 2(ICFP), 2018. doi:10.1145/3236774.

Clive Newstead. *Algebraic Models of Dependent Type Theory*. PhD thesis, Carnegie Mellon University, 2018.

Yue Niu, Jonathan Sterling, Harrison Grodin, and Robert Harper. A cost-aware logical framework. *Proc. ACM Program. Lang.*, 6(POPL), 2022. doi:10.1145/3498670.

nLab. Cantor's theorem. `https://ncatlab.org/nlab/show/Cantor%27s+theorem`, 2024a. Revision 25.

nLab. complete small category. `https://ncatlab.org/nlab/show/complete+small+category`, 2024b. Revision 16.

nLab. concept with an attitude. `https://ncatlab.org/nlab/show/concept+with+an+attitude`, 2024. Revision 23.

nLab. free monad. `https://ncatlab.org/nlab/show/free+monad`, 2024a. Revision 20.

nLab. mate. `https://ncatlab.org/nlab/show/mate`, 2024b. Revision 26.

nLab. monads of probability, measures, and valuations. `https://ncatlab.org/nlab/show/monads+of+probability%2C+measures%2C+and+valuations`, 2024c. Revision 45.

nLab. slice of presheaves is presheaves on slice. `https://ncatlab.org/nlab/show/slice+of+presheaves+is+presheaves+on+slice`, 2024d. Revision 20.

Bengt Nordström, Kent Petersson, and Jan M Smith. *Programming in Martin-Löf's type theory*, volume 200. Oxford University Press, 1990.

Ulf Norell. Dependently typed programming in Agda. In Pieter Koopman, Rinus Plasmeijer, and Doaitse Swierstra, editors, *Advanced Functional Programming: 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*, pages 230–266. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-04652-0_5.

M. Okada and P. J. Scott. A note on rewriting theory for uniqueness of iteration. *Theory and Applications of Categories*, 6:47–64, 2000. URL http://www.tac.mta.ca/tac/volumes/6/n4/6-04abs.html.

Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998. ISBN 9780511530104. doi:10.1017/cbo9780511530104.

Jaap van Oosten. *Realizability: an introduction to its categorical side*. Studies in logic and the foundations of mathematics. Elsevier, Oxford, 1st edition, 2008.

Dominic Orchard, Philip Wadler, and Harley Eades. Unifying graded and parameterised monads. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, 317(MSFP):18–38, 2020. doi:10.4204/EPTCS.317.2.

Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999. doi:10.2307/421090.

Ross Paterson. Constructing applicative functors. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7342 LNCS:300–323, 2012. doi:10.1007/978-3-642-31113-0_15.

Wesley Phoa. *Domain Theory in Realizability Toposes*. PhD thesis, University of Edinburgh, 1991. URL https://www.lfcs.inf.ed.ac.uk/reports/91/ECS-LFCS-91-171/.

Wesley Phoa. An introduction to fibrations, topos theory, the effective topos and modest sets. Technical Report ECS-LFCS-92-208, University of Edinburgh, 1992. URL https://www.lfcs.inf.ed.ac.uk/reports/92/ECS-LFCS-92-208/.

Ruben P. Pieters, Exequiel Rivas, and Tom Schrijvers. Generalized monoidal effects and handlers. *Journal of Functional Programming*, 30:e23, 2020. doi:10.1017/S0956796820000106.

Maciej Piróg. Eilenberg–Moore monoids and backtracking monad transformers. *Electronic Proceedings in Theoretical Computer Science*, 207:23–56, 2016. doi:10.4204/EPTCS.207.2.

Maciej Piróg, Tom Schrijvers, Nicolas Wu, and Mauro Jaskelioff. Syntax and semantics for operations with scopes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 809–818. Association for Computing Machinery, 2018. doi:10.1145/3209108.3209166.

Andrew M. Pitts. *Polymorphism is set theoretic, constructively*, page 12–39. Springer Berlin Heidelberg, 1987. doi:10.1007/3-540-18508-9_18.

Andrew M. Pitts. Categorical logic. In *Handbook of Logic in Computer Science: Volume 5. Algebraic and Logical Structures*. Oxford University Press, 2001. doi:10.1093/oso/9780198537816.003.0002.

Gordon Plotkin. Lambda definability and logical relations. Memorandum SAI-RM-4, University of Edinburgh, 1973. URL `https://homepages.inf.ed.ac.uk/gdp/publications/logical_relations_1973.pdf`.

Gordon Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3): 452–487, 1976. doi:10.1137/0205035.

Gordon Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977. doi:10.1016/0304-3975(77)90044-5.

Gordon Plotkin. Lambda-definability in the full type hierarchy. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 363–373. Academic Press, 1980. URL `https://homepages.inf.ed.ac.uk/gdp/publications/Lambda_Definability.pdf`.

Gordon Plotkin and Martín Abadi. *A logic for parametric polymorphism*, page 361–375. Springer-Verlag, 1993. doi:10.1007/bfb0037118.

Gordon Plotkin and John Power. Semantics for algebraic operations. *Electronic Notes in Theoretical Computer Science*, 45:332–345, 2001a. doi:10.1016/S1571-0661(04)80970-8.

Gordon Plotkin and John Power. Semantics for algebraic operations. *Electronic Notes in Theoretical Computer Science*, 45:332–345, 2001b. doi:10.1016/S1571-0661(04)80970-8.

Gordon Plotkin and John Power. Notions of computation determine monads. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference*, FOSSACS 2002, pages 342–356. Springer, 2002. doi:10.1007/3-540-45931-6_24.

Gordon Plotkin and John Power. Algebraic Operations and Generic Effects. *Applied Categorical Structures*, 11(1):69–94, 2003. doi:10.1023/A:1023064908962.

Gordon Plotkin and John Power. Computational effects and operations: An overview. *Electronic Notes in Theoretical Computer Science*, 73:149–163, 2004. doi:10.1016/j.entcs.2004.08.008.

Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. In Giuseppe Castagna, editor, *Programming Languages and Systems*, pages 80–94. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-00590-9_7.

Gordon Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/lmcs-9(4:23)2013.

John Power. Enriched lawvere theories. *Theory and Applications of Categories*, 6(7): 83–93, 1999.

Loïc Pujet and Nicolas Tabareau. Observational equality: now for good. *Proc. ACM Program. Lang.*, 6(POPL), 2022. doi:10.1145/3498693.

Bernhard Reus. *Program verification in synthetic domain theory*. PhD thesis, Ludwig Maximilian University of Munich, 1996. URL `https://www2.mathematik.tu-darmstadt.de/~streicher/THESES/reus.pdf`.

Bernhard Reus. Formalizing Synthetic Domain Theory. *Journal of Automated Reasoning*, 23(3):411–444, 1999. doi:10.1023/A:1006258506401.

Bernhard Reus and Thomas Streicher. General synthetic domain theory – a logical approach. *Mathematical Structures in Computer Science*, 9(2):177–223, 1999. doi:10.1017/S096012959900273X.

John C. Reynolds. Towards a theory of type structure. In Bernard Robinet, editor, *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974. doi:10.1007/3-540-06859-7_148.

John C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, 1983.

Exequiel Rivas and Mauro Jaskelioff. Notions of computation as monoids. *Journal of Functional Programming*, 27(September), 2017. doi:10.1017/S0956796817000132.

Edmund Robinson. Variations on algebra: Monadicity and generalisations of equational theories. 13(3):308–326, 2002. doi:10.1007/s001650200014.

Mario Román. Promonads and string diagrams for effectful categories. In *ACT '22: Applied Category Theory, Glasgow, United Kingdom, 18–22 July, 2022*, 2022. doi:10.48550/arXiv.2205.07664.

Giuseppe Rosolini. *Continuity and effectiveness in topoi*. PhD thesis, University of Oxford, 1986.

Tom Schrijvers, Maciej Piróg, Nicolas Wu, and Mauro Jaskelioff. Monad transformers and modular algebraic effects: What binds them together. In *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell,*

Haskell@ICFP 2019, Berlin, Germany, August 18-23, 2019, pages 98–113, 2019. doi:10.1145/3331545.3342595.

Dana Scott. Continuous lattices. In F. W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, pages 97–136, Berlin, Heidelberg, 1972. Springer Berlin Heidelberg. doi:10.1007/BFb0073967.

Dana Scott. Data Types as Lattices. *SIAM Journal on Computing*, 5(3):522–587, 1976. doi:10.1137/0205037.

Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121(1):411–440, 1993. doi:10.1016/0304-3975(93)90095-B.

R. A.G. Seely. Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society*, 95(1):33–48, 1984. doi:10.1017/S0305004100061284.

Michael A. Shulman. Set theory for category theory, 2008. URL `https://arxiv.org/abs/0810.1279`.

Alex Simpson. Computational adequacy for recursive types in models of intuitionistic set theory. *Annals of Pure and Applied Logic*, 130(1-3):207–275, 2004. doi:10.1016/j.apal.2003.12.005.

Alex K. Simpson. Computational Adequacy in an Elementary Topos. In *Computer Science Logic*, volume 1584, pages 323–342. Springer Berlin Heidelberg, 1999. doi:10.1007/10703163_22. Series Title: Lecture Notes in Computer Science.

Michael B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16(1): 23–36, 1978. doi:10.1016/0022-0000(78)90048-X.

Ian Stark. Free-algebra models for the π-calculus. *Theoretical Computer Science*, 390(2):248–270, 2008. doi:10.1016/j.tcs.2007.09.024. Foundations of Software Science and Computational Structures.

R. Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65(2):85–97, 1985. doi:10.1016/S0019-9958(85)80001-2.

Sam Staton. Completeness for algebraic theories of local state. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6014 LNCS:48–63, 2010. doi:10.1007/978-3-642-12032-9_5.

Jonathan Sterling. Algebraic type theory and universe hierarchies, 2019. doi:10.48550/arXiv.1902.08848.

Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. PhD thesis, Carnegie Mellon University, 2021. Version 1.1, revised May 2022.

Jonathan Sterling. Adequacy of sheaf semantics of noninterference, 2023. URL `https://www.jonmsterling.com/jms-005Z.xml`. Erratum.

Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. *Proceedings - Symposium on Logic in Computer Science*, 2021-June:1–22, 2021. doi:10.1109/LICS52264.2021.9470719. arXiv: 2101.11479.

Jonathan Sterling and Robert Harper. Logical relations as types: Proof-relevant parametricity for program modules. *Journal of the ACM*, 68(6), 2021. doi:10.1145/3474834.

Jonathan Sterling and Robert Harper. Sheaf semantics of termination-insensitive noninterference. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FSCD.2022.5.

Jonathan Sterling and Bas Spitters. Normalization by gluing for free λ-theories, 2018. doi:10.48550/arXiv.1809.08646.

Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. Denotational semantics of general store and polymorphism, 2023. URL `https://arxiv.org/abs/2210.02169`.

Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. Towards univalent reference types: The impact of univalence on denotational semantics. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:21, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.47.

Thomas Streicher. *Semantics of Type Theory: Correctness, Completeness and Independence Results*. Birkhäuser Boston, Boston, MA, 1991. doi:10.1007/978-1-4612-0433-6_2.

Thomas Streicher. Universes in toposes. *From Sets and Types to Topology and Analysis: Towards practical foundations for constructive mathematics.*, 48, 2005. doi:10.1093/acprof:oso/9780198566519.001.0001.

Thomas Streicher. *Domain-theoretic foundations of functional programming*. World Scientific Publishing Company, 2006.

Thomas Streicher. Fibered categories a la Jean Benabou, 2023. URL https://arxiv.org/abs/1801.02927.

William W. Tait. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 32(2):198–212, 1967. URL http://www.jstor.org/stable/2271658.

Robert D Tennent. *Semantics of programming languages*, volume 1. Prentice Hall New York, 1991.

David A Turner. Total functional programming. *J. Univers. Comput. Sci.*, 10(7): 751–768, 2004. doi:10.3217/JUCS-010-07-0751.

Taichi Uemura. *Abstract and concrete type theories*. PhD thesis, University of Amsterdam, 2021. URL https://dare.uva.nl/search?identifier=41ff0b60-64d4-4003-8182-c244a9afab3b.

Taichi Uemura. A general framework for the semantics of type theory. *Mathematical Structures in Computer Science*, 33(3):134–179, 2023. doi:10.1017/s0960129523000208.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, 2013.

Birthe van den Berg and Tom Schrijvers. A framework for higher-order effects & handlers. *Sci. Comput. Program.*, 234(C), 2024. doi:10.1016/j.scico.2024.103086.

Vladimir Voevodsky. A C-system defined by a universe category. *Theory and Applications of Categories*, 30:1181–1214, 2015.

Vladimir Voevodsky. The (Π, λ)-structures on the C-systems defined by universe categories. *Theory and Applications of Categories*, 32:113–121, 2017.

Philip Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional programming languages and computer architecture - FPCA '89*, volume 19, page 347–359. ACM Press, 1989. doi:10.1145/99370.99404.

Glynn Winskel. On powerdomains and modality. *Theoretical Computer Science*, 36:127–137, 1985. doi:10.1016/0304-3975(85)90037-4.

Nicolas Wu, Tom Schrijvers, and Ralf Hinze. Effect handlers in scope. *Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell - Haskell '14*, pages 1–12, 2014. doi:10.1145/2633357.2633358.

Zhixuan Yang. Efficient free applicatives from Okasaki's functional data structures, 2022. URL https://gist.github.com/yangzhixuan/07c2b3abf676b797cdfa0e77aa7700e8.

Zhixuan Yang and Nicolas Wu. Reasoning about effect interaction by fusion. *Proc. ACM Program. Lang.*, 5(ICFP), 2021. doi:10.1145/3473578.

Zhixuan Yang and Nicolas Wu. Modular models of monoids with operations. *Proc. ACM Program. Lang.*, 7(ICFP), 2023. doi:10.1145/3607850.

Zhixuan Yang, Marco Paviotti, Nicolas Wu, Birthe van den Berg, and Tom Schrijvers. *Structured Handling of Scoped Effects*, page 462–491. Springer International Publishing, 2022. doi:10.1007/978-3-030-99336-8_17.

Appendix A

Complete Signatures of Languages

This appendix contains the full signatures of the languages in this thesis.

A.1 Signature of System F^{ha}_{ω}

The following is the signature of System F^{ha}_{ω} from Section 5.1.

* Kinds

$$
\begin{aligned}
&ki && : \mathbb{J} \\
&el && : ki \to \mathbb{J} \\
&ty && : ki \\
&\_\Rightarrow_k\_ && : ki \to ki \to ki
\end{aligned}
$$

* Elements of the kind of types

$$
\begin{aligned}
&unit && : el\ ty \\
&bool && : el\ ty \\
&\_\Rightarrow_t\_ && : el\ ty \to el\ ty \to el\ ty \\
&All && : (k : ki) \to (el\ k \to el\ ty) \to el\ ty
\end{aligned}
$$

* Elements of function kinds

$$
\begin{aligned}
&\textsc{record}\ A \cong B : \mathbb{J}\ \textsc{where} \\
&\quad fwd && : A \to B \\
&\quad bwd && : B \to A \\
&\quad \_ && : (a : A) \to bwd\ (fwd\ a) = a \\
&\quad \_ && : (b : B) \to fwd\ (bwd\ b) = b \\
&\Rightarrow_k\text{-}iso && : \{A, B : ki\} \to el\ (A \Rightarrow_k B) \cong (el\ A \to el\ B)
\end{aligned}
$$

* Terms

$$
\begin{aligned}
&tm && : el\ ty \to \mathbb{J} \\
&unit\text{-}iso && : tm\ unit \cong 1
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow_t\text{-}iso &&: \{A, B : el\ ty\} \to tm\ (A \Rightarrow_t B) \cong (tm\ A \to tm\ B)\\
&All\text{-}iso &&: \{k : \_\}\ \{A : \_\} \to tm\ (All\ k\ A) \cong ((\alpha : el\ k) \to tm\ (A\ \alpha))\\
&tt &&: tm\ bool\\
&ff &&: tm\ bool
\end{aligned}
$$

* Functors

$$
\begin{aligned}
&tyco : ki\\
&tyco = (ty \Rightarrow_k ty)\\
&fmap\text{-}ty : (F : el\ tyco) \to el\ ty\\
&fmap\text{-}ty\ F = All\ ty\ (\lambda\alpha.\ All\ ty\ (\lambda\beta.\ (\alpha \Rightarrow_t \beta) \Rightarrow_t (F\ \alpha \Rightarrow_t F\ \beta)))\\
&\textsc{record}\ RawFunctor : \mathbb{J}\ \textsc{where}\\
&\quad F_0 : el\ tyco\\
&\quad F_1 : tm\ (fmap\text{-}ty\ F_0)
\end{aligned}
$$

* Monads

$$
\begin{aligned}
&\textsc{record}\ RawMonad : \mathbb{J}\ \textsc{where}\\
&\quad M_0 : el\ tyco\\
&\quad ret : tm\ (All\ ty\ (\lambda\alpha.\ \alpha \Rightarrow_t M_0\ \alpha))\\
&\quad bind : tm\ (All\ ty\ (\lambda\alpha.\ All\ ty\ (\lambda\beta.\ M_0\ \alpha \Rightarrow_t (\alpha \Rightarrow_t M_0\ \beta) \Rightarrow_t M_0\ \beta)))
\end{aligned}
$$

* Higher-order functors

$$
\begin{aligned}
&htyco : ki\\
&htyco = tyco \Rightarrow_k tyco\\
&trans : (F, G : el\ tyco) \to el\ ty\\
&trans\ F\ G = All\ ty\ (\lambda\alpha.\ F\ \alpha \Rightarrow_t G\ \alpha)\\
&\textsc{record}\ RawHFunctor : \mathbb{J}\ \textsc{where}\\
&\quad H_0 : el\ htyco\\
&\quad hfmap : (F : RawFunctor) \to tm\ (fmap\text{-}ty\ (H_0\ (F\ .F_0)))\\
&\quad hmap : (F, G : RawFunctor) \to tm\ (trans\ (F\ .F_0)\ (G\ .F_0))\\
&\qquad\quad \to tm\ (trans\ (H_0\ (F\ .F_0))\ (H_0\ (G\ .F_0)))
\end{aligned}
$$

* Computations

$$
\begin{aligned}
&co &&: (H : RawHFunctor) \to (A : el\ ty) \to \mathbb{J}\\
&val &&: \{H, A\} \to tm\ A \to co\ H\ A\\
&let\text{-}in &&: \{H, A, B\} \to co\ H\ A \to (tm\ A \to co\ H\ B) \to co\ H\ B
\end{aligned}
$$

* Laws of computations

$$
\begin{aligned}
&\mathit{val\text{-}let} \quad : \{H, A, B\} \to (a : \mathit{tm}\ A) \to (k : \mathit{tm}\ A \to \mathit{co}\ H\ B) \\
&\qquad \to \mathit{let\text{-}in}\ (\mathit{val}\ a)\ k = k\ a \\
&\mathit{let\text{-}val} \quad : \{H, A\} \to (m : \mathit{co}\ H\ A) \to \mathit{let\text{-}in}\ m\ \mathit{val} = m \\
&\mathit{let\text{-}assoc} : \{H, A, B, C\} \to (m_1 : \mathit{co}\ H\ A) \\
&\qquad \to (m_2 : \mathit{tm}\ A \to \mathit{co}\ H\ B) \to (m_3 : \mathit{tm}\ B \to \mathit{co}\ H\ C) \\
&\qquad \to \mathit{let\text{-}in}\ (\mathit{let\text{-}in}\ m_1\ m_2)\ m_3 = \mathit{let\text{-}in}\ m_1\ (\lambda a.\ \mathit{let\text{-}in}\ (m_2\ a)\ m_3)
\end{aligned}
$$

* Thunks

$$
\begin{aligned}
&\mathit{th} : \mathit{RawHFunctor} \to \mathit{el}\ \mathit{ty} \to \mathit{el}\ \mathit{ty} \\
&\mathit{th\text{-}iso} : \{H, A\} \to \mathit{tm}\ (\mathit{th}\ H\ A) \cong \mathit{co}\ H\ A \\
&\Uparrow : \{H, A\} \to \mathit{tm}\ (\mathit{th}\ H\ A) \to \mathit{co}\ H\ A \\
&\Uparrow = \mathit{th\text{-}iso}\ .\mathit{fwd} \\
&\Downarrow : \{H, A\} \to \mathit{co}\ H\ A \to \mathit{tm}\ (\mathit{th}\ H\ A) \\
&\Downarrow = \mathit{th\text{-}iso}\ .\mathit{bwd} \\
&\mathit{th\text{-}mnd} : \mathit{RawHFunctor} \to \mathit{RawMonad} \\
&\mathit{th\text{-}mnd}\ H\ .M_0 \ = \mathit{th}\ H \\
&\mathit{th\text{-}mnd}\ H\ .\mathit{ret} \ = \lambda A\ x.\ \Downarrow (\mathit{val}\ x) \\
&\mathit{th\text{-}mnd}\ H\ .\mathit{bind} = \lambda A\ B\ m\ k.\ \Downarrow (\mathit{let\text{-}in}\ (\mathit{force}\ m)\ (\lambda a.\ \Uparrow (k\ a)))
\end{aligned}
$$

* Operations

$$
\begin{aligned}
&\mathit{op} : \{H, A, B\} \to \mathit{tm}\ (H\ (\mathit{th}\ H)\ A) \to (\mathit{tm}\ A \to \mathit{co}\ H\ B) \to \mathit{co}\ H\ B \\
&\mathit{let\text{-}op} : \{H, A, B, C\} \to (p : \mathit{tm}\ (H\ (\mathit{th}\ H)\ A)) \\
&\qquad \to (k : \mathit{tm}\ A \to \mathit{co}\ H\ B) \to (k' : \mathit{tm}\ B \to \mathit{co}\ H\ C) \\
&\qquad \to \mathit{let\text{-}in}\ (\mathit{op}\ p\ k)\ k' = \mathit{op}\ p\ (\lambda a.\ \mathit{let\text{-}in}\ (k\ a)\ k')
\end{aligned}
$$

* Monads with algebras

$$
\begin{aligned}
&\textsc{record}\ \mathit{MonadAlg}\ (H : \mathit{RawHFunctor}) : \mathbb{J}\ \textsc{where} \\
&\quad \textsc{include}\ \mathit{RawMonad}\ \textsc{as}\ M \\
&\quad \mathit{malg} : \mathit{tm}\ (\mathit{trans}\ (H\ .H_0\ M_0)\ M_0) \\
&\mathit{th\text{-}alg} : (H : \mathit{RawHFunctor}) \to \mathit{MonadAlg}\ H \\
&\mathit{th\text{-}alg}\ H\ .M \ \ = \mathit{th\text{-}mnd}\ H \\
&\mathit{th\text{-}alg}\ H\ .\mathit{malg} = \lambda \alpha\ o.\ \Downarrow (\mathit{op}\ o\ \mathit{val})
\end{aligned}
$$

* Evaluation of computations

$$
\begin{aligned}
&\mathit{eval} : \{H\} \to (m : \mathit{MonadAlg}\ H) \to (A : \mathit{el}\ \mathit{ty}) \to \mathit{co}\ H\ A \to \mathit{tm}\ (m\ .M_0\ A) \\
&\mathit{eval\text{-}val} : \{H, A\} \to (m : \mathit{MonadAlg}\ H) \to (a : \mathit{tm}\ A)
\end{aligned}
$$

$$\to \mathit{eval}\ m\ A\ (\mathit{val}\ a) = m\ .\mathit{ret}\ A\ a$$

$$\begin{aligned}
&\mathit{eval\text{-}op} : \{H, A, B\} \to (m : \mathit{MonadAlg}\ H) \\
&\qquad \to (p : \mathit{tm}\ (H\ (\mathit{th}\ H)\ A)) \to (k : \mathit{tm}\ A \to \mathit{co}\ H\ B) \\
&\qquad \to \textsc{let}\ \mathit{bind} = m\ .\mathit{bind}\ A\ B \\
&\qquad\qquad\ \ \mathit{malg} = m\ .\mathit{malg}\ A \\
&\qquad\qquad\ \ T \quad = \mathit{fct\text{-}of\text{-}mnd}\ (\mathit{th\text{-}mnd}\ H) \\
&\qquad\qquad\ \ M \quad = \mathit{fct\text{-}of\text{-}mnd}\ (m\ .M) \\
&\qquad\ \ \textsc{in}\ \mathit{eval}\ m\ B\ (\mathit{op}\ p\ k) \\
&\qquad\qquad = \mathit{bind}\ (\mathit{malg}\ (H\ .\mathit{hmap}\ T\ M\ (\lambda\alpha\ c.\ \mathit{eval}\ m\ \alpha\ (\Uparrow c))\ A\ p)) \\
&\qquad\qquad\qquad (\lambda a.\ \mathit{eval}\ m\ B\ (k\ a))
\end{aligned}$$

$$\begin{aligned}
&\mathit{fct\text{-}of\text{-}mnd} : \mathit{RawMonad} \to \mathit{RawFunctor} \\
&\mathit{fct\text{-}of\text{-}mnd}\ m\ .F_0 = m\ .M_0 \\
&\mathit{fct\text{-}of\text{-}mnd}\ m\ .F_1\ \alpha\ \beta\ f\ ma = m\ .\mathit{bind}\ \alpha\ \beta\ ma\ (\lambda a.\ m\ .\mathit{ret}\ _\ (f\ a))
\end{aligned}$$

A.2 Effect Families in $\mathbf{F}^{\mathbf{ha}}_{\omega}$

The definition of effect families and modular handlers in 5.1.4∗3, together with the full definition of the effect family *algFam* in Example 5.1.4∗4 (including the required type connectives), is collected in this section for reference.

∗ Families

$$\begin{aligned}
&\textsc{record}\ \mathit{Fam} : \mathbb{J}\ \textsc{where} \\
&\quad \mathit{eff}\ : \mathit{ki} \\
&\quad \mathit{sig}\ : \mathit{el}\ \mathit{eff} \to \mathit{RawHFunctor} \\
&\quad \mathit{add} : \mathit{el}\ \mathit{eff} \to \mathit{el}\ \mathit{eff} \to \mathit{el}\ \mathit{eff}
\end{aligned}$$

$$\begin{aligned}
&\mathit{MonadEff} : (F : \mathit{Fam}) \to (e : \mathit{el}\ (F\ .\mathit{eff})) \to \mathbb{J} \\
&\mathit{MonadEff}\ F\ e = \mathit{MonadAlg}\ (F\ .\mathit{sig}\ e)
\end{aligned}$$

$$\begin{aligned}
&\mathit{co}[_\ni_] : (F : \mathit{Fam}) \to (e : \mathit{el}\ (F\ .\mathit{eff})) \to \mathit{el}\ \mathit{ty} \to \mathbb{J} \\
&\mathit{co}[F \ni e] = \mathit{co}\ (F\ .\mathit{sig}\ e)
\end{aligned}$$

∗ Modular handlers

$$\begin{aligned}
&\textsc{record}\ \mathit{Hdl}\ (F : \mathit{Fam})\ (e\ o : \mathit{el}\ (F\ .\mathit{eff})) : \mathbb{J}\ \textsc{where} \\
&\quad \mathit{alg}\ : (\mu : \mathit{el}\ (F\ .\mathit{eff})) \to \mathit{MonadEff}\ F\ (F\ .\mathit{add}\ o\ \mu) \\
&\qquad\ \to \mathit{MonadEff}\ F\ (F\ .\mathit{add}\ e\ \mu) \\
&\quad \mathit{res}\ : \mathit{el}\ (\mathit{ty} \Rightarrow_k \mathit{ty}) \\
&\quad \mathit{run} : (\mu : \mathit{el}\ (F\ .\mathit{eff})) \to (\mathit{Mo} : \mathit{MonadEff}\ F\ (F\ .\mathit{add}\ o\ \mu)) \\
&\qquad\ \to \mathit{tm}\ (\mathit{trans}\ (\mathit{alg}\ \mu\ \mathit{Mo}\ .M_0)\ (\lambda A.\ \mathit{Mo}\ .M_0\ (\mathit{res}\ A)))
\end{aligned}$$

$$\begin{aligned}
&\mathit{handle} : \{F, e, o, \mu, A\} \to (h : \mathit{Hdl}\ F\ e\ o) \to \mathit{co}[F \ni (F\ .\mathit{add}\ e\ \mu)]\ A \\
&\qquad \to \mathit{co}[F \ni (F\ .\mathit{add}\ o\ \mu)]\ (h\ .\mathit{res}\ A)
\end{aligned}$$

$$
\begin{aligned}
&\mathit{handle}\ h\ c = \Uparrow (h\ .\mathit{run}\ \mu\ T\ A\ c') \ \textsc{where}\\
&\quad T : \mathit{MonadEff}\ F\ (F\ .\mathit{add}\ o\ \mu)\\
&\quad T = \mathit{th\text{-}alg}\ (F\ .\mathit{sig}\ (F\ .\mathit{add}\ o\ \mu))\\
&\quad c' : \mathit{tm}\ (h\ .\mathit{alg}\ \mu\ T\ .M_0\ A)\\
&\quad c' = \mathit{eval}\ (h\ .\mathit{alg}\ \mu\ (\mathit{th\text{-}alg}\ (F\ .\mathit{sig}\ (F\ .\mathit{add}\ o\ \mu))))\ \_\ c
\end{aligned}
$$

* Kind-level and type-level products

$$
\begin{aligned}
&\_ \times_k \_ : \mathit{ki} \to \mathit{ki} \to \mathit{ki}\\
&\times_k\text{-}\mathit{iso} : \{k\ k' : \mathit{ki}\} \to \mathit{el}\ (k\ \times_k\ k') = \Sigma\ (\mathit{el}\ k)\ (\lambda\_.\ \mathit{el}\ k')\\
&\_ \times_t \_ : \mathit{el}\ \mathit{ty} \to \mathit{el}\ \mathit{ty} \to \mathit{el}\ \mathit{ty}\\
&\times_t\text{-}\mathit{iso} : \{A\ B : \mathit{el}\ \mathit{ty}\} \to \mathit{tm}\ (A\ \times_t\ B) = \Sigma\ (\mathit{tm}\ A)\ (\lambda\_.\ \mathit{tm}\ B)
\end{aligned}
$$

* The empty type

$$
\begin{aligned}
&\mathit{empty} : \mathit{el}\ \mathit{ty}\\
&\mathit{absurd} : (A : \mathit{el}\ \mathit{ty}) \to \mathit{tm}\ \mathit{empty} \to \mathit{tm}\ A\\
&\mathit{absurd\text{-}uniq} : \{A : \mathit{el}\ \mathit{ty}\} \to (f : \mathit{tm}\ \mathit{empty} \to \mathit{tm}\ A) \to f = \mathit{absurd}\ A
\end{aligned}
$$

* Coproducts

We only need type-level coproducts, but for generality we define coproducts parameterised by judgements $U : \mathbb{J}$ and $T : U \to \mathbb{J}$:

$$
\begin{aligned}
&\textsc{record}\ \mathit{coprod\_intro}\ (U : \mathbb{J})\ (T : U \to \mathbb{J}) : \mathbb{J}\ \textsc{where}\\
&\quad \_+\_ : U \to U \to U\\
&\quad \mathit{inl}\ \ : \{a, b\} \to T\ a \to T\ (a + b)\\
&\quad \mathit{inr}\ \ : \{a, b\} \to T\ b \to T\ (a + b)\\
&\textsc{record}\ \mathit{coprod\_elim}\ (U : \mathbb{J})\ (T : U \to \mathbb{J})\ (V : \mathbb{J})\ (S : U \to \mathbb{J})\\
&\quad (\mathit{intr} : \mathit{coprod\_intro}\ U\ T) : \mathbb{J}\ \textsc{where}\\
&\quad \textsc{open}\ \mathit{coprod\_intro}\ \mathit{intr}\\
&\quad \mathit{case} : \{a, b, c\} \to (T\ a \to S\ c) \to (T\ b \to S\ c) \to (T\ (a + b) \to S\ c)\\
&\quad \mathit{case\_\beta\_l} : \{a, b, c\} \to (l : T\ a \to S\ c) \to (r : T\ b \to S\ c) \to (x : T\ a)\\
&\qquad\qquad \to \mathit{case}\ l\ r\ (\mathit{inl}\ x) = l\ x\\
&\quad \mathit{case\_\beta\_r} : \{a, b, c\} \to (l : T\ a \to S\ c) \to (r : T\ b \to S\ c) \to (x : T\ b)\\
&\qquad\qquad \to \mathit{case}\ l\ r\ (\mathit{inr}\ x) = r\ x\\
&\quad \mathit{case\_\eta} : \{a, b, c\} \to (f : T\ (a + b) \to S\ c)\\
&\qquad\qquad \to \mathit{case}\ (\lambda x.\ f\ (\mathit{inl}\ x))\ (\lambda x.\ f\ (\mathit{inr}\ x)) = f\\
&\textsc{record}\ \mathit{coprod}\ (U : \mathbb{J})\ (T : U \to \mathbb{J}) : \mathbb{J}\ \textsc{where}\\
&\quad \mathit{cpintr} : \mathit{coprod\_intro}\ U\ T\\
&\quad \mathit{cpelim} : \mathit{coprod\_elim}\ U\ T\ U\ T\ \mathit{cpintr}
\end{aligned}
$$

We then instantiate with $U = el\ ty$ and $T = tm$ to get type-level coproducts:

$$coprodTy : coprod\ (el\ ty)\ tm$$

In this way, if kind-level coproducts are also needed, they can be easily added by a declaration $coprodKi : coprod\ ki\ el$.

* Kind-level lists with elimination to ML-style signatures

We first define the judgements of lists parameterised by the universe (U, T) that the lists live in and the universe (V, S) that the lists can eliminate into:

$$\begin{aligned}
&\textsc{record}\ ListAlg\ \{U : \mathbb{J}\}\ \{V : \mathbb{J}\}\ (T : U \to \mathbb{J})\ (S : V \to \mathbb{J}) \\
&\quad (k : U)\ (a : V) : \mathbb{J} \\
&\quad \textsc{where} \\
&\quad fst : S\ a \\
&\quad snd : T\ k \to S\ a \to S\ a \\
&\textsc{record}\ ListHom\ \{U : \mathbb{J}\}\ \{V : \mathbb{J}\}\ \{W : \mathbb{J}\} \\
&\quad \{T : U \to \mathbb{J}\}\ \{S : V \to \mathbb{J}\}\ \{R : W \to \mathbb{J}\} \\
&\quad \{k : U\}\ \{a : V\}\ \{b : W\} \\
&\quad (alga : ListAlg\ T\ S\ k\ a)\ (algb : ListAlg\ T\ R\ k\ b) : \mathbb{J} \\
&\quad \textsc{where} \\
&\quad f : S\ a \to R\ b \\
&\quad homnil : f\ (alga\ .fst) = algb\ .fst \\
&\quad homcons : (x : T\ k) \to (a : S\ a) \to f\ (alga\ .snd\ x\ a) = algb\ .snd\ x\ (f\ a) \\
&\textsc{record}\ ListIntro\ (U : \mathbb{J})\ (T : U \to \mathbb{J}) : \mathbb{J}\ \textsc{where} \\
&\quad listc : U \to U \\
&\quad listcalg : \{k : U\} \to ListAlg\ T\ T\ k\ (listc\ k) \\
&\quad nil : (k : U) \to T\ (listc\ k) \\
&\quad nil\ k = listcalg\ .fst \\
&\quad cons : \{k : U\} \to T\ k \to T\ (listc\ k) \to T\ (listc\ k) \\
&\quad cons\ x\ xs = listcalg\ .snd\ x\ xs \\
&\textsc{record}\ ListElim\ (U : \mathbb{J})\ (T : U \to \mathbb{J})\ (V : \mathbb{J})\ (S : V \to \mathbb{J}) \\
&\quad (intr : ListIntro\ U\ T) : \mathbb{J} \\
&\quad \textsc{where} \\
&\quad \textsc{open}\ ListIntro\ intr \\
&\quad fold : \{k : U\} \to \{a : V\} \to (alga : ListAlg\ T\ S\ k\ a) \\
&\qquad \to T\ (listc\ k) \to S\ a \\
&\quad foldβnil : \{k, a\} \to (alga : ListAlg\ T\ S\ k\ a) \\
&\qquad\quad \to fold\ alga\ (nil\ k) = alga\ .fst \\
&\quad foldβcons : \{k, a\} \to (alga : ListAlg\ T\ S\ k\ a) \\
&\qquad\quad \to (x : T\ k)\ (xs : T\ (listc\ k))
\end{aligned}$$

$$\to \mathit{fold}\ \mathit{alga}\ (\mathit{cons}\ x\ \mathit{xs}) = \mathit{alga}\ .\mathit{snd}\ x\ (\mathit{fold}\ \mathit{alga}\ \mathit{xs})$$

$$\mathit{fold}\eta : \{k, a\} \to (\mathit{alga} : \mathit{ListAlg}\ T\ S\ k\ a) \to (h : \mathit{ListHom}\ \mathit{listcalg}\ \mathit{alga}) \to \mathit{fold}\ \mathit{alga} = h\ .f$$

$$\textsc{record}\ \mathit{List}\ (U : \mathbb{J})\ (T : U \to \mathbb{J})\ (V : \mathbb{J})\ (S : V \to \mathbb{J}) : \mathbb{J}\ \textsc{where}$$
$$\mathit{intr} : \mathit{ListIntro}\ U\ T$$
$$\mathit{elim} : \mathit{ListElim}\ U\ T\ V\ S\ \mathit{intr}$$

We have kind-level lists with declarations

$$\mathit{ListKi} : \mathit{List}\ \mathit{ki}\ \mathit{el}\ \mathit{ki}\ \mathit{el}$$

We additionally have elimination of kind-level lists to ML-style signatures

$$\mathit{si} : \mathbb{J}$$
$$\mathit{si} = \Sigma\ \mathit{ki}\ (\lambda k.\ (\mathit{el}\ k \to \mathit{el}\ \mathit{ty}))$$
$$\mathit{mo} : \mathit{si} \to \mathbb{J}$$
$$\mathit{mo}\ (k, t) = \Sigma\ (\mathit{el}\ k)\ (\lambda \alpha.\ \mathit{tm}\ (t\ \alpha))$$
$$\mathit{ListKiTyElim} : \mathit{ListElim}\ \mathit{ki}\ \mathit{el}\ \mathit{si}\ \mathit{mo}\ (\mathit{ListKi}\ .\mathit{intr})$$

In the following we will rename the components of lists as follows:

$$\textsc{open}\ \mathit{List}\ \mathit{ListKi}\ \textsc{renaming}$$
$$(\mathit{listc} \mapsto \mathit{list}_k;\ \mathit{nil} \mapsto \mathit{nil}_k;\quad \mathit{cons} \mapsto \mathit{cons}_k;$$
$$\mathit{fold} \mapsto \mathit{fold}_k;\mathit{foldβnil} \mapsto \mathit{foldkβni};$$
$$\mathit{foldβcons} \mapsto \mathit{fold}_k\beta\mathit{cons};\mathit{foldη} \mapsto \mathit{fold}_k\eta)$$

$$\textsc{open}\ \mathit{ListElim}\ \mathit{ListKiTyElim}\ \textsc{renaming}$$
$$(\mathit{fold} \mapsto \mathit{fold}_{kt};\quad \mathit{foldβnil} \mapsto \mathit{fold}_{kt}\beta\mathit{nil};$$
$$\mathit{foldβcons} \mapsto \mathit{fold}_{kt}\beta\mathit{cons};\mathit{foldη} \mapsto \mathit{fold}_{kt}\eta)$$

* The constantly empty higher-order functor

$$\mathit{VoidH} : \mathit{RawHFunctor}$$
$$\mathit{VoidH} = \textsc{record}\ \{H_0 = \mathit{voidH0}\ ;\mathit{hfmap} = \mathit{hfmapVoid}\ ;\mathit{hmap} = \mathit{hmapVoid}\}$$
$$\textsc{where}$$
$$\mathit{voidH0} : \mathit{el}\ \mathit{htyco}$$
$$\mathit{voidH0}\ \_\ \_ = \mathit{empty}$$
$$\mathit{hfmapVoid} : (F : \mathit{RawFunctor}) \to \mathit{tm}\ (\mathit{fmap\text{-}ty}\ (\mathit{voidH}_0\ (F_0\ F)))$$
$$\mathit{hfmapVoid}\ F\ \alpha\ \beta\ f\ x = x$$
$$\mathit{hmapVoid} : (F\ G : \mathit{RawFunctor}) \to \mathit{tm}\ (\mathit{trans}\ (F_0\ F)\ (F_0\ G))$$

$$\to tm\ (nat - ty\ (voidH_0\ (F_0\ F))\ (voidH_0\ (F_0\ G)))$$
$$hmapVoid\ F\ G\ \_\ \alpha\ x = x$$

* Coproduct of higher-order functors

$$coprodHF : RawHFunctor \to RawHFunctor \to RawHFunctor$$
$$coprodHF\ H_1\ H_2\ .H_0 = \lambda F\ A.\ (H_1\ .H_0\ F\ A) + (H_2\ .H_0\ F\ A)$$
$$coprodHF\ H_1\ H_2\ .hfmap = \lambda F\ \alpha\ \beta\ f\ x.$$
$$case\ \{c = H_1\ .H_0\ (F_0\ F)\ \beta + H_2\ .H_0\ (F_0\ F)\ \beta\}$$
$$(\lambda l.\ inl\ (H_1\ .hfmap\ F\ \alpha\ \beta\ f\ l))$$
$$(\lambda r.\ inr\ (H_2\ .hfmap\ F\ \alpha\ \beta\ f\ r))$$
$$x$$
$$coprodHF\ H_1\ H_2\ .hmap = \lambda F\ G\ s\ \alpha\ x.$$
$$case\ \{c = H_1\ .H_0\ (F_0\ G)\ \alpha + H_2\ .H_0\ (F_0\ G)\ \alpha\}$$
$$(\lambda l.\ inl\ (H_1\ .hmap\ F\ G\ s\ \alpha\ l))$$
$$(\lambda r.\ inr\ (H_2\ .hmap\ F\ G\ s\ \alpha\ r))$$
$$x$$

* Higher-order functor for an algebraic operation

$$AlgOpHFun : el\ ty \to el\ ty \to RawHFunctor$$
$$AlgOpHFun\ P\ A =$$
$$\textsc{record}\ \{H_0 = \lambda\_\ X.\ P\ \times_t\ (A \Rightarrow_t X)$$
$$;hfmap = \lambda F\ \alpha\ \beta\ f\ (p, k).\ (p, (\lambda x.\ f\ (k\ x)))$$
$$;hmap = \lambda F\ G\ s\ \alpha\ pk.\ pk\}$$

* The ML-style signature corresponding of functors

$$FctSig : si$$
$$FctSig = tyco, fmap\text{-}ty$$
$$FctToMod : RawFunctor \to mo\ FctSig$$
$$FctToMod\ F = (F_0\ F), (F_1\ F)$$
$$FctFromMod : mo\ FctSig \to RawFunctor$$
$$FctFromMod\ (F, fmap) = \textsc{record}\ \{F_0 = F; F_1 = fmap\}$$

* The ML-style signature corresponding of higher-order functors

$$HFctSig : si$$
$$HFctSig = htyco, (\lambda H.\ hfmapTy\ H\ \times_t\ hmapTy\ H)\ \textsc{where}$$
$$hfmapTy : (H : el\ htyco) \to el\ ty$$
$$hfmapTy\ H = All\ tyco\ (\lambda F.\ fmap\text{-}ty\ F \Rightarrow_t fmap\text{-}ty\ (H\ F))$$
$$hmapTy : (H : el\ htyco) \to el\ ty$$

$$
\begin{array}{l}
\quad hmapTy\ H = All\ tyco\ (\lambda F.\ fmap\text{-}ty\ F \Rightarrow_t \\
\qquad All\ tyco\ (\lambda G.\ fmap\text{-}ty\ G \Rightarrow_t \\
\qquad\qquad (trans\ F\ G \Rightarrow_t nat\_ty\ (H\ F)\ (H\ G))))
\end{array}
$$

$$
\begin{array}{l}
HFctToMod : RawHFunctor \rightarrow mo\ HFctSig \\
HFctToMod\ H = (H\ .H_0), \\
\quad ((\lambda F\ fmap.\ H\ .hfmap\ (FctFromMod\ (F, fmap))) \\
\quad , \lambda F\ fmap_1\ G\ fmap_2. \\
\qquad H\ .hmap\ (FctFromMod\ (F, fmap_1)) \\
\qquad\qquad (FctFromMod\ (G, fmap_2)))
\end{array}
$$

$$
\begin{array}{l}
HFctFromMod : mo\ HFctSig \rightarrow RawHFunctor \\
HFctFromMod\ (H, (hfmap, hmap)) = \\
\quad \textsc{record}\ \{H_0 = H \\
\qquad\qquad ; hfmap = \lambda F.\ hfmap\ (F\ .F_0)\ (F\ .F_1) \\
\qquad\qquad ; hmap = \lambda F\ G.\ hmap\ (F\ .F_0)\ (F\ .F_1)\ (G\ .F_0)\ (G\ .F_1)\}
\end{array}
$$

* The family of algebraic operations

$$
\begin{array}{l}
AlgSig : el\ (list_k\ (ty \Rightarrow_k ty)) \rightarrow RawHFunctor \\
AlgSig\ es = HFctFromMod \\
\quad (fold_{kt}\ \{a = HFctSig\} \\
\qquad (\textsc{record}\ \{fst = HFctToMod\ VoidH \\
\qquad\qquad\qquad ; snd = \lambda(P, A)\ H. \\
\qquad\qquad\qquad\quad HFctToMod\ (coprodHF\ (AlgOpHFun\ P\ A) \\
\qquad\qquad\qquad\qquad\qquad\qquad (HFctFromMod\ H))\}) \\
\qquad es)
\end{array}
$$

$$
\begin{array}{l}
ListApp_k : \{k : ki\} \rightarrow el\ (list_k\ k) \rightarrow el\ (list_k\ k) \rightarrow el\ (list_k\ k) \\
ListApp_k\ \{k\}\ x\ y = \\
\quad fold_k\ \{a = list_k\ k\} \\
\qquad (\textsc{record}\ \{fst = y; snd = cons_k\})\ x
\end{array}
$$

$$
\begin{array}{l}
algFam : Fam \\
algFam = \textsc{record}\ \{eff = list_k\ (ty\ \times_k\ ty) \\
\qquad\qquad\qquad\qquad ; sig = AlgSig \\
\qquad\qquad\qquad\qquad ; add = ListApp_k\}
\end{array}
$$